University of Fort Hare

Together in Excellence

Implementation of a Facebook Crawler for Opinion Monitoring and Trend Analysis Purposes: A Case Study of Government Service Delivery in Dwesa

A thesis submitted in fulfillment of the requirements of the degree of

Masters of Science

In

Computer Science

By

Mfenyana Sinesihle Ignetious

Acknowledgements

I would like to express my sincere gratitude to my supervisors Ms. Nyalleng Moroosi and Prof. Mamello Thinyane for their continuous support during the time of my research. They were patient, motivational and imparted immense knowledge to me. Their invaluable guidance was of greater significance during the processes of carrying out my research and the writing of this thesis. I could not have imagined having better supervisors and mentors for my research.

I would also like to thank the University of Fort Hare Computer Science Department and its Head of Department, Mr. S.M Scott for allowing me the privilege of pursuing the Masters of Computer Science degree in their department.

Further, I would like to thank Telkom South Africa, for sponsoring my undergraduate and postgraduate studies. Without the financial support from Telkom South Africa, I would not have reached this level in my studies.

Special mention also goes to the Telkom Centres of Excellence for their support during the process of conducting the research project by providing equipment used in this research and transport when visiting the research site (Dwesa community).

A vote of thanks also goes to my lab mates for their valuable suggestions and ideas at a time when the research had become extremely tough and confusing.

I would also like to thank my mother, Ms. Xoliswa Mfenyana, for her motivational words during the challenging phases of my research.

Above all, I would like to thank the Lord God Almighty for the blessings and wisdom He bestowed upon me.

Declaration

I, Mfenyana Sinesihle Ignetious, student number 200807365, hereby declare that this dissertation is my own original work. All the information extracted from other sources is acknowledged as such.

Signature: _____

Date: _____

Publications

Some parts of this research were published during the course this research. The following are the two papers which were published in relation to this research:

Sinesihle I Mfenyana, Nyalleng Moroosi, 2012: Development of Social Networking Sites (SNSs) Trend Analysis Tool. SATNAC conference held from 2 to 5 September in 2012, at Fancourt in George, Western Cape, South Africa.

S.I. Mfenyana, N. Moroosi, M Thinyane and S.M. Scott: Development of a Facebook Crawler for Opinion Trend Monitoring and Analysis Purposes: Case Study of Government Service Delivery in Dwesa. *International Journal of Computer Applications* 79(17):32-39, October 2013. It was published by the Foundation of Computer Science, New York, USA.

Abstract

The Internet has shifted from the Web 1.0 era to the Web 2.0 era. In the contemporary era of web 2.0, the Internet is being used to build and reflect social relationships among people who share similar interests and activities. This is done through services such as Social Networking Sites (Facebook, Twitter etc.) and the web blogs. Currently, there is a very high usage of Social Networking Sites (SNSs) and blogs where people share their views, opinions, and thoughts. This leads to the production of a lot of data by people who post such content on SNSs. As a result, SNSs and blogs become the ideal platforms for opinion monitoring and the trend analysis. These SNSs and Blogs could be used by service providers for tracking what the public thinks or requires. The reason being, having such knowledge can help in decision making and future planning. If service providers can keep track of such views, opinions or thoughts with regard to the services they provide, they can better their understanding about the public or clients' needs and improve the provision of relevant services. This research project presents a system prototype for performing opinion monitoring and trend analysis on Facebook. The proposed system crawl Facebook, indexes the data and provides user interface (UI) where end users can search and see the trending of a topics of their choice. The system prototype could also be used to check the trending topics without having to search. The main objective of this research project was to develop a framework that will contribute in improving the way government officials, companies or any service providers and normal citizens communicate regarding services they provide. This research project is premised on the conceptualization that if the government officials, companies or any service providers can keep track of the citizen's opinions, views and thoughts with regards to services they provide it can help improve the delivery of such services. This research and the implementation of the trend analysis tool is undertaken in the context of the Siyakhula Living Lab (SLL), an Information and Communication Technologies for Development (ICTD) intervention for Dwesa marginalized community.

Keywords: Social networking Sites, Focused Web Crawlers, Information Retrieval, Facebook, ICTD, SLL.

Acronyms and Abbreviations

API	Application Programming Interface
End-User	Person who actually uses a particular product
GUI	Graphic User Interface
HTML	Hyper Text Markup Language
ICT	Information and Communications Technology
ICTD	Information and Communication Technology for Development
IDE	Integrated Development Environment
JSP	Java Server Page
JSS	Junior Secondary School
OWL	Ontology Web Language
OWLIR	Ontology Web Language and Information Retrieval
SLL	Siyakhula Living Lab
SSS	Senior Secondary School
SUMO	Suggested Upper Merged Ontology
System-Administrator	Person who is responsible for upkeep, configuration, and reliable operation of the system
UI	User Interface
URL	Uniform Resource Locator
WWW	World Wide Web
JSON	JavaScript Object Notation

Table of Figures

Contents

Chapter One – Research Introduction

Introduction

Developing countries like South Africa have adopted the use of Information and Communication Technologies (ICTs) to support rural development. This is usually termed ICT for Development (ICTD) and has seen many initiatives being undertaken towards achieving the socio-economic goals for the development of rural communities. One such initiative is the Siyakhula Living Lab (SLL). This initiative aims to improve social and economic situations in rural areas [1]. This research proposes the development of a system which can collect data from populations in disparate geographic and socio-economic using Social Networking Sites (SNSs) from which opinion monitoring and trend analysis will be performed. While a system such as the one we propose has a wide applicability we will look at it in the space of rural communities using it to communicate to government agencies.

SNSs are used by people to connect with each other for business or personal purposes. They build and reflect social relationships among people who share similar interests and activities [2][3][4]. These SNSs are also used by many businesses and governments to inform, engage and serve citizens. In government, this is done through government 2.0, which is the usage of social media tools to facilitate the conversations between governments institutions and citizens to shape polices, support local government democracy and improve services [5].

These free and easy to use cloud based applications provide individuals, organizations and societies an easy and cheaper way of communication [6]. SNSs' services provide content sharing through the publication of documents and links and also through messages exchanged using communication tools. These functionalities allow for the use of SNSs as a collaborative opinion mining platform. There are a lot of factors motivating for the use of SNSs for opinion trend analysis. Firstly, SNSs are easy to use and provide a cheap way of communication as evidenced by a growth in their usage [7]. Additionally, most SNSs data can easily be accessed by the use of supported Application Programming Interface (APIs). These are commonly developed by the SNS development team.

The applicability of the proposed system is wide-ranging; however in this research the focus is on improving service delivery in marginalized rural communities, which form the field site of this research. In this research we mine opinions from constituents with a special interest in monitoring trends about service delivery in marginalized rural communities of South Africa. The proposed system seeks to improve or optimize consultation principle, one of the eight Batho Pele Principles initiatives. Batho Pele (Sesotho word, meaning "People First" in English) was launched in 1997 in South Africa because the preceding public service system was believed not to be people-friendly and lacked the skills and attitudes to meet the developmental challenges facing the country. The idea of this initiative was the need to transform public service at all levels [8]. The consultation principle refers to the process of interacting with, listening to and learning from the people whom you serve by staying in touch with them so that you can find out what services they need, how they would like their services to be delivered and what they are not satisfied with [8]. The following are factors that affect service delivery in rural communities:

Access - long distances and poor road infrastructures which separate rural areas and developed regions of the country, leave populations in rural areas poorly integrated in their economies and polities.

Cost - poor road infrastructures separating rural areas and developed regions of the country increases the costs of goods deliveries and that discourages providers of goods and services from reaching the rural areas resulting in a situation where rural areas are left under-developed.

Quality - infrastructures (e.g. road, telecommunications, education, and health care centers) are poor in most rural areas of South Africa [9]. These usually chase away professional and skilled people who would bring development in such areas.

The SLL, where this research is being undertaken, is an ICTD initiative which was developed to provide ICT solutions for most of the problems being faced by the Dwesa community through e-Commerce portals, e-Learning solutions, e-Government services and e-Health applications [9]. Dwesa is a coastal region located in the former homeland of the Transkei in the Eastern Cape, South Africa. Residents of Dwesa have to travel more than 40km to visit different government offices at the Willowvale Business Centre and approximately 80km to the Idutywa Business Centre to submit their complaints or access public services [10]. Taking into account the fact that

rural areas of South Africa consist largely of unemployed people, the costs of reaching municipal offices can be prohibitively expensive. A classical example was observed during the visits at Dwesa community for computer literacy trainings which were conducted by the SLL team every month during the course of this research. Some community residents were traveling approximately more than 5km to 10km to reach schools where the computer literacy trainings where usually held. Some of the residents would not have transport for traveling to those schools where the trainings where normally held. That in turn would result in situation where some of the residents would be left out from the service which was being provided. That also suggests that some community residents do not fully participate in the community activities such as ward meetings, budget meetings etc. which are commonly held at the same schools.

The proposed system prototype will provide a cost effective solution for communication between companies, government officials, or organizations which seek to find out about the public or clients' views, opinions, ideas etc. about the services they provide. In this research project, the proposed system will be used to enable the local government in Dwesa (and other communities in South Africa) to determine the trending topics and discussions that are related to service delivery within their jurisdiction. This is achieved through the implementation of a focused Facebook crawler and opinion trend analysis tool. Focused crawler is a topic-driven web crawler which selectively retrieves relevant web pages to a predefined set of queries or topics [11]. The crawler extracts statuses and comments feeds which are then used for opinion monitoring and trend analysis purposes.

Proposed System Overview

Figure 1**,** gives the proposed system overview. It is followed by a discussion of how data flow will occur inside the system and the functionality of modules on the proposed system.

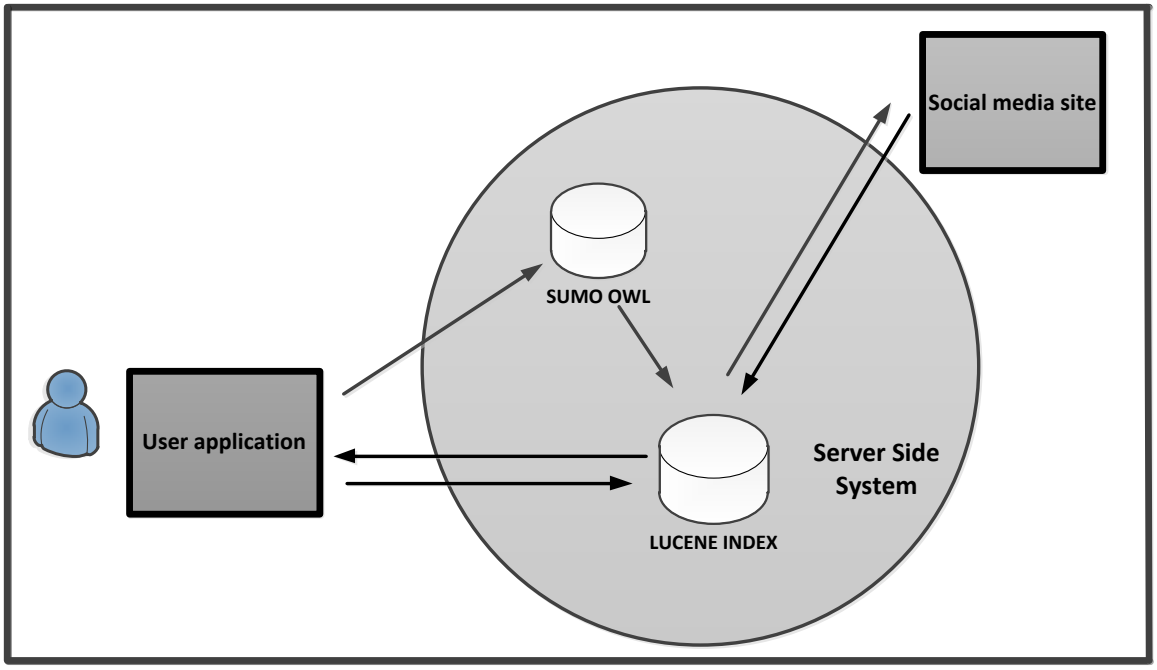

Figure 1: High level system overview

The proposed system will consist of two parts; a user graphic interface which will be responsible for facilitating user query searches, data visualization and data display, and a server-side system. The Second is the server-side, which will perform most of the proposed system functionalities such as data gathering from Facebook, text parsing, prepossessing, indexing and handling user queries by performing the logic of the system.

The server-side system will consist of a back-end server, ontology and index, and front-end server. The back-end server will be responsible for crawling SNS gathering data, parsing text and indexing the text. The front-end server will be responsible for handling the communication between user graphical interface and the index by providing the logical handling of queries and performing the content matching, correlation analysis, trend analysis, keyword searching from the indexed textual data and gives the feedback to end-user queries. The detailed system is given in Chapter 3.

Research Problem Statement

There are many challenges when it comes to communication between service providers and the people they are serving. It is always utmost important for the service providers to know the

sentiments of the public and/or clients towards the services or products they provide. The problem of communication between service providers and their clients usually arises because the clients could be widely geographically separated. That makes it very difficult for service providers to collect data which they can use to better understand their clients' needs. This research is aimed at providing a communication bridge between the consumers and the service providers.

The current government public service system has many drawbacks when it comes to its ability to satisfy the consultation principle. These include long distances, lack of transport and high travel costs. These are the main challenges when it comes to fully accomplishing the consultation principle between rural areas' residents and their local governments of South Africa[12]. Currently, local governments achieve the consultation principle through call-ups of ward committees, budget consultations, ward meetings, Integrated Development Planning Forums, among other issues, whenever they want people to participate in consultations and decision-making processes at a local level [13]. Due to the complexities and costs associated with the above mentioned solutions many people do not participate in consultative and decision-making processes. This is compounded by the fact that there is a high prevalence of unemployment amongst most rural residents in South Africa. A classical example is that of the Dwesa community where the SLL is located and is spread over a radius of 40km [14].

The current challenges faced in rural areas of South Africa, namely; poor or limited access to socio-economic infrastructure and services such as water resources, reliable road network, power supply, and telecommunication networks, have not been fully attended to by government officials because:

1. Local government is not aware of how bad the situation is for people residing in rural areas because they cannot express their views.
2. People in rural areas do not have good and cheap means of expressing/reporting their living situations to government officials due to the fact that there are no government infrastructure setups in those areas which collect their views and options. This was also observed at Dwesa community where this research was based at.

Project Aim and Objectives

The main aim of this research was to implement a focused Facebook crawler which will extract and analyze views and opinions of people from Facebook. In general, the proposed framework provides two high level functionalities: (a) text extraction from SNS (Facebook) and (b) data analysis and visualization, which consists of parsing text from data obtained from Facebook, preprocess, index, visualize and provides search functionality.

The proposed system is intended to be deployed at SLL and used by the residents of Dwesa community. The system after deployment will then be used by the local government officials in Dwesa community to monitor trending topics about the services they provide. The specific objectives for this research project were to:

1. Investigate the feasibility of an electronic sentiment monitoring tool for rural area residents. Specifically, this is to learn if the use of SNSs is prevalent enough in rural environments that an SNS sentiment mining tool can be used to track views and opinion of marginalized communities.
2. Evaluate different crawling policies and select one we believe to be suitable for our research.
3. Investigate and identify the key requirements for a Facebook Crawler that can be used for opinion monitoring and trend analysis purposes.
4. Design, develop, implement and test a Facebook Crawler for Opinion Monitoring and Trend Analysis Purposes.
5. Study the feasibility of sentiment analysis and general Data Mining in multi-lingual/non-English Speaking environments.
6. Investigate general problems faced by rural communities when communicating with their local government.
7. Investigate the methods they are currently using to communicate with their government.

Thesis Outline

Chapter One: [Introduction to the research project] this chapter provides an introduction to the research project. It sets out the research aims and objectives, research problem statement, projected system overview and provides the thesis outline.

Chapter Two: [Literature Review] this chapter sets out the literature review. It gives a necessary research background by exploring the theoretical framework related to our research domain. It then analyses the related work which has been done under our research domain.

Chapter Three: [Research Methodology, System Design and Core Used Technologies] this chapter discusses the research methodology used in this research. It also discusses the functional and non-functional system attributes. The chapter also gives the low and high level system design.

Chapter four: [System Implementation] this chapter discusses the implementation details of the developed system in this research.

Chapter Five: [System Testing, Experimental Results and Future Work] this chapter discusses the system testing, experimental results and concludes by giving the possible extension suggestion which could in future be added to our developed system.

Conclusion

This chapter introduced the research overview and also presented the proposed system overview. In addition the problem statement and the project aim as well as the specific of objectives were outlined. The chapter concludes by giving the thesis organization.

Chapter Two: Literature Review and Background Study

Introduction

This chapter gives the background to the study. It will help readers to understand our proposed research domain which is social web mining, with applications in opinion monitoring and trend analysis from SNSs.

Social Networking Sites (SNSs)

In literature there are many definitions of social networking sites. Two of these were chosen for the purposes of this research project. The first one was advanced by [15] where they defined SNSs as networks which allow individuals to: (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system. The second definition was advanced in 2007 [16] and also provides a useful definition of SNS. In this definition, SNSs were defined as networks made of a set of nodes and a set of ties. Nodes refer to the individuals, organizations or societies, while ties represent a particular type of relationship between the nodes such as friendship, family and co-workers among others.

SNSs are ideal examples of Web 2.0, not only because of their social networking aspects which include the user as a first class object, but also due to their use of new user UI technologies. Web 2.0 is an umbrella term that is used to represent Web sites which incorporate a strong social component; involving user profiles, friend links or Web sites which encourage user generated content in the form of; text, video, and photo postings along with comments, tags, and ratings. Web 2.0 may also include Web sites that have gained popularity in recent years and are subject to fevered speculations about valuations and initial public offering prospects [17].

Popular SNSs, their Brief History and their Statistical Usage

Currently there are hundreds of SNSs in use and among them Facebook, Twitter and LinkedIn are the top three popular SNSs. Facebook has 750,000,000 estimated unique monthly visitors and is ranked second on the compete rank. Twitter has 250,000,000 estimated unique monthly visitors and is ranked twenty four on compete rank and LinkedIn has 110,000,000 estimated

unique monthly visitors and is ranked forty four on the compete rank [18]. All of these SNSs support different technological affordances, wide range of interests and practices [15].

Facebook was launched by Mark Zuckerberg in 2004. Initially, Facebook was only available to Harvard university students. Today, anybody who is 13 years or older and has a valid E-mail address can register to Facebook [19]. Facebook, like any other SNS, could be regarded as the message board which serves as the primary asynchronous messaging mechanism between friends [20], by controlling the posting of statuses, comments, uploading of photos etc. by the Facebook users. Facebook servers and organizes the content posted on its servers in a chronologically order with the date and time stamped. The most recently posted content is put at the top of the Facebook page. Facebook also tags the posted content to the Facebook users who posted the actual content through use of user IDs (Identify Documents) and names of the facebook users who posted the actual contents. Facebook has more than 800 million active users and more than 50% of these users log on in any given day[21].

Twitter is a micro blogging service launched in 2006 by Jack Dorsey, Biz Stone, and Even Williams and currently records over 140 million active users that generate over 340 million tweets per day [22]. Twitter is currently the most popular on line micro blogging service, which enables its users to send and receive text-based posts, known as "*tweets*" [22]. Each message sent through twitter cannot be more than 140 characters in length. That is the feature which differentiates it from traditional web blogging because the content that can be posted is limited. The Micro blogging service has a strict constraint in content size unlike traditional blogging services[23]. This idea originated from Short Message Service (SMS) technology over mobile devices such as Smartphones where a user can use SMS to send a message to twitter, and the message will be forwarded to many other Twitter users that are related or connected to the sender. Twitter was built to share thoughts or ideas instantly. However, major Television (TV) networks and news gradually started to use Twitter to provide updated news to the public. Twitter is now a media platform in which ordinary users can be in the front line of breaking news [19].

LinkedIn was launched in 2003 and currently is the world's largest professional network with over 175 million members and growing rapidly [24][25]. LinkedIn is used by people to build professional networks and find career development opportunities. Employers also use this SNS

to look into the profile information of users to search for potential employees. LinkedIn helps people to advertise themselves and it also helps employers to spot potential candidates for the job posts they have [26].

The infographic below shows the sizes as well as the rate of growth of the most popular SNSs. This data was prepared by the GlobalWebIndex study, which prepared data on web usability statistics. In this research we did not look at Google+ and Pinterest for study as they are not as widely used in South Africa as they are in the United States.

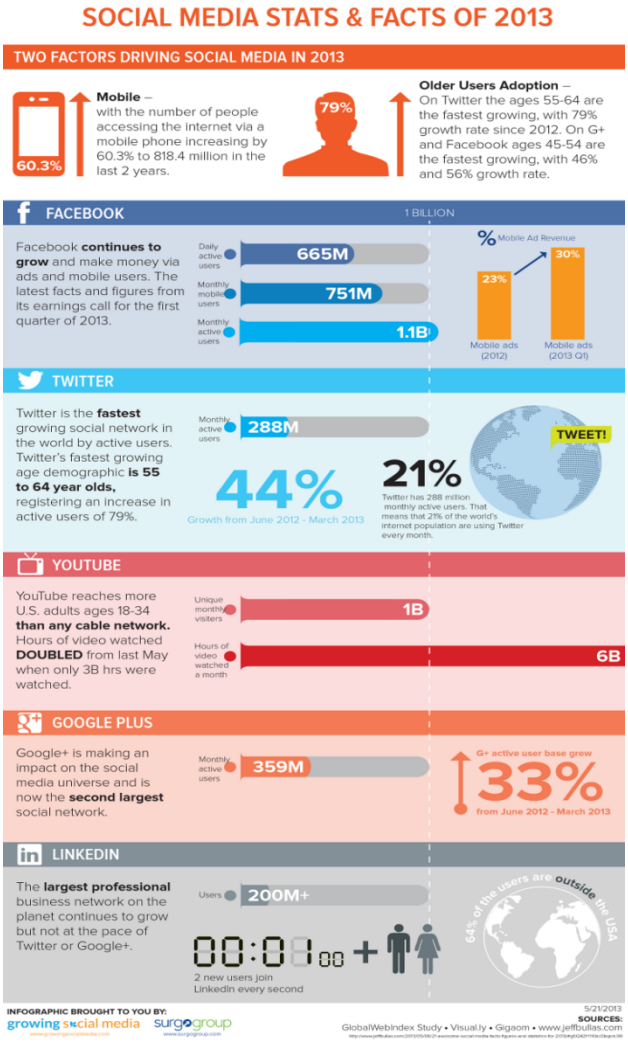

Figure 2: Social Media Stats and Facts of 2013

As indicated by the infographic, while the rate of growth of Twitter and Google+ is much higher than that of Facebook, the monthly Facebook users far-outstrip Twitter and Google+ users.

Additionally, Facebook is much older than both Tweeter and Google+ and thus we believe that our targets users, rural marginalized, will most probably know of Facebook over these other two.

YouTube is another SNS that has very high usability. However, given the cost of data in South Africa, we doubt that a lot of our users have the means to be very regular users of YouTube. While they may occasionally use YouTube, for our purposes we need a highly utilized system so that we can get enough data for statistical significance. With all this in mind, we decided on Facebook as the best SNS to crawl.

Web Crawlers

A web crawler is a computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion [27]. Web crawlers are used by search engines to create a copy of all the visited pages, which they save on a single directory and subsequently indexes to provide fast searches. Web crawlers are also used for automating maintenance tasks on web sites, such as checking links or validating Hyper Text Markup Language (HTML) code. Also, they are used along with data extractors to collect structured knowledge in different fields by processing unstructured textual data automatically [28]. Web crawlers can also be used the same way as they are used for traditional websites for gathering data on SNSs but, when they are employed for such tasks they have to go through SNSs' APIs designated for those SNSs[29].

In our research Facebook Graph API was used in the development of our crawler. Graph API is a simple HTTP based API that gives access to the Facebook social graph. It acts as the interface which application developers can use to access Facebook database servers [30]. In Facebook database servers, Facebook users are uniquely identified by IDs. To access users' data, their user account IDs are used. The way of accessing data on different SNSs is determined by the APIs of those specific SNSs. This is because SNSs' APIs supports different types of data accessing mechanisms; some users allow their data to be accessed through queries. In that case, SNS provides searchable interface, either by providing an API or HTML form as the way of accepting queries and giving feedbacks to the end user. Some SNSs' APIs allow their data to be accessed more or less the same way as traditional crawling methods, where links that are found from seed HTML are extracted and added to the to-be visited list which is then used for crawling such a specific SNS [29].

Crawling Policies

SNSs are similar to the World Wide Web (WWW) which is a huge collection of web pages linked to each other by hyperlinks and keeps evolving every second, where a new piece of information is added [31]. Given its dynamic nature, the WWW, crawler, for it to be more effective and more efficient, should take into consideration the four crawling policies which determine the behavior of any web crawler [28]. These polices optimize the work follow of any web crawler by taking time, relevance of feedback and network resources into consideration. The following are the four crawling policies:

Selection policy - Web crawler should be able to download or crawl a fraction of the Web due to the size of the Web, because trying to crawl the entire Web will require a lot of bandwidth and bandwidth is a limited resource[28]. In short, this police states which pages to be downloaded or crawled from the Web [27].

Revisit policy – crawling the web can take really long time and the by the time the crawling process finishes there is high possible that there could be some events that could have occurred, such creation, updating and deletion of the web content[28]. This police states when a crawler should check for changes to the web content[27].

Politeness policy – according to [27], this police specify how web crawlers should behave in order for them not to overload web sites and/or database servers.

Parallelization policy - A parallel crawler is a crawler that runs multiple processes in parallel with the goal of maximizing the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page [32].

General Crawler Architecture

In this section we discuss the general activity flow in the web crawling process. This will help to give a clear overview of how a web crawler functions. In its simplest form, a crawler starts with one or few Uniform Resource Locator(s) (URL) to reach specific web pages. From there, it uses the links from such pages to reach other pages [33]. In general, every crawler consists of the following:

Frontier - the list of seed URLs or user account IDs in SNS such as Facebook, populated by a user or any other program such as non-visited nodes. This list is updated as the crawling process is occurring by adding newly found URLs to the to-be visited list.

Crawling loop – this is the stage where most of the crawling process occurs. When web crawling initiates, the web crawler starts by selecting the next URL(s) or user account ID from the to-be visited from the frontier. It then fetches the corresponding pages through a Hypertext Transfer Protocol (HTTP) request and extracts relevant information from the web page. If new URLs or user account IDs are discovered, they are added to the to-be visited list of URLs in the frontier[33].

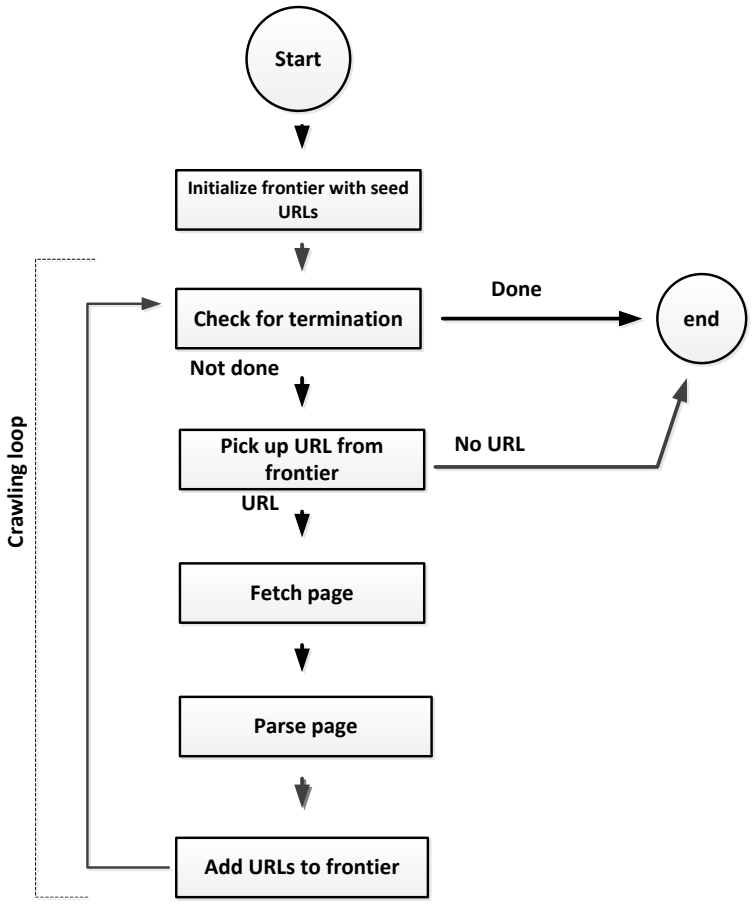

Figure 3: Crawler Overview [33]

Referring to Figure 3, the following are the processes which occur during the crawling process in their preceding order:

1. Initialize frontier: with seed URLs, the frontier is the to-be visited list of a crawler that contains the URLs of unvisited pages.
2. Check for termination: Checks if the frontier is empty. If not, it returns the next URI to be analyzed.
3. Fetch page: Obtains the web page through an HTTP client request.
4. Parse page: Creates a parser on the page to extract useful information such as URLs or the wanted content and possibly guide the next step of the crawler, and also eliminate irrelevant information.

The crawler we developed is initialized by providing it with the ID of Facebook user which his/her friends are to be crawled. The crawler starts buffering all the IDs of user's friends, from there, it then extracts first user account ID then crawl the latest data feed when it is done move to the next user stops when last friend on the list is crawled.

Crawling Algorithms

The web as mentioned is a huge collection of web pages which evolves every second, it is important to select a good crawling algorithm that crawls only a fraction of the web not the entire web. Even large search engines do not crawl and index the entire web [34]. The Facebook statics usage given above of active SNSs user (Facebook) suggest that it is also impossible and ineffective to crawler the entire SNS graph, there is need of choosing crawling algorithm which will be able to determine the fraction of SNS graph that is most relevant to a user's needs. In this section, we will give a brief overview of two main algorithms which already exist on this field of web data mining; general purpose crawlers and focused crawlers.

General Purpose Crawlers

This is an algorithm intended to crawl as many as possible web resources. This algorithm is used to traverse the web structure as complete as possible in order to find all the possible targets web database servers reachable from the given starting set of URLs through the outgoing links. General-purpose crawlers collect pages and/or web resources covering different topics. This algorithm is used by traditional search engines, for collecting and indexing as many web resources as possible. This algorithm allows traditional search engines to be able to retrieve pages that are relevant to every user's queries. When a user submits the query, the engine will

give feedback as an ordered list of pages ranked according to a particular matching algorithm [35]. Users can submit their queries in the form of a text, describing the information they are requesting for, the underlying Information Retrieval Model will then find the documents or web resources of interest which will help users to meet their information needs and make it easier for them to accomplish their information seeking activities.

This algorithm is used by search engines when they try to build up web graphs as complete as possible, which they use to give feedbacks to most of users' queries or topics [36]. Unlike focused crawlers, the feedback of general purpose crawler is not personalized. Different users with different interests, knowledge and preferences, if they all submit the same query, will obtain same results [35]. This algorithm cannot be used if the intention is to obtain information from the web database servers for a specific domain.

Focused Crawlers

According to [11] a focused crawler is a topic-driven web crawler which selectively retrieves relevant web pages to a predefined set of queries or topics. The advantages of focused crawlers are that they require minimum storage space, can employ techniques to crawl part of the deep Web that General Purpose Crawlers cannot reach, update frequently to keep stored information up to date, gives accurate results and utilizes minimum time as well as bandwidth to download pages [31][35] . They can be used for many applications such as public sentiment analysis, keeping track of public opinions about certain market products [37], keep track java code repositories [38] etc. On this research focused crawler will be employed for Facebook data feed collects which will then be used for trend analysis purposes.

Information Retrieval

According to [39] Information retrieval (IR) is the process of finding material (usually documents) of unstructured nature (usually text) that satisfies information from within collections (usually stored on computers) [39]. The actual goal of IR systems is to search and retrieve the most relevant documents to the user's information needs.

The General IR Processes

Figure 4 shows the general processes of IR system, squared boxes representing data and rounded boxes representing processes [40]. According to [41] any information retrieval should support

three basic processes which are content representation of documents, user's information need representation and the comparison of the two representations to give feedback to the end user. IR system for it to accomplish the above mentioned processes should also have these aspects UI, query processing, searching and ranking [42].

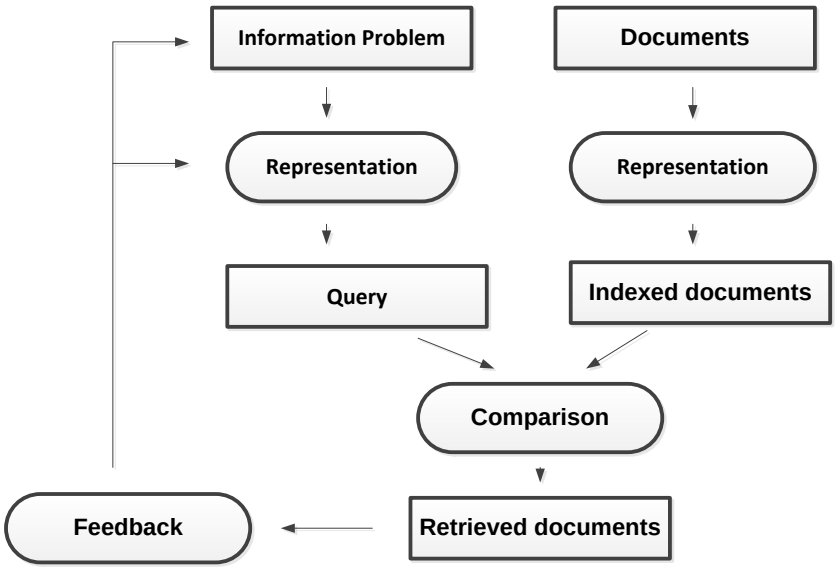

Figure 4:The General IR Processes [40].

User Interface

The IR system used in this thesis is Lucene java search library (the high-performance, scalable IR library [43]) and it does not provide UI. The UI is the most important aspect of any IR system as its design brings the tradeoff between user-friendliness and performance of any IR system [44]. In literature there are many different kinds of commonly used UI methods such as keyword-based, natural language-based, form-based and graph based interfaces. In this thesis keyword-based and graph-based methods were used. In Figure 4**"**feedback" part represents user UI as it accepts user queries and gives back feedback to the users. Further discussions will be done in the design and implementation chapters as the display and visualization of data matters the most in this thesis.

Query Processing

Query processing is the process of transforming a query into an internal form that can be interpreted by the system. Lucene achieves this by the use of its already developed analyzers to convert user query text into terms that can be used for searching the index. During this process text user query undergoes several processes such as stop-word elimination, stemming and other

application specific tasks[43]. In Figure 4, query processing is the Information Problem presentation. Lucene StandardAnalyzer was used for query processing. Further discussion on its usage will be given on the implementation chapter.

Searching

In Figure 4 is the phase where Query and the Indexed documents comparison occur. In general this is the phase where query terms are searched against the inverted index and retrieve all the documents that contain the occurrences of the query terms. There are different methods of searching the index but in this thesis two searching methods, Lucene keyword search and MatchAllDocsQuery search were used.

Indexing

Lucene only indexes data available in textual format. The input text data in Lucene is stored in an inverted index data structure, which is stored on file system or in memory as a set of index files[43]. Figure 4 illustrates the documents representation process. Documents indexing optimize the query performance and improve the response times to user queries. The index was developed in this thesis and its development details will be given in the implantation chapter.

Ranking

Ranking is the process of assigning scores to search results according to the matching quality between the query terms and the documents [41]. The documents are then presented to the user in a descending order. Documents with high scores are regarded as the most relevant documents presented to the user on top of the retrieval documents list. To rank the search results and give back the most relevant documents based from the scores, Lucene uses a combination of Vector Space Model (VSM) and the Boolean model [45]. The VSM is used to calculate the number of times the query term appears in a document relative to the number of times the query term appears in all the documents in a collection with the document containing a large number of the query term regarded as the most relevant document. The Boolean model is used to narrow down the documents to be scored, by selecting the matching documents according to user query using Boolean logic [41][45]. Basically, the IR model used by Lucene is VSM with Boolean capabilities.

Opinion Tracking and Trend analysis

SNSs' data can be a valuable source of data for opinion trend analysis if mined and filtered properly. This data can be used for many purposes such as customer relationship management, public opinion tracking about politics and sports. Through these blogs and SNSs, subjective information is generated by people expressing their views and opinions based on various topics or subjects.

A good example of subjective information collection can be found on blogs on "Yahoo news" which amongst other topics, focuses on various aspects of political covering the entire spectrum of politics related issues [46]. In this arena normal citizens express their opinions on everyday political issues, politicians communicate their ideas, journalists criticizing the government and all this data is open to all. Such data can be gathered through systems which can automatically track opinions of the public enabling decision makers to make sense of the enormous body of opinions expressed on the SNSs and traditional blogs.

The process of tracking most emerging topics or events that attract the attention of a large fraction of Blogs or SNSs users is called trend analysis [47]. Trend analysis by definition is the analysis of changes over time. In this research we are mainly focusing on opinion trend analysis, we proposed to develop a system which will help government officials to see most trending government service delivery related topics on Facebook.

Trend Detection from Text

Pranav [48] classifies trend detection methods which are currently being used, from a text data, as fully-automatic and semi-automatic.

Fully-automatic these are the systems that accept the input collection of textual data and generate a list of emerging topics. These systems provide the graphical visualization which helps the end user to examine the actual emerging trends based on the evidence provided by the system[48].

Semi-automatic these are the systems that require a user input such as a topic and then outputs the evidence that helps in determining whether that topic is emerging or not. Such systems provide a descriptive report of the available evidence[48].

The two methods are used in this thesis and their usages are described in Chapter 3 and Chapter 4 when discussing the system design and system implementation.

Related work

In this section, we give a review of prior work on crawling social networking sites, web mining and sentiment or opinion analysis. In literature most research papers on SNS are based on SNSs structure analysis and sentiment analysis. We start by giving the background SNSs data mining, how it started and progressed till today as it has attracted many researchers form different fields.

In 2012 [49], the editors of Special Issue on a Decade of Mining the Web, provided a brief overview of how Web mining evolved from the first Web mining workshop (WEBKDD'99). The workshop took place at the KDD'99, with the theme International Conference on Discovery and Data mining. They mentioned that during that time web data mining challenges and opportunities were very different compared to contemporary ones. Social Web by that time did not exist, Semantic Web was emerging. The major aim of web mining was to understand what users want and to help them to perform simple tasks inside web sites.

In 2008 in the 10th WEBKDD workshop the series of core web mining were closed because they reached maturity. From there, the research domain of web mining shifted to topics of knowledge discovery such as recommendation engines, model adaptation for user profiling, understanding communities and monitoring their evolution, modeling and interpreting user searches [49]. In 2102**,** Boldrini et al did their research with the aim of creating EmotiBlog, a fine grained annotation scheme for labeling subjectivity data in non-traditional textual genres [50]. Their research was motivated by massive data which is available on Web 2.0, which is made available by people through the use of new communication tools such as blogs, forums and reviews which people from all over the world employ as a source of information in addition to traditional textual genres such as newspaper articles[50]. They argued that it can be difficult to identify subjective data because of the mixture of text styles, a wide range of topics and sources, multiple languages, grammar and spelling mistakes, informal language, use of colloquialisms or slang, extensive amounts of data, continuous updating of information.

The other part of their research was to develop technologies to organize textual information, not just in terms of topical content, but also taking into account the emotions and opinions

embedded, as well as the source of the discourse. They confined their scope of research by only focusing on three languages (English, Spanish and Italian) and three topics. The main objectives of their research were to: design a fine-grained annotation schema able to capture the linguistic means of affective expression in non-traditional textual genres, annotate a collection of blog posts using the resulting schema, and evaluate the robustness of the scheme creating Machine Learning models using the annotated elements [50].

In 2011, Papadopoulos et al proposed a survey for community detection in the context of social media [51]. Their objective was to address two aspects which they felt are not addressed in the context of social media which are: performance aspects of community detection methods, namely computational complexity, memory requirements and possibility for incremental updates of already identified community structure, and interpretation and exploitation of community detection results by Social Media applications. They elaborated on methods for detecting and monitoring communities as the SNS evolves. They were focusing particularly on the issue of algorithm performance, but they also elaborated on different applications of community detection, such as user profiling, event detection and tag disambiguation.

In 2012, Tuzhilin did a research survey on Customer relationship management (CRM) and Web mining: the next frontier [52]. The argument of the research was that CRM can advance the series of Web mining field for the next decade even though CRM in mid-2000s had a period in which it was ineffective due to various reasons, including (but not limited to) high failure rates of various CRM projects, disillusionment with CRM systems, and disappointments with the results generated by various CRM applications in the industry. In 2012, Hagberg developed a news reading system [53]. The system was responsible for delivering interesting news to specified user based from the previous activities on Facebook. The system crawl Facebook for user information and subsequently retrieves relevant news for the specified user.

In 2007, Chau et al [54] proposed parallel crawling for online social networks. This was a continuation of the work by Cho et al [55] where they proposed three general architectures for parallel crawlers and the metrics which could be used to evaluate them. Those architectures were; Independent architecture, an architecture that has no coordination which exists among crawlers, Dynamic assignment architecture, architecture with a central coordinator, and the Static

assignment which is the architecture where the web is partitioned and assigned to each crawler, and the crawlers coordinate themselves once crawling has started.

Then Chau et al [54] proposed Dynamic assignment architecture which they established would be suitable to be used for crawling SNSs. On their architecture they enhanced the naive breath-first crawling strategy by making it parallel by moving the queue to a master machine and distribute crawling of web pages across agent machines which were distributed across network. Agents were responsible for the crawling of a user data on the network and return the feedback to the master machine. The master machine was responsible for maintaining global consistency of the queue and ensures that a user was crawled only once.

In 2011, Catanese et al [56] they did their research on Crawling Facebook for Social Network Analysis Purposes. They were collecting datasets about friendships of users in Facebook. They collected friend lists of friends and stored them for analysis where they were looking for connectivity between friends. They used two different SNS sampling algorithms Breath-first-search (BFS) and Uniform sampling. They evaluated the two algorithms but chose BFS because it was giving them good results. BFS includes an agent that executes data tasks and a First-In-First-Out (FIFO) queue.

BFS operates as follows, an agent contacts the Facebook server, providing credentials required for the authentication through cookies. Once logged in, the agent starts crawling pages, visiting the friend list page of the logged user and extracts friend list, adding friends' user-IDs to be visited FIFO queue and, cyclically, they were revisited in order to retrieve their friends lists. In 2011, Costa et al [7] proposed a framework for building web mining applications in the world of blogs and their case study was product sentiment analysis. They used BFS sampling on their system. Their work is closely related to our proposal the difference being that they were mining blogs not SNSs and their system was not for rural development in the sense that they assumed everyone can speak, read and write proper English.

In 2013, Kim et al [57] implemented a prototype system for detecting trend and bursty keywords from Twitter stream data. To detect trend and bursty keywords, it first selects candidate keywords from tweets by performing simple syntactic feature based filtering. And then, merge various keyword variants using several heuristics and select bursty keywords based on the term

frequency. By tracing the popularity transition of such trend keywords, it then determined bursty keywords. The following diagram Figure 5, is their system prototype overall architecture.

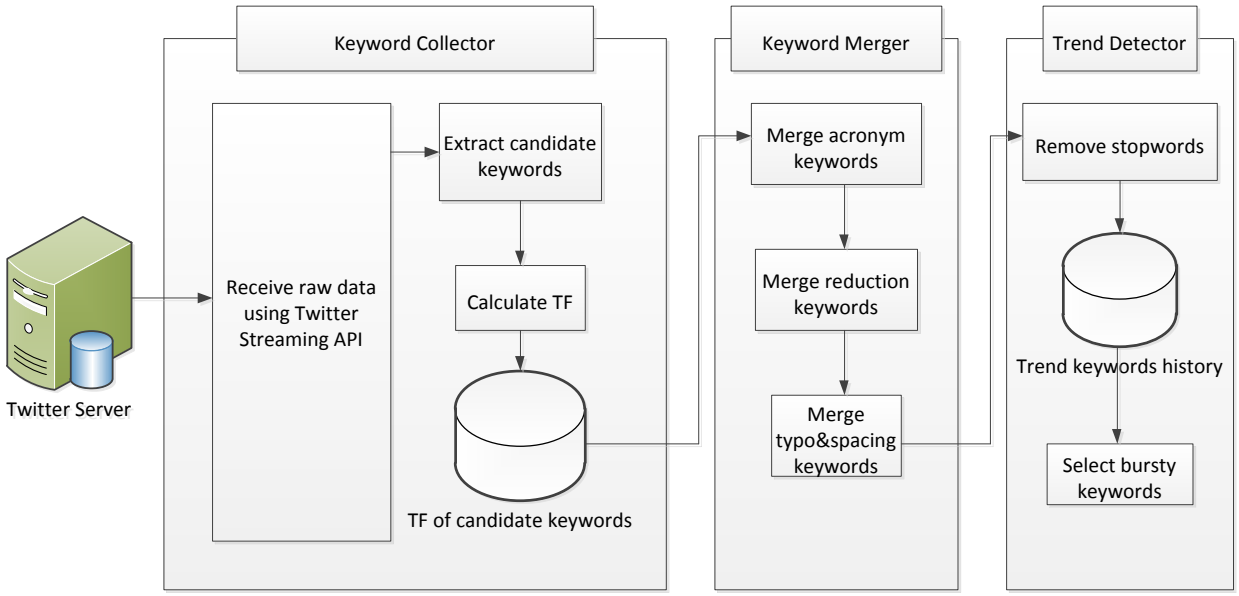

Figure 5: Overall System Architecture[57]

Their system first collects user tweets via Tweeter Streaming API and extracts candidate keywords from them (tweets) by calculating their term frequency (TF). It then identifies various word variants which are considered some full keywords and merge them semantically with the same keywords also adjusting their term frequencies accordingly. It then determines trend keywords based on their rank and bursty keywords are selected from them based on the temporal pattern of their popularity. Their work is different to ours in that their work is not domain specific as ours focuses on government service delivery. The other difference is that their work was based tweets, which are limited to only 140 characters unlike Facebook statuses which are not limited.

Conclusion

This chapter presented a brief literature review relating to this research project. The literature review was done so that an understanding of the tools that were to used when developing the proposed system could be obtained. The chapter was basically focusing on the background study of the proposed domain in this study. The chapter concludes by giving an outline of a few related works which have been done under research domain.

Chapter Three: Research Methodology, System Design and Core Used Technologies

Introduction

This chapter describes the research methodology employed in collecting data, advancing the developmental model and identifying the prototyping method relevant for this research project. It also discusses the low level system and high level system. The Low level system in this chapter focuses on the structures of each and every module in the developed system while high level system design gives the interaction details between the modules that constitute the system. This chapter basically discusses the research methodology used in conducting this research and the system design.

Research Methodology

The research method that was followed in this research consists of a combination of well-established research methods which are requirements gathering, system development and system testing. The research methodology that was used was the quantitative research methodology as we had to collect data from Dwesa community using questionnaires.

Feasibility Study

Informal interviews, Observations and Questionnaires

The system is developed specifically for use by the Dwesa community but the system can be easily deployed for use in any rural area with Information and Communications Technology (ICT) access; upon that we had to understand the community so we could properly specify the functional and non-functional system requirements.

During the course of this research, the research site was constantly visited for one week per month by the SLL research team. During the visits, the SLL research team conducted computer literacy trainings and also got to know more about the research site at large. The literacy trainings were mainly focusing on teaching Dwesa residents how to use computers, teaching them how to create word documents, how to use LibreOffice calc for calculations etc. and the basics of using internet such as creating emails, Google searching, singing up on SNSs sites etc.

During these computer literacy trainings we could get time to interact with Dwesa residents and teachers who were attending to and get to know more about the situations they are living under. Just one thing that most Dwesa residents complained about was the availability of transport when they need to go to Willowvale town, a town located approximately 40 km from their location. The residents use public buses mostly as their main source of transport to and from town. The buses cannot reach some of the places because of the bad road networks. They also travel long distances to and from bus stations. The time spent there by the research team gave a clear understating of the need of these ICTD initiatives in rural areas of South Africa. Informal interviews were also conducted during these computer literacy trainings. The objectives which were motivating informal interviews were to find out; how people of Dwesa community are currently reporting the issues regarding government service delivery to their local government and what are the drawbacks on the system they are using. The informal interviews and observations were preliminary studies, which were then used to formulate a proper questionnaire for data collection. The main objective of the questionnaire was to validate the problems which were noticed during the visits while the research was being carried out.

The questionnaire was used to find out how people of Dwesa are currently reporting the issues related to government service delivery that they are facing. The questionnaire focused on finding out about; SNSs currently being known and used, methods currently being used to communicate their problems with their current local government, and the possibility of the improvements the developed system could bring. The use of the questionnaire made us to describe our study as quantitative research. Nancy [58] describes quantitative research as the research that uses numerical data and statistical analysis to obtain information about the world, giving the opportunity to describe and examine the possible relationship among variables. According to [59], quantitative research can be classified into four categories as survey research, correlational research, experimental research and causal-comparative research. In this research, quantitative survey research method was used for data collection and analysis.

Sukamolson [59], described survey research as the research which uses scientific sampling and questionnaire design to measure characteristics of the population with the statically precision. This research method is usually used for finding people's sentiments towards any phenomenon of interest, with the common goal of collecting data that will represent a certain population. The

accuracy of the collected data is mainly dependent on the sample size compared to the actual population size being studied. The idea behind this approach is that the margin for random error should be smaller by finding a larger survey sample [60]. In this research, the survey sample was smaller due to the number of drawbacks but assumptions had to be made by the author, because the author also grew up in marginalized rural areas like the research site of this study. A few of the drawbacks that resulted in finding small survey sample will be outlined in Chapter 5 under the section Problems Encountered. The idea of collecting data was to collect data after computer literacy trainings which were usually held at Nqabarha Senior Secondary School (SSS) and Mpume Junior Secondary School (JSS) conducted by SLL research team which the author was part of.

The problems mentioned in Chapter 5 under the section Problems Encountered resulted in a situation where the author would have to visit the schools that he could, to collect data. The schools that were randomly visited to constitute the survey sample were: Mpume JSS, Ngwane JSS, Ngqeza JSS, Lurwayizo JSS and Nondobo JSS. The travel costs and time constraints were the reasons for not visiting all the schools that are involved in the SLL initiative. In terms of the cost, the SLL research team's transport was not budgeted for visiting all the schools for data collection. In terms of time, there was a need for sufficient time to consider the main reason behind the visits to Dwesa community which was to conduct computer literacy trainings. Nevertheless, the author was able to get 16 respondents from the schools which were visited. The questionnaire which was used in collecting data entails three sections; **Section A**, **Section B**, and **Section C** which all have different objectives.

Questionnaire Results Discussion

The results which were found from the different sections will be discussed in this section. To start with, the findings of **section A** will be discussed, as it is the first section from the questionnaire which was used in collecting data.

Section A

The main objective of this section was to find out about the SNSs residents of Dwesa know, the SNSs they have accounts on and the SNSs they use the most. This section in short was mainly meant to establish the people's attitudes towards SNSs networks. This was done so as to see if the use of SNSs as means to better Dwesa residents' communication with their local government

would be possible. The other reason was to find out if the chosen SNS, Facebook is known and used in that region in order to avoid the assumption that Facebook is being used everywhere.

The first question from Figure 6 was meant to see if the respondents know any SNS. The top three most popular SNSs, Facebook, Twitter and LinkedIn were used as the options for respondents to choose from and the option to mention a few other SNSs they know. The results displayed on Figure 6, Facebook was the most known SNS with 14 respondents out of the 16 respondents which were involved in the survey sampling professing knowledge of its existence. The second question was meant to find out the SNSs that people of Dwesa already have accounts on with Facebook also being the SNS that most respondents have accounts on. Only one respondent had accounts on Twitter and LinkedIn. Based on these results, it can be easy to assume that there are many people who did not participate in the survey, who know and have Facebook accounts compared to other SNSs. There were only two respondents who did not know any SNSs and seven who did not have accounts from any SNSs.

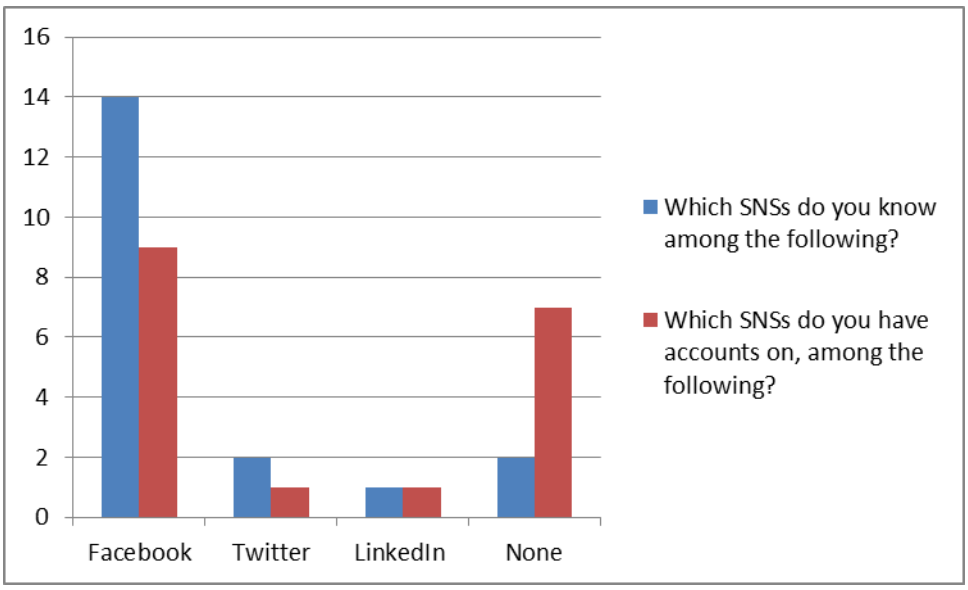

Figure 6: SNS Known and have Accounts on Results

The option where people were given a chance to mention few SNSs they know, "WhatsApp" the cross-platform mobile messaging [61] was confused by respondents for being an SNS-. Almost all the respondents knew "WhatsApp" and were using this application more as compared to the SNSs themselves.

The following question on Figure 7 was meant to find out the most used SNSs by the people who have accounts in different SNSs. Out of the sixteen respondents which were involved in the survey, eight of them professed to be using Facebook frequently as compared to other SNSs. There were no respondents who frequently used other SNSs as compared to Facebook. The other half of the respondents indicated that they did not use SNSs mostly. This question was mainly used to check whether or not there are any people who use Facebook, so that we could know if the choice of using Facebook as the source data for performing the objectives of this research was justifiable. the fact there are people who are using Facebook in Dwesa implies that, data will be available from the people of Dwesa on Facebook which represents their views, opinions, thoughts etc. concerning the government service delivery and issues they are facing.

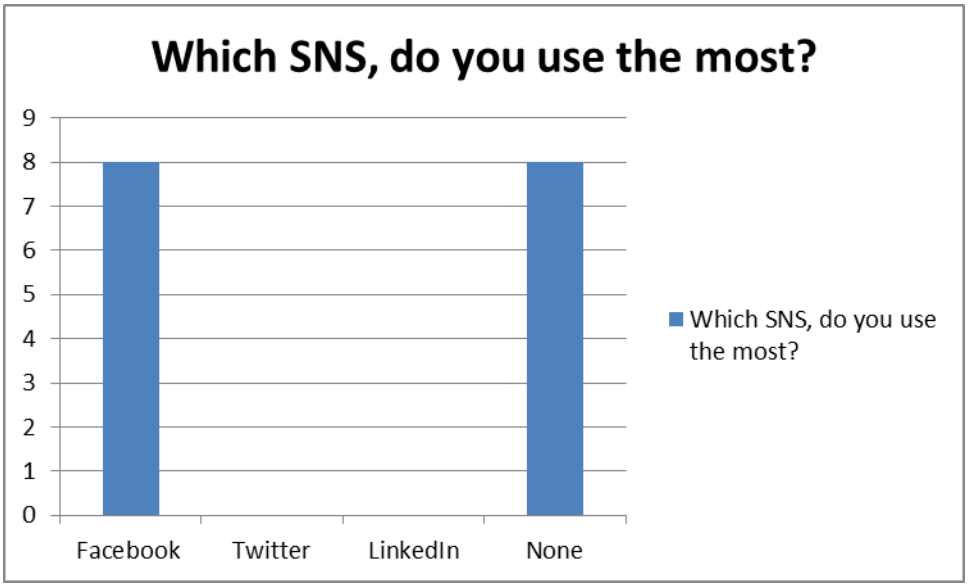

Figure 7: SNS being used the most Results

The following question was an extension of the question from Figure 7. The reason for this was to know how much data should be expected from the SNS they chose from the above questions. The reason being, when people hardly update their status, comment, upload a picture, it implies that there will not be much data to expect. Figure 8, shows their usage scale of this specific SNS from less than once a month, meaning that such a person hardly ever logs in on the SNSs as compared to people who use SNSs on a daily routine. Most respondents, who were involved in the research survey, indicated that they use SNS daily meaning that they login on this SNS almost most every day. There were two respondents that said they used a SNS once in two

weeks. The other two said they used SNS once in a week and less than once a month. These results are presented on Figure 8.

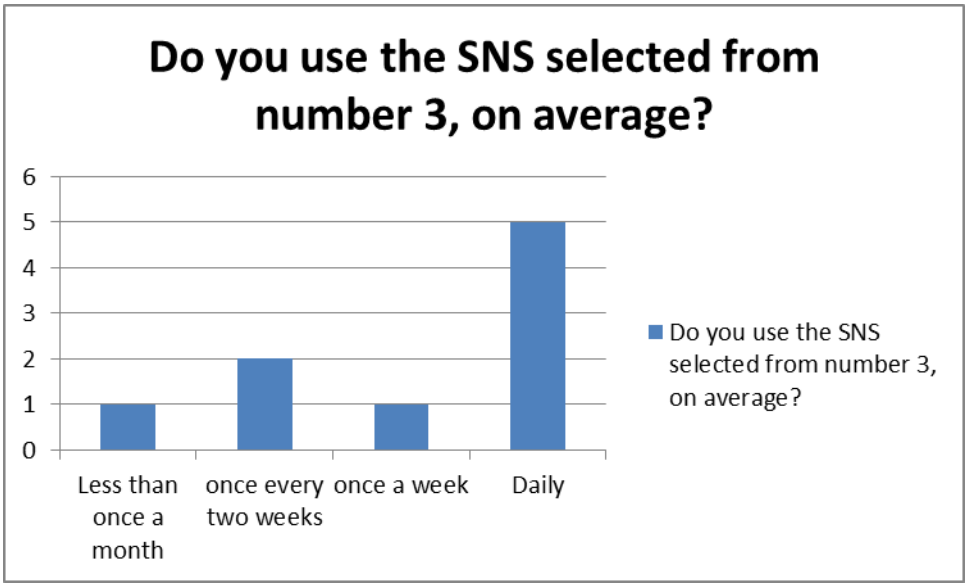

Figure 8: Average scale Usage of SNS

The fact that people at Dwesa use Facebook, and that there are people who use Facebook in their daily life routines does not suggest that they are always posting statuses, comments or uploading pictures. There are a lot of things which they could be doing ranging from reading other people statuses, checking trending news at those specific moments or just looking at other people's statuses. With those facts in mind, we had to supplement the questions in Figure 8, by the following question in Figure 9 which was meant to find out the most activities they usually perform on the SNSs.

The following questions were used to measure the possibility of the data that could be found from the SNSs, the people of Dwesa are using. Few actions which were normally done by people on Facebook ranged from updating statuses, reading other people's statuses, advertising etc. and were used as the questions to measure the activities the people Dwesa are mostly performing. The assumption being, if many people post statuses and comments it would be possible to get their subjective data which could be used to track their opinions, views, thoughts, ideas etc. and also perform trend analysis from the textual data they would be producing. Figure 9, presents the results found from the survey sample. Four respondents said they used SNSs most of the time to

send messages and three used SNS mostly to update statuses and the other two used SNSs to read other people's statues and uploading statuses.

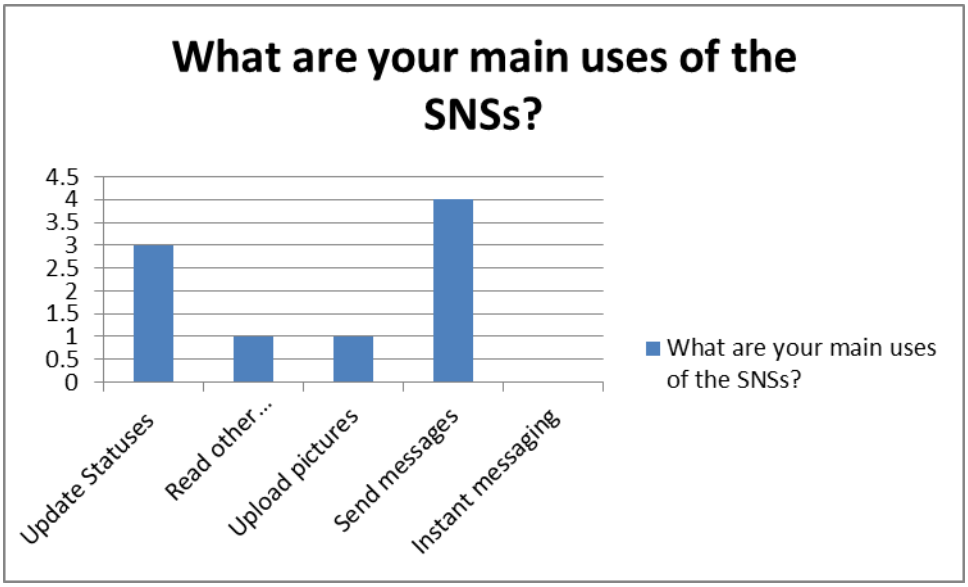

Figure 9: Main Usage of SNSs Results

There were no respondents who said they used SNSs for instant chatting. This could be attributed to the fact that such people may be using WhatsApp as this application was found to be known by most respondents. The other assumption could be the reason that all of the respondents said they mostly used their mobile phones to access the internet and/or SNSs. The results are provided in Figure 10.

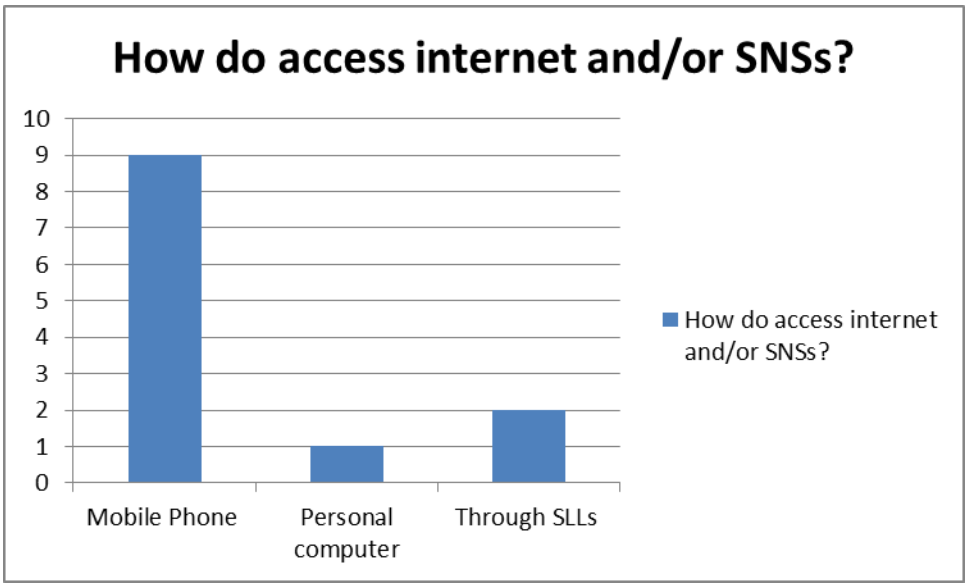

Figure 10: How do they access SNSs Results

This implies that they buy data bundles or airtime to access these services. It costs more to stay online on a SNS chatting compared to the already developed mobile instant applications like WhatsApp. There was one respondent who also uses a personal computer and two which access internet and/or SNSs through SLLs. This leads to the conclusion that most people in Dwesa frequently use their mobile phones to access the internet because 100% of the respondents use their mobile phones to access the internet with a few using devices other than mobile phones.

Section B

The main objectives of this section were to find out the reasons of not using SNSs from the respondents who were not using SNSs. It was also intended to be used in order to find out if they would ever be interested in using SNSs in the near future.

The first question was intended to ask the reasons of not using SNSs. The researcher's assumptions were that people might not be using SNSs because they either do not know SNSs exists or were not interested on using them or do not know how to use them or do not have access to them and they are reluctant to ask for help in cases where they do know how to use them. The second question was intended to ascertain if they would ever be interested in being trained on how to use SNSs. Three respondents from the first question said they were not interested in using the SNSs and on the second question four respondents indicated that they maybe be interested in using SNSs.

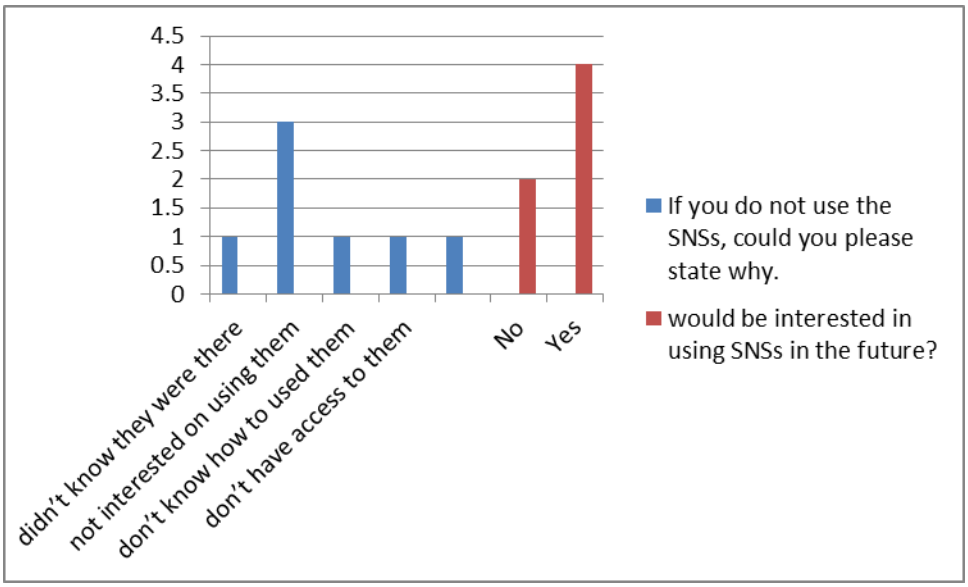

Figure 11: Possibility of Using SNSs in Future for the Residents not using the SNSs Results

These findings in section B were then compared with the findings in section because it is possible that people could not be using the SNSs or not accessing internet because they do not have devices to access internet or they do not have money to buy internet bundles or air-time. The other reason could be that they are old, based on the researcher's experience as elderly people in rural areas do not use SNSs with some not even knowing that SNSs exist.

Section C

This section was used to find the particulars of the respondents such as employment status and their age groups. The survey respondents which participated in our data collection were mostly teachers from the schools we visited. The teachers were either employed on a full-time or a part time basis. The other two participants were local residents who were attending the computer literacy training.

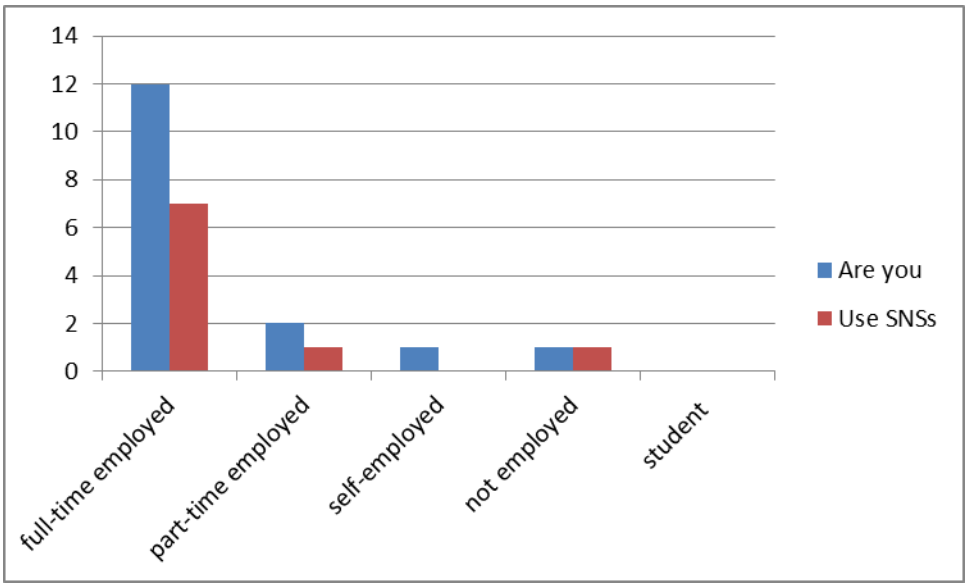

Figure 12: Employment Statuses Results

As can be seen in Figure 12, out of the 12 full-time employed teachers, seven of them used social network and one from the local residents also use SNSs. This implies that most employed people can afford to use SNS and some already use SNSs. The other thing which was also noticed from the results is that even unemployed people can afford to and use SNSs. As earlier stated, another reason why people are not using SNSs is their age. The results from Figure 13 bellow, focused on the age group of people between 46-55 years. Of the six respondents who belong in that age group only one used SNSs. This was to validate the notion that elderly people in most rural areas of South Africa do not use SNSs.

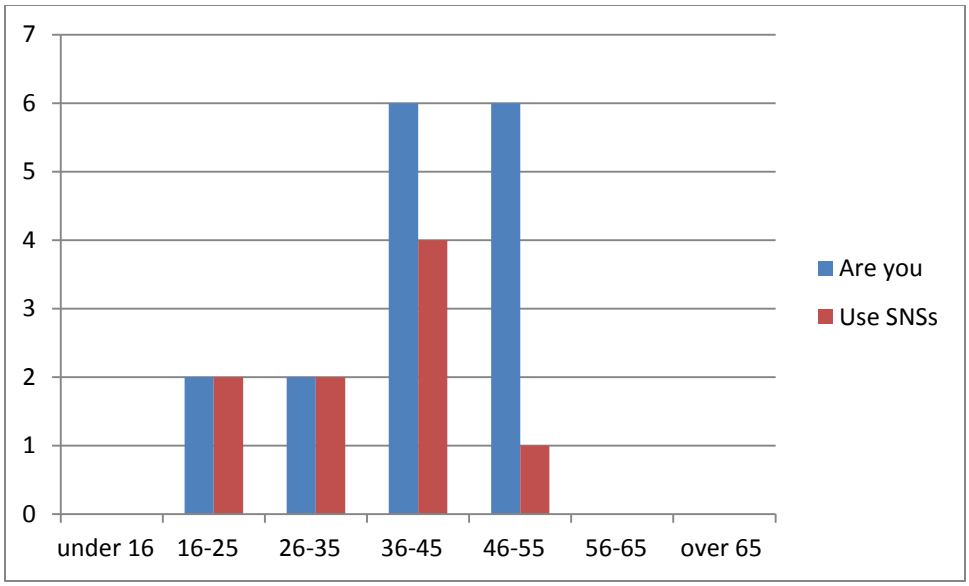

Figure 13: Age group and SNSs Usage Results

This section aimed at establishing the factors which usually lead a situation in which some people do not use SNSs at all. The following section was the last section from the questionnaire which was used for data collection.

Section D

This section mainly focused on finding out about the methods currently employed by the people of Dwesa community to relay the problems they are facing to their current local government, so as to establish the possible improvements the developed system could bring to the community. All respondents were asked to respond to this section where someone is using SNSs or not but, one of the respondents did not respond to it on the basis that he/she was not into politics. This was also observed as one of the reasons why there is lack of development in rural areas. Some people do not want to participate in anything which involves government related issues.

The first question was meant to find out about the method they are using communicate with their local government. This was done so as to know how appropriate the proposed method in this research would be when used in Dwesa. Based on the results on Figure 14, it was found out that people in Dwesa call ward meetings whenever they want to address community related issues. There were few respondents who were saying they phone call, send E-mails or go to the government offices themselves.

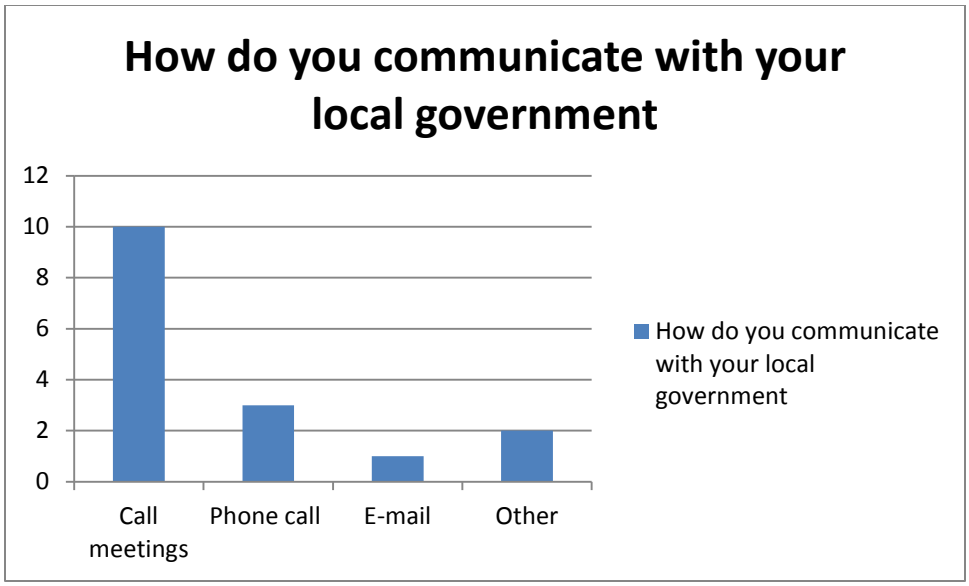

Figure 14: Current way of Communication with their Local Government Results

The other two questions which were asked on this section are the ones on Figure 15.

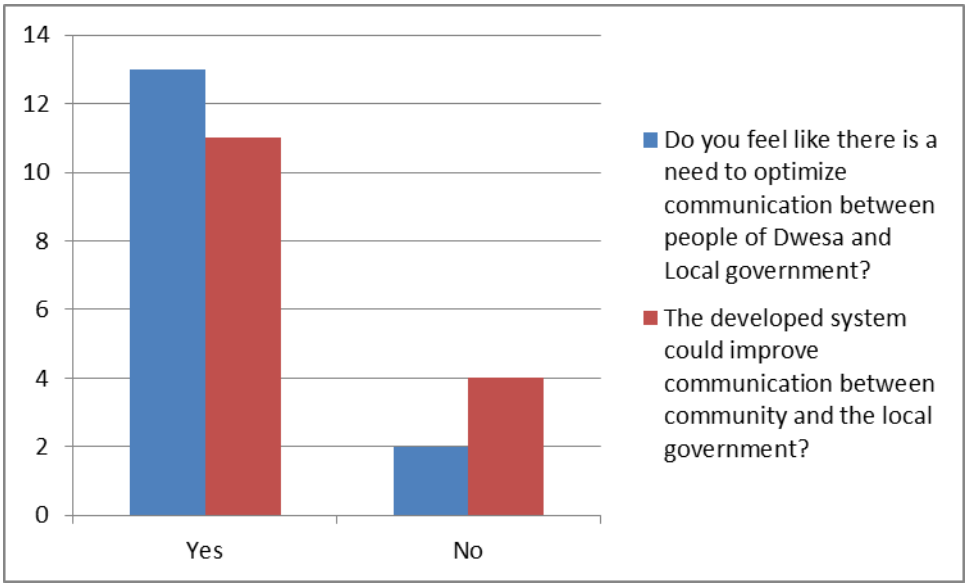

Figure 15: Need for Optimizing the current way of Communication and the possible Improvements by developed system results

The reasons for asking these two questions were to find out if people of Dwesa are happy with current way they are using and to establish if they think the system developed in this research will optimize their communication strategies or not. Thirteen respondents said there is a need for the optimization of the current way of communicating with the government. Further, eleven of

the respondents thought the developed system in this research would optimize their communication with their local government.

Through informal interviews reasons for the need for a change in their communication were asked for. The respondents said: 1) some people do not have time to attend meetings; 2) if there are many people who ask questions they usually get the feedback long after the meetings were held, sometimes they usually have to collect money to phone call the government officials for the responses because they usually take time to respond; 3) it is difficult for them to tell government officials what they are thinking or want because government offices are far from their community (Dwesa).

This information helped us to realize the possible important communication aspects which our developed system will bring to the Dwesa community. The system is developed in open source technologies and it uses Facebook which is the most widely used SNSs. The advantages that this SNS offers such as a cheaper way of communication and easy-to- use, will be valuable when the system is deployed to monitor the trends as the local government and local people will be able to communicate in a free and cheaper way.

Literature review

Through a literature review of similar previous works on our research domain, we were able to design and foresee the appropriate tools which could be used to develop our system. A review of programming languages and most of the tools which were previously used to develop similar systems was also carried out. A brief review of the tools that were chosen and used to develop the system and its usage is given on the System Architecture subsection below.

System Development

The Iterative Incremental Development approach was used in developing the system. The Iterative Incremental Development approach is a combination of the Iterative approach, which refers to an approach that allows cycling through the development phases from requirements gather to the final developed system deliverable[62] and Incremental approach which is an approach that allows the development of various parts in different stages and schedules which are then integrated when they are completed[62]. This development approach allowed us to

develop the proposed system in different modules which were then incremented and were also revised later whenever research objectives were not met.

Evolutionary prototyping

Evolutionary prototyping was used on the development of the system because the actual system's functional and non-functional requirements were not all known from the beginning of the project. During the process of the development of the prototype, every part of the prototype was reviewed and tested. If the results were satisfactory we would then move to the next step and develop the next part of the prototype. The first system prototype was developed and tested and additions were later made to optimize the prototype functionality and its usage. The first system prototype was then published in International Journal of Computer Applications IJCA[63].

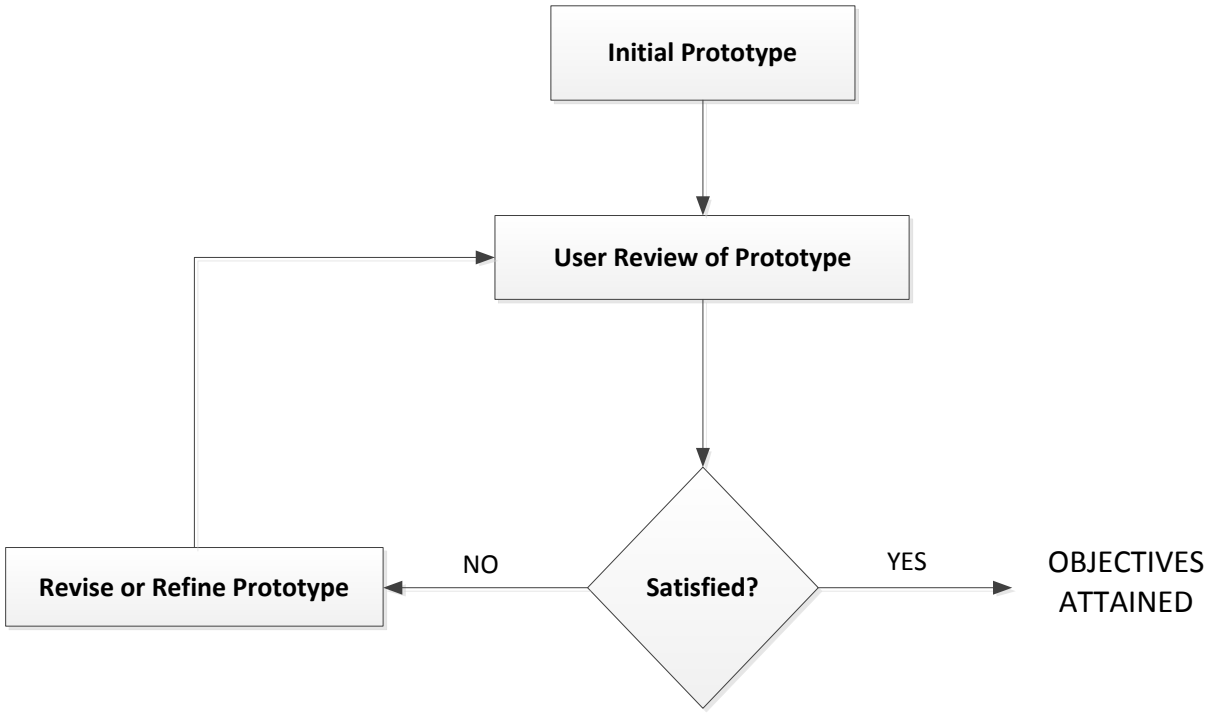

Figure 16: The Prototyping Process[64]

The above diagram, Figure 16, was adopted from [64], to best illustrate the process which followed during the time of the development of the system. After the initial prototype, the prototype was then reviewed and some refinements were made with new additions being made to ensure that the system is more usable. The following were the few limitations which were found on first prototype after it was reviewed: The first prototype needed too much human intervention. It would require the system administrator to supply train dataset when needed for

calculating the trends from Facebook. It would also require the system end-user to think of almost all the keywords he/she wishes to search. A more detailed and improved system design will be discussed in the System High Level Design subsection.

System Testing

During its development phase, the system was constantly modularly tested. The detailed system testing and evaluation with the experimental results are given in chapter 5.

System Design

The section is mainly meant to give the detail system user functional and non-functional requirements, low level system design and the high level system design.

System User Functional Requirements

Figure 16 shows the actual users which are expected to use and operate the system, the end-users and system administrator. To start with, the roles of the system administrator will be elaborated, followed by the actions the end-user will be able to perform in order to get the data of interest.

For the administrator to be able to operate the developed system, he/she needs to have Facebook account and be registered as a Facebook developer on it. This enables one to get the access token, which is needed for communication between Facebook servers and the developed system. The access token is an opaque string used by applications to make graph API calls when posting, deleting, or retrieving data from Facebook servers. It is also used by other applications which are not using it for making graph calls and other Facebook pages. The system administrator, after registering as Facebook developer, will need to get the access token from Facebook and supply it to the system. By so doing, the system will be authenticated to access Facebook users' data. The system will then retrieve data form Facebook database servers and when it is done, it will stop. After which, the time taken to crawl Facebook and index data will be displayed on the console when the process is done.

There are four systems functional requirements that the end-user can perform to get data of interest. These are, keyword searching, searching correlated words, search word frequencies and lastly checking words frequencies from the entire index. There is no specific order that should be followed when executing these functionalities. They all are built to optimize the system and to

make more data that can be retrieved and analyzed to make sense out of the data found from Facebook, the SNS used in this research.

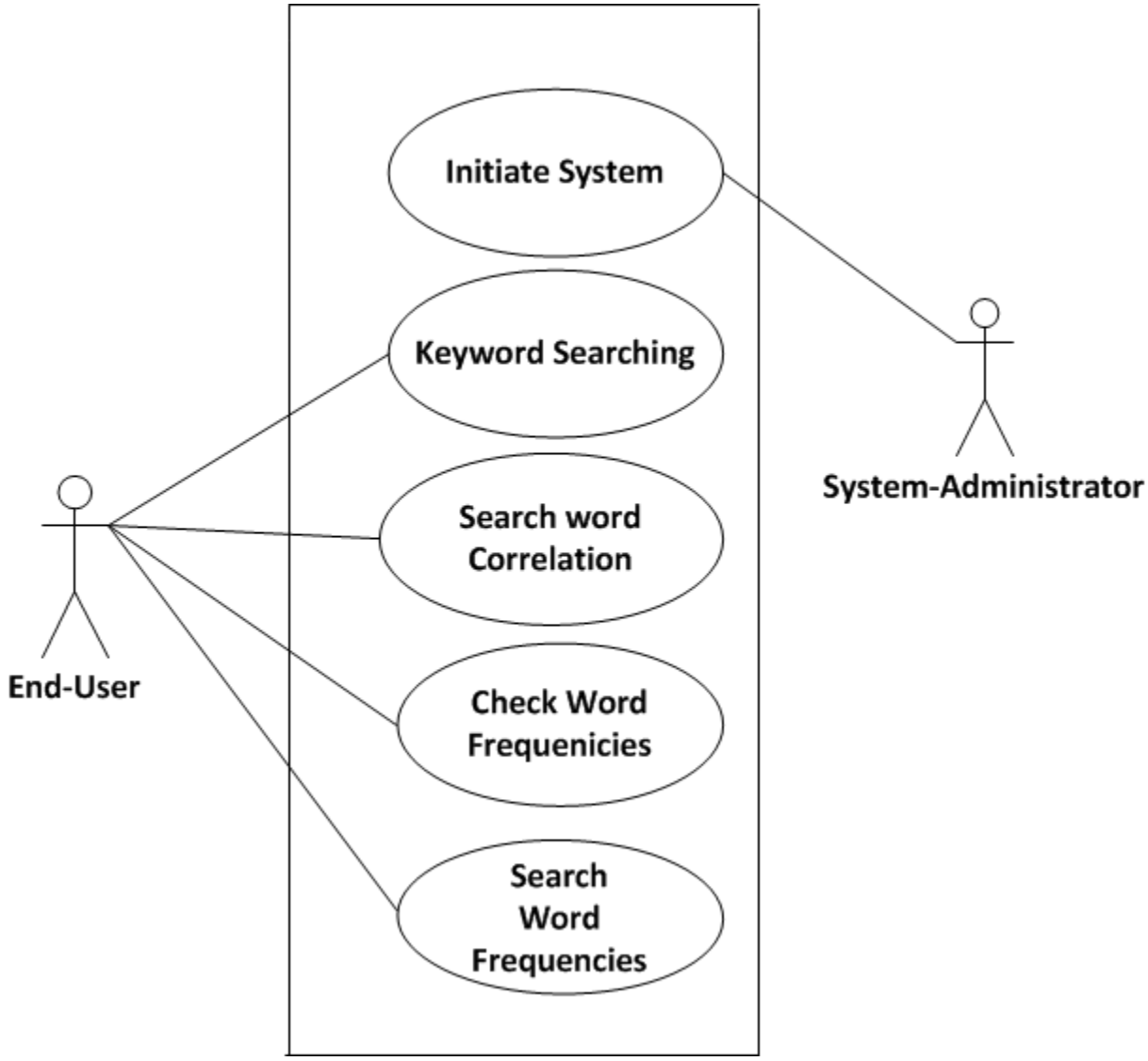

Figure 17: System User Functionalities

The functionalities which are provided by the developed system all have different business logics but share the UI. They also give back different kinds of feedback to end-users based on their logics. As prior stated, there is no stipulated manner in which these functionalities should be executed. An end-user can start Keyword Searching by typing the search keyword and the system will retrieve all the content from the index which contains the searched word. The results will be the top ten statuses and comments, their selection criteria and the reason for limiting the displayed results will be discussed in the high level system design subsection. The end-user

could also be interested in knowing the words which mostly occur with the searched keyword that functionality is handled by the search word correlation. The system accepts the keyword from the end-user then searches the index, and then retrieves all the content which matches the searched keyword and tokenizes, removes all the stop-words (the common words that contain no sematic content) which will be found from the content retrieved from the index based from the stop-words provided to the system as the train dataset. It then calculates the frequencies of the tokens, from which tokens with the highest frequencies being regard as the words which are highly correlated with searched word. The results are then displayed in a scatter plot for end-user to analyze.

The check word frequencies module is different to the other modules as it does not require keyword search for searching the index. This sub-module accepts the user input as the button click then retrieves all the data found from the index. The data is then preprocessed by tokenizing and filtering the stop-words. The frequencies of the tokens which are not filtered are then calculated. The results are then visual in the bar graph.

The last system module which is search word frequencies is used in conjunction with the already developed Suggested Upper Merged Ontology (SUMO). The reasons for using the already developed ontology were; to make the developed system more dynamic so that it could be possible to use for different purposes but not limiting it government service delivery, it was impossible to predict the possible words which could be found from the data retrieved from Facebook SUMO was to help through the words already on it. This system sub-module accepts the user input as the keyword then searches the ontology and retrieves data from the ontology.

The data is then processed by separating joined words and removed the found delimiters. Thereafter this sub-module retrieves all the data found from the index and then matches it with data from the ontology. The frequencies of the matching words are then calculated. From there, the end-user can know what trending words from Facebook are. The results are visualized on bar graphs so that the end-user can easily see the words with high frequencies (trending). All the index searching sub-modules require search keyword for searching the index except check word frequencies sub-module. For all index searching sub-modules if there is no data found matching the searched keyword, there will be no results returned to the end-user.

System Low Level Design

This section is aims at giving the detailed interactions between the components that constitute each every module in the developed system in this research. The developed system consists of two modules Facebook Crawling, Text Extraction and Text Indexing and Index Searching. Facebook Crawling, Text Extraction and Text Indexing serve the purposes of collecting data from Facebook. Index Searching module has four sub-modules serve the purposes of checking the correlation of the keywords, checking trending words, trend analysis based from the data found from the ontology and index, and the keyword searching for retrieving the actual content in the index.

The module for collecting data from Facebook will be discussed first giving the actions needed to be performed by the system administrator and the expected actions to be performed by the system in order to accomplish the desired goal, which is to get data feeds from Facebook, preprocess and index for later usage. As previously stated this module will be operated by the system administrator and the administrator will be required to have obtained the access token from Facebook servers.

The actions that will be performed by the system administrator, end-users and components that constitute each system module will be denoted by the word “message and the number” from the top to the bottom based from their sequence of occupancies.

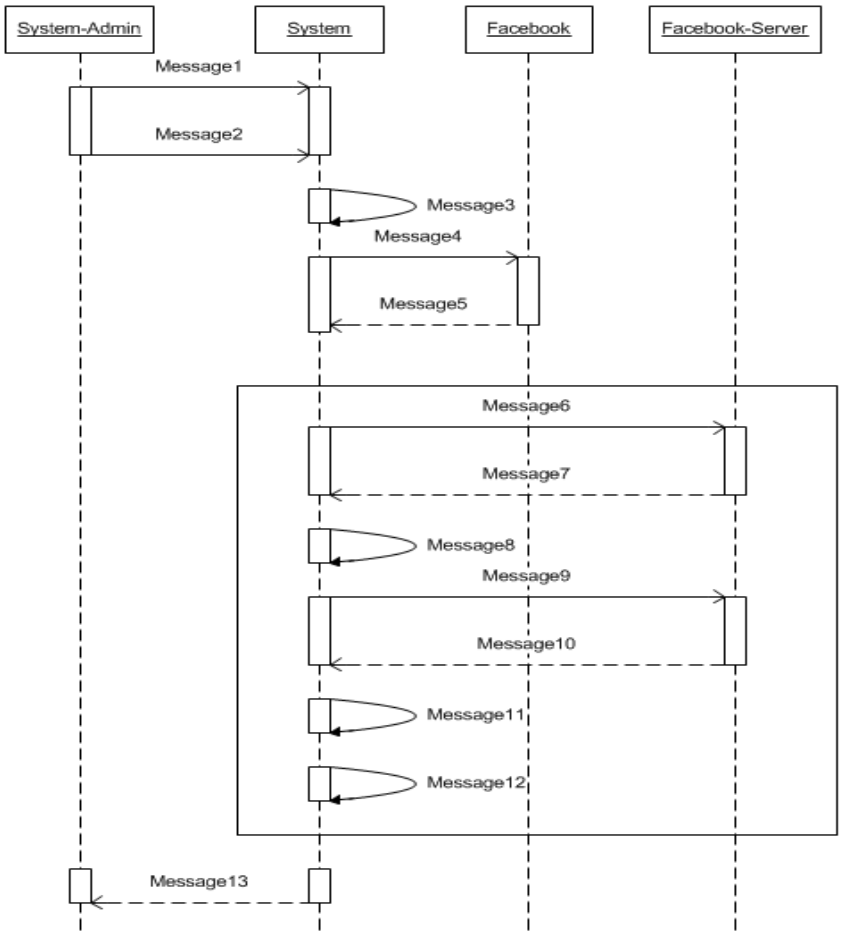

Figure 18: Facebook Crawling and Data Collection

Message 1 - the administrator supplies the access token to the system, by copy the string provided by Facebook server and supplying the string to the system.

Message 2 - the administrator is required to initiate the system so that system can start communicating with Facebook servers.

Message 3 - the system, after it has been initiated, starts by checking if there is any created index. If found, the index then deletes all the contents found from the index and if there is no index found, it creates a new index.

Message 4 – the system sends the authentication request to Facebook using the provided access token.

Message 5 - Facebook replies by accepting or denying the request if the access token is invalid.

Message 6 – the messages in the square boxes from messages 6 to 12 represent the processes which reoccur up until the system finishes crawling and indexing the data from Facebook. Message 6 is the message sent by the system to the Facebook servers requesting for friends' user IDs.

Message 7 – Facebook replies by providing the requested data, which are Facebook user account IDs.

Message 8 - the system calls the class for parsing the actual user IDs into plain textual data.

Message 9 - Uses the Facebook user account IDs from Message 8 to request for the statuses and comments of the users with account IDs and if found takes one latest status of every user.

Message 10 - Facebook sends back the actual requested data statuses and their comments.

Message 11 - calls the class for parsing the retrieved data into plain textual data.

Message 12 - the system then writes the data to the index. The messages reoccur until the last friend's user ID and the status are both crawled from Facebook.

Message 13 - the system then sends the actual time it took to crawl Facebook and index data in the console when it is done.

The following four diagrams represent the underlying system architectures of the four system user functionalities mentioned in Figure 17. The following is the underlying system architecture for keyword searching, content matching and term frequency analysis sub-module (Search word Frequencies, Figure 17). This sub-module is an OWLIR (Ontology Web Language and Information Retrieval) sub-system because it uses SUMO and Lucene information retrieval systems. According to [65] the use of this system architecture has the following advantages:

1. It uses the sematic information to guide the query answering process;
2. It enables answers with a well-defined syntax and semantics that can directly be understood and further processed by automatic agents or other software tools; and
3. Provides information that is not directly represented as facts in the WWW, but which can be derived from other facts and some background knowledge.

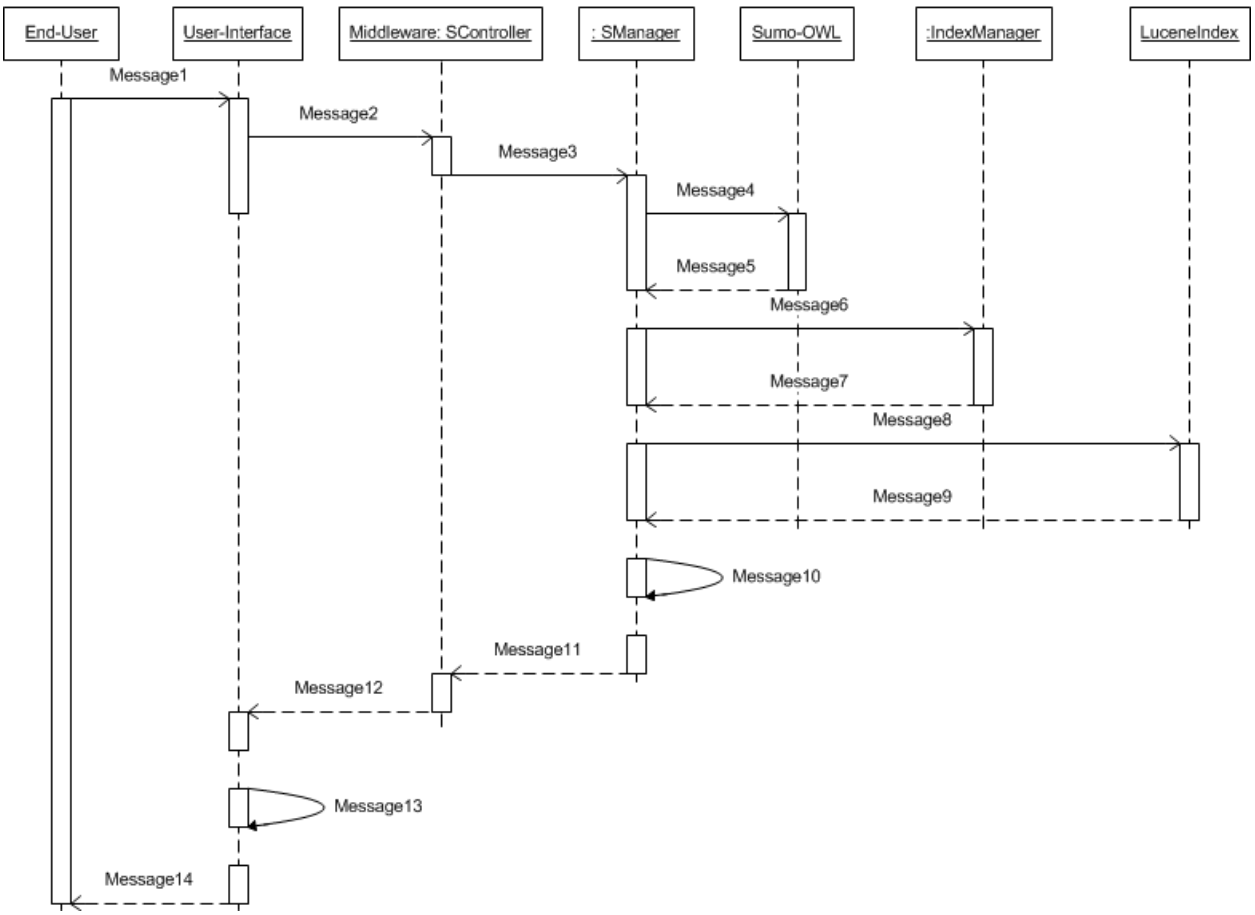

Figure 19: Keyword Searching, Content Matching and Term Frequency Analysis module

Message 1 – the user supplies the keyword to be used to search the index.

Message 2 – the UI accepts the keyword parameter and passes it to system middleware.

Message 3 – the middleware then passes the searched keyword to search manager class, which is the class that performs most of the core tasks of this module.

Message 4 – the search manger class then searches for the ontology using the provided keyword.

Message 5 – the ontology gives the feedback as the data that matches the search keyword, preprocessed and the data is then stored in an array.

Message 6 - the search manager then requests for the path to the index directory by verifying the provided path to the index, also checking if there is any index in such specified index directory.

Message 7 – message regarding the verification of the index path.

Message 8 - the search manager then searches for the index by retrieving all the data found in the index.

Message 9 – The index then gives feedback by providing all the contents from it. The data is then tokenized so that it could be single which could be matched against the data found from the ontology.

Message 10 - the search manager matches the words from the ontology and the words from the index from the two developed arrays. The frequencies of the matching words are then calculated. The frequencies and the matching words are then stored in an array.

Message 11 - the search manager then send feedback the array created in Massage 10, to the middleware.

Message 12 - the middleware also passes the array to UI.

Message 13 - At background of the UI, a bar graph is created based from the data on the array from middleware.

Message 14 - the data then visualized for the end users to do analysis from it.

The following diagram represents the underlying system architecture of the search keyword correlation module from Figure 17. This system module is similar to the above mentioned module Figure 19; they both require keyword for searching the index. The first three messages from Figure 19 and Figure 20 are similar. However, the difference from the two diagrams lies in that search manager in Figure 19 starts by searching the ontology while in Figure 20 the search manger starts by searching the index only and does not use the ontology at all.

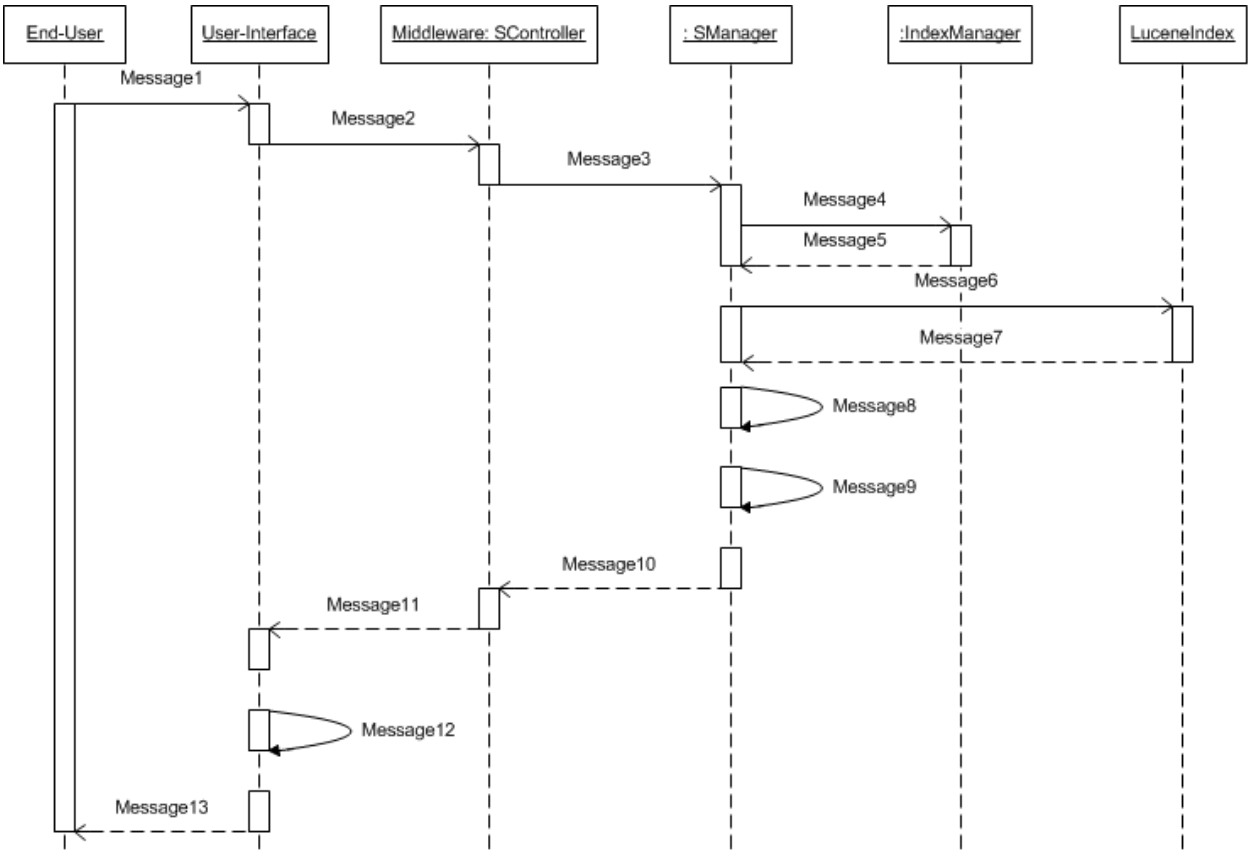

Figure 20: Words Correlation Analysis

Message 4 - the search manager then requests for the path to the index directory by verifying the provided path to the index, also checking if there is any index in such specified index directory.

Message 5 – is a message regarding the verification of the index path.

Message 6 – the search manager then searches the index by retrieving all data that matches the keyword from the index.

Message 7 – The index then gives feedback by providing all the contents matching the searched keyword. The data is then tokenized and stored in an array so that it can be used for word frequencies calculation.

Message 8 - matches the array of the word tokens from the index with stop-words array provided as training dataset. The words found from both arrays are regarded as the stop words and do not add to the array which is created for calculating the frequencies of the words which are not the

stop-words. The array of the words which are not found with the words from the stop-words will be added on the new array.

Message 9 -calculates the frequencies of the words from the array created from Massage 8. The frequencies and the actual words are then stored in the final array which will be used to visualize the data.

Message 10 - the search manager then sends the feedback, the array created in Massage 9, to the middleware.

Message 11- the middleware also passes the array to the UI.

Message 12 - At background of the UI, the scatter plot graph is created from the data in the array located in the middleware and the array of the random which are used at the values for axes. The range of random numbers depends on the size of the array from the middleware.

Message 13 - the results are then visualized to the end-user.

The following is the underlying system architecture of the keyword search module mentioned in Figure 17.

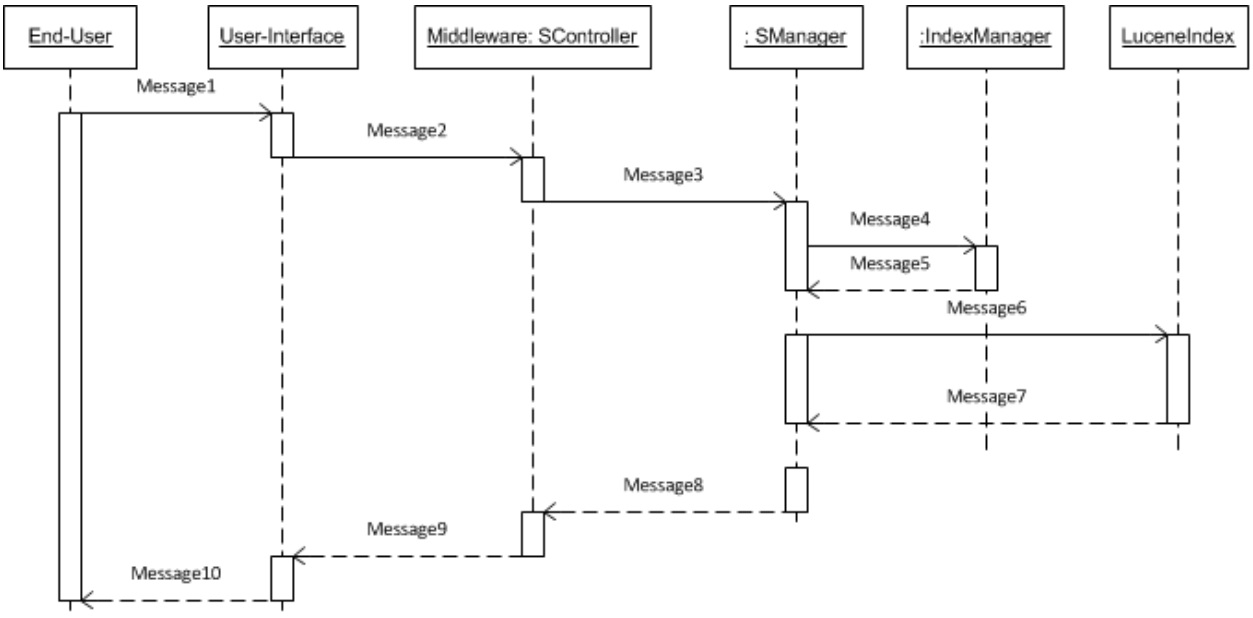

Figure 21: Keyword Searching

Starting from message 1 to message 6, messages and responses sent and received by different components constituting the modules from Figure 20 and Figure 21 are similar. There is nothing much which is performed by this module besides reading the index and displaying the content (Statuses or Comments) which match the searched keyword.

Message 7 – The index then gives feedback by providing all the contents matching the searched keyword. The data that is given as the feedback is the top ten statuses and comments that have higher hits of the searched keyword. The results are then stored in the array.

Message 8 - the search manager then sends feedback, the array created in Massage 9, to the middleware.

Message 9 - the middleware also passes the array to the UI.

Message 10 - the results are then displayed to the end-user, and thereafter the end-user can read them.

The following figure, Figure 22 is the underlying architecture of check word frequencies module described in Figure 17. This module is different from the other three modules which are designed to be used by the end-users in that it does not requires the keyword to search the index and calculate the word frequencies.

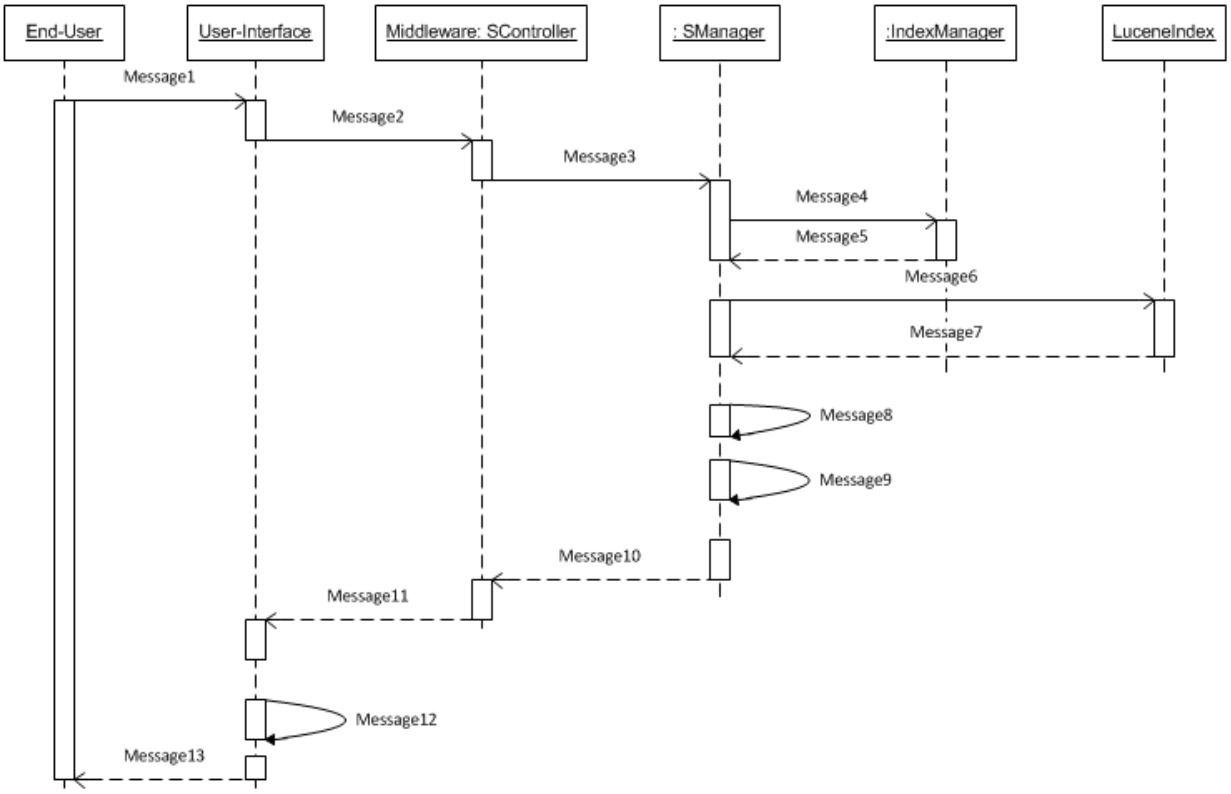

Figure 22: Check Word Frequencies

This module only requires the end-users to click the button and then the system will search the index, retrieve data from the index, calculate word frequencies and visualize the data in a bar graph. This module is designed to be used when the end-user wants to check the trending words from data collected from Facebook. The following messages below describe the processes that occur when this module is being executed.

Message 1 - the user submits the search request by clicking the button.

Message 2 - the middleware then gets initiated.

Message 3 – the middleware initiates the search manager class through the use of the construct with no parameter.

Message 4 - the search manager then requests for the path to the index directory by verifying the provided path to the index, in the process checking whether or not there is any index in such specified index directory.

Message 5 – is a message regarding the verification of the index path.

Message 6 – the search manager then searches the index by retrieving all data that is found from the index.

Message 7 – The index then gives feedback by providing all of its data contents. The data is then tokenized and stored in an array so that it could be used for content matching and words frequency calculation.

Message 8 - the content matching is done with the provided stop-words and the words from the array created in message 7. The words which are not found matching are then stored in the new array.

Message 9 - the frequencies of the words from the array created from Message 8 are then calculated and also sorted in a descending order and stored in the final array to be used for data visualization.

Message 10 - the search manager then sends feedback, the array created in Massage 9, to the middleware.

Message 11 - the middleware also passes the array to the UI.

Message 12 - At background of the UI, a bar plot graph is created based on the data on the array from middleware.

Message 13 - the results are then visualized to the end-user.

The following subsection is the high level system design which is meant to give the interactions of the modules that constitute the entire developed system package and the brief background study of the used technologies in developing the system.

System High Level Design

The system is developed with open source technologies, because the SLL from where the system will be deployed uses open source software which is free. The system is developed in different modules which were constantly tested and later integrated together to form one system package. The developed framework serves two high level functionalities which are outlined as follows:

1. Data extraction from Facebook, text extraction, text preprocessing and text indexing.
2. Index searching which consists of four sub-modules which are outlined as follows:
   - Keyword searching, content matching and frequency analysis;
   - Keyword searching, content matching and correlation analysis;
   - Content matching and frequency analysis; and
   - Keyword searching.

These sub-modules are integrated together with one UI. The UI is developed in such a manner that it eases the usage of the system. These high level system functionalities and the undertaken literature review informed the design of the system. The following diagram illustrates the high level system architecture of the developed system in this research.

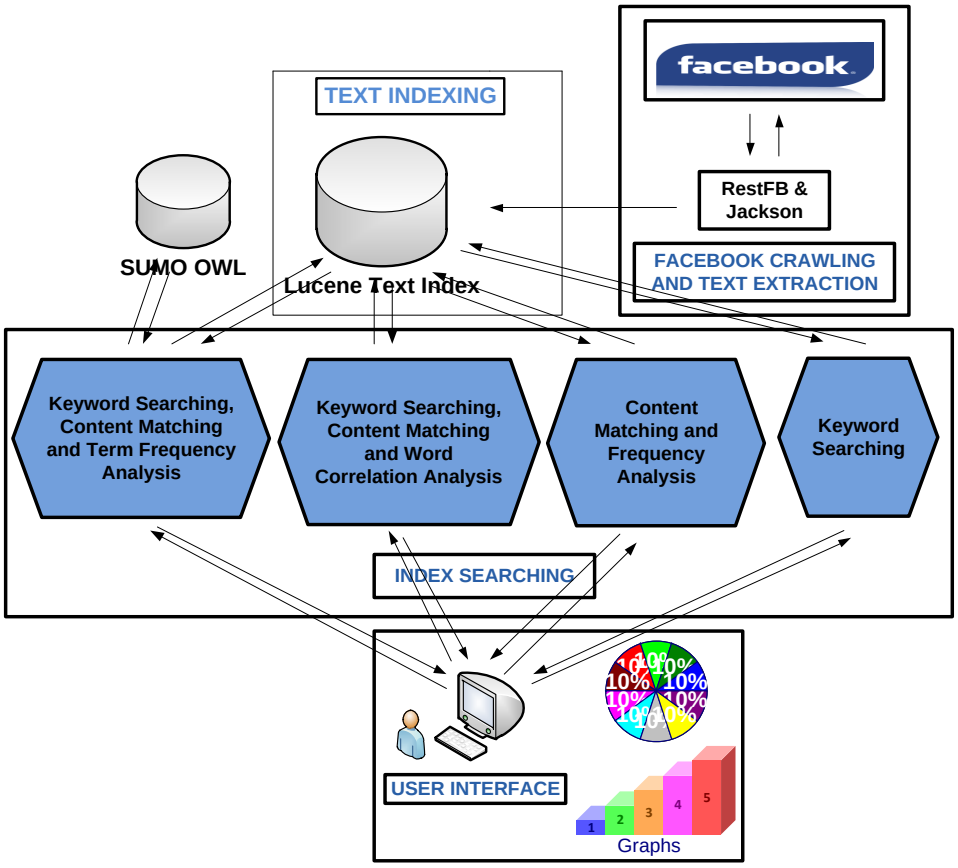

Figure 23: System Architecture

The system architecture above in Figure 23Figure 23 illustrates the internal and external structure of system modules integrated together in one package to form one system. The following subsections provide a brief background overview of the tools used and the design details of the different modules of the developed system starting from the back-end to the front-end. The whole system is developed on Linux Ubuntu operating system and Netbeans IDE platforms because they are open source technologies.

Facebook Crawling and Text Extraction

This system module was developed to crawl Facebook. There was a need to take into consideration the four polices described in chapter 2, under sub-section Web Crawling Policies. The following sub-sections discuss how we used the policies when we were designing and implementing the proposed system in this research.

Selection policy

The Facebook graph is larger and keeps evolving. Trying to crawl whole of it would require more bandwidth and larger data storages. As such, we created the crawler in the manner that it only crawl data feeds of users who are direct friends with the seed node. We used Breath-first-search (BFS) SNS sampling algorithm, which includes an agent that collects seed's friends user-ID's and an agent which is responsible for crawling friends' data feeds.

In BFS sampling algorithm web pages are crawled the way they are discovered [66]. Friends' user-IDs were put in a First-In First-out queue. Our crawler operates as follows: an agent contacts the Facebook server, providing a Facebook access token required for the authentication and permissions for accessing users' data. Once logged in, the agent starts crawling seed friends list collecting friends user-IDs putting the in FIFO queue. This queue enables the developed crawler to crawl the data feeds of all the seed's friends, taking one latest feed. The developed crawler is a focused crawler because it only crawl Facebook SNS and even from Facebook is basically focusing on data feeds. Focused crawlers can operate in two different modes batch mode, where they collect pages related to the topic periodically or in on-demand crawling mode where it collects pages at the request of a user query[67]. Our developed crawler currently operates on-demand crawling; a user has to log in first to Facebook and request an access which it then supplies to the crawler as ID seed or the start node of crawling process. The main

advantages of using Focused crawlers are that they require minimum local storage as they download only relevant pages and utilize bandwidth efficiently.

Revisit policy

Given the dynamic nature of Facebook this policy needed to be taken into account so that it could be possible to collect raw data which will be used for current or recent opinion trend analysis from Facebook.

Politeness policy

To accomplish this policy, the crawler should follow the three obligations; Firstly, it should identify its self as it is, this will serve many purposes such as counting the number of visits to the web database server and to manage the bandwidth allocated to web crawlers. Secondly, it must obey the crawler exclusion protocols which specify the parts of the server an administrator is willing to allow for crawling. Lastly the crawler should use its bandwidth efficiently; it should not download one page from the web site simultaneously and should give enough time between the two consecutive download[68]. Our crawler accomplished this policy because to identify itself it uses a Facebook access token to communicate with Facebook database web servers through Graph API. Through the same Facebook access token, our crawler is able to request data access permissions from Facebook database server. Further, friends data feeds are collected sequentially starting from the first user, collecting all data feeds and going to the next user data feeds when crawling is done, up to the end of the friends list.

Parallelization policy

We did not accomplish this policy on our crawler as it only uses user account IDs from FIFO queue as seeds for our crawler to crawl the Facebook graph. These user account IDs are loaded to our crawler frontier sequentially.

This is the system back-end module developed on top RestFB and Jackson java libraries. RestFB is a simple client java based library alternative for the Facebook Graph API written in Java [69]. This library is used by developers for communicating with Facebook database servers. FacebookClient interface was used to communicate with Facebook Graph API. This interface specifies how the Facebook graph API client is supposed to operate [70]. The sub- classes FetchConnection and FetchObject of FacebookClient were used for extracting friends' user

account ID's and data feeds from Facebook respectively. All data extracted from Facebook was returned in JsonObjects, an unordered collection of name/value pairs. JavaScript Object Notation (JSON) is a lightweight data-interchange format between machines and humans [71]. The data was also filtered by specifying the parameters to retrieve from Facebook and that eased the process of data mapping through java beans when parsing text. The extracted Facebook data was then parsed to plain text using Jackson library. The parsing of Facebook data was done for two reasons:

1. The user account IDs were parsed to get actual value of user account ID. The user account ID was then used as reference to retrieve the latest status update together with associated comments of the specific user.

2. The data feeds which is the actual data indexed, were parsed because Lucene only indexes plain text.

Lucene open source java library developed by the Apache organization for high text indexing and efficient search algorithms performance was used for indexing and adding searching functionality on the system. Collecting data from Facebook Breath-first-search (BFS) SNS sampling algorithm was used, which includes an agent that collects seed's friends user-ID's and an agent which is responsible for crawling friends' data feeds. In BFS sampling algorithm web pages are crawled according to the way they are discovered [66].

The Facebook Crawling and Text Extraction module operates as follows: it contacts the Facebook server, providing Facebook access token required for the authentication and permissions for accessing users' data. Once logged in, the agent starts crawling seed's (logged in Facebook user) friends list extracting friends' user-IDs and also crawling data feeds of each and every user in First-In-First-Out (FIFO) queue manner. Facebook data feeds are arranged in a chronological order with the most recent data feed appearing at the top Only the top data feed that is the last status update in every friend's Facebook account is crawled and indexed. This allows the tracking of the trending topic, on the network of the seed node.

Text Indexing

This system module was built on top Lucene information retrieval java library. The Facebook Crawling and Text Extraction module as aforementioned was developed for Facebook data

extraction and text parsing; the text was then indexed using Lucene. Lucene only indexes data available in textual format. The input text data in Lucene is stored in an inverted index data structure, which is stored on file system or in memory as a set of index files. On the development of this module we used different Lucene library classes to accomplish the objective of developing a working index. The following are the core indexing classes which were used during the development of the index: Directory, Analyzer, and IndexWriter.

Directory is the abstract class used to represent the location where the index files are to be or stored. We also used its sub-class FSDirectory store index files in the actual file system. This sub-class was used because we were getting large sizes of data and such data was also intended to be used for trend analysis purposes. The Analyzer class was used for converting the text data into a fundamental unit of searching, which is called a term. During analysis, the text data goes through multiple operations such as extracting the words, removing common words, ignoring punctuation, words stemming, also converted into tokens, and these tokens are added as terms in the Lucene index. There are already built-in Lucene analyzers which differ in the way they tokenize the text and apply filters, but in our development we used the StandardAnalyzer.

The IndexWriter class was used for creating and maintaining an index and its constructor accepts a Boolean that determines whether a new index must be created or an existing index is opened [43]. The IndexWriter class also provides methods for adding, deleting, or updating documents in the index. To avoid duplications and ensuring that most recent statuses were used, we used the deleteAll method every time new data was about to indexed. Lastly, Document class was used to represent a collection of fields, and the data was stored under two fields “title” and “content”. Title being status or comments and the content being the actual data we obtained from Facebook.

Index Searching

This module defines how end user’s queries, content matching, correlation analysis and trend analysis are done. This module from our system consists of four modules whose designs will be discussed in the following sub-sections.

Content Matching and Frequency Analysis

This system module adopted a fully-automatic method for trend analysis from text and term-document matrix. Term-document matrix is the frequency of terms or words in a collection of

documents, with documents being columns and rows representing terms. We provided our system with the collection of textual data in an array, which are regarded as stop-words train dataset. This module then, starts by checking if the words from the array are also available from the index. The frequencies of words which are found from the array and the index are not calculated. To access data from Lucene we used MatchAll query to retrieve all the documents from the Lucene index, then tokenized the retrieved data to enable us to match terms from the array and tokenize terms from index and to also be able to calculate word frequencies which are found not matching.

Keyword Searching, Content Matching and Frequency Analysis

This system module adopted the semi-automatic method for trend analysis for text in the index. It used SUMO data and data from the Lucene index for content matching and frequency analysis. SUMO is the largest formal public ontology in existence today with different domain ontologies on it [72]. The use of SUMO was to ensure that our system would not be restricted to the government service delivery related topics only. The usage of the SUMO aimed at giving the developed system in this research the ability to retrieve data from different domains in cases where end-users wanted to check trending topics which are not government services delivery related.

Through this system module, an end-user supplied a keyword to be searched. The module then regards the search keyword as the class which could be found from the SUMO. Thereafter, the module then searches for the SUMO with the keyword. If the class name matching the searched keyword is found the supper-class, sub-class and the instances of such specific found are then retrieved and preprocessed by removing unnecessary delimiters and separating the joined words. The words are then stored in an array to be used as the train dataset.

This module also uses MatchAll Lucence query to retrieve all the data found from the index. The data found from the index is then tokenized to single tokens. The tokens are then also stored in an array. The two arrays, the one which acts as the train dataset created from data found from the ontology and the actual data found from the Lucene index are then matched. The words which are found matching from the two arrays are stored in a different array. The algorithm for selecting these words is: every token from the index is checked if it is there from the train dataset array. If the token is found from the train dataset array is then added to a new array regardless of

how many times the token is found from the index data. Its instances are stored in the new array according to its number occurs. The newly created array is then used to calculate the frequency of the words that are found matching from the SUMO and Lucene index. The results are then visualized to the end-user the trending words on a bar graph. This module is used to check the trends from the entire index in the system.

Keyword Searching, Content Matching and Correlation Analysis

This system module also adopted a semi-automatic method for detecting the trends from the textual data found from our index. This module also requires the keyword to be searched but it differs from the previously described module in that it searches the index only and it does not use SUMO. This module searches the index using the normal keyword searching method and retrieves all the data found matching the searched keyword. The data is then tokenized and stored in an array. This module has a stop-words array provided to it as the train dataset for preprocessing data retrieved from the index by removing them, as the stop-words occur almost in every textual data. The two arrays are then matched by looking for the words that occur in both arrays. The words whose frequencies match are not calculated. The correlation in this module is calculated and determined based on the word frequencies, with the words with higher frequencies being regarded as the words with higher correlations. The results are then visualized on a scatter plot for the end-users to analyze and draw conclusions.

Keyword Searching

This sub-model enables the keyword searches. Based on results from the above described modules an end-user can search the most correlating and/or trending words through this system feature, to get the content where they can see what has been said on Facebook in relation to the words which were found correlating or trending. Keyword searching is not limited to searching for words found in the above described modules but can also accept any query and retrieves the documents that match the query. To limit the number of documents retrieved we used Lucene TopScoreCollector class and limited the results to top 10 documents with the highest hit scores. The reason for limiting the results which are retrieved through this system module is to make it easy for the end-user to easily get the sentiments of the retrieved content form the index.

User Interface

The UI was created using JavaServer Pages (JSP) and linked to each of the four index searching modules with different java servlets. The servlets are responsible for accepting queries from JSP and passing them to index searching modules and accepts the feedback from these modules and passes them back to be displayed and/or visualized on JSP page. Jfreechart bar graphs and scatter plot are used to visualize results from content matching and frequency analysis, keyword searching, content matching and frequency analysis, and keyword searching, content matching and correlation analysis modules. Results from the keyword are displayed in the play text.

Figure 24: System User Interface

The UIs for each and every module was developed and tested separately and they were all called to function in one UI to make one system package. The interfaces were developed according to the functionality they are supposed to offer and the expected results.

Conclusion

In this chapter we discussed the research methodology used and described the method used for collecting data at Dwesa, development model used during the development of the system and the prototyping method used for developing our system. This chapter also gave a brief background study of the technologies which were selected from the literature and used in the development of our system. The chapter sought to ensure that the reader easily understands the implementation details of the system which will be given in the next chapter, chapter four.

Chapter four: System Implementation

Introduction

This chapter discusses the implementation details of the system developed in this research. The previous chapter discussed the research methodology, the system design and the brief review of the technologies used. This chapter will focus on the actual system implementation starting with the module for crawling Facebook, gathering data, parsing text and text indexing. It will also discuss the sub-modules developed for indexing searching module and concludes with a discussion on the implementation of the UI.

Facebook Crawling, Text Extraction and Text Indexing

This module was developed in java programming language on Netbeans and Linux Ubuntu as mentioned in the previous chapter. Figure 25 is the actual code and the logic of this module. The other codes for parsing and indexing textual data will also be given and explained in this section.

```
35          FacebookClient facebookClient = new DefaultFacebookClient("CAACEdEose0cBAOT1jPoHKuz2WW2OK60T3gD907gATUvbkNEDUNgJ2oS1kuYvKcg
36          com.restfb.Connection<JsonObject> connection = facebookClient.fetchConnection("/me/friends/", JsonObject.class,
37                  Parameter.with("fields", "id"));
38          ObjectMapper mapper = new ObjectMapper();
39          for (List<JsonObject> jsonObjects : connection) {
40              for (JsonObject jsonObject : jsonObjects) {
41                  String json = jsonObject.toString();
42                  IdParser user = mapper.readValue(json, IdParser.class);
43                  JsonObject connect = facebookClient.fetchObject(user.getID() + "/feed/", JsonObject.class,
44                          Parameter.with("summary", true), Parameter.with("limit", 1),
45                          Parameter.with("fields", "message,created_time, comments"));
46                  String json1 = connect.toString(2);
47                  JavaParser user1 = mapper.readValue(json1, JavaParser.class);
48                  try {
49                      ArrayList<JavaParser.Data> projects = (ArrayList<JavaParser.Data>) user1.getData();
50                      for (JavaParser.Data p : projects) {
51                          addDoc(w, "Statuses", p.getMessage());
52                          ArrayList<JavaParser.Data.Comments.Data1> project = (ArrayList<JavaParser.Data.Comments.Data1>) p.getCommen
53                          for (JavaParser.Data.Comments.Data1 c : project) {
54                              addDoc(w, "Comments", c.getMessage());
55                          }
56                      }
58                  } catch (Exception e) {
59                      System.out.print(e.getMessage());
60                  }
61              }
          }
```

Figure 25: Facebook Crawling, Text Extraction and Text Indexing

The first line, line 35 in Figure 25 is the FacebookClient class which is used to communicate with Facebook. This class accepts the string access token for authenticating our crawler to Facebook servers. The access token as already stated entails the permissions of the data that we were allowed to access on Facebook servers. The second line uses the privileges of the FacebookClient class to access the friends list of the first node (the logged in user Facebook user, who requested the access token). To request the friends list of the first node fetchConnection

method of FacebookClient is used, to specify that we are requesting to access friends' data we "/me/friends/". To specify the type of the data to be returned, we used JsonObject class and also specified the actual data that we wanted to retrieve about each and every friend in the friends list using "Parameter.with("")" method to request only friends user IDs. The fetchConnection method returns data in JsonObject list. Lines 39 and 40 in Figure 25 show how to read the objects in the JsonObject list. Every read object in the list is then converted to string (Figure 25, line 41) so that it can be possible to parse to plain text which in turn can be used to request the latest status and comments of such specific user ID.

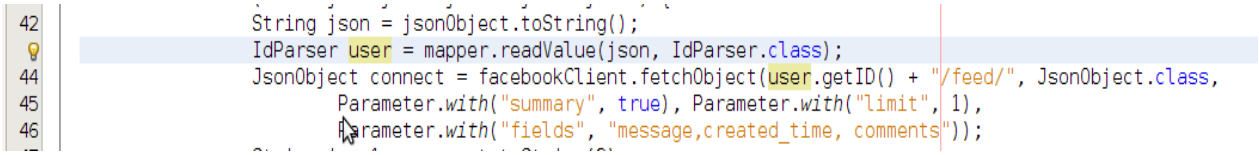

Figure 26: Call IdParser Class

Line 43 in Figure 26 shows how to call a class that is used to pass the ID JsonObjects to plain text. We declared a variable of a class IdParser and used getID method of the IdParser method to get the actual ID in the JsonObject. The IdParser class is used in conjunction with ObjectMapper class (Line 38, Figure 25) to map JsonObjects of Jackson library described in the previous chapter.

Line 44 in Figure 26 shows how status is read from Facebook of every use ID using fetchObject method FacebookClient. To specify that, we wanted the data feeds of the friends from Facebook servers we used "user.getID +"/feed/"" and the data JsonObjects as specified by the use of JsonObject class. The data retrieved was also filtered through the use of "Parameter.with ('')" method by specifying the fields we wanted to get on the JsonObject data.

Figure 27 is the actual implementation IdParser class developed in Java Beans. This was developed according to the data we were getting from Facebook of friends user IDs. This class is called every time a new or next user ID is to be used and needs it to be parsed to plain text.

```
7   public class IdParser {
8       private String id;
9
10              public String getID() {
11                  return this.id;
12              }
13              public void setID(String id) {
14                  this.id = id;
15              }
16
17
18  }
```

Figure 27: IdParser class

The feeds that we were getting also needed to be passed as they were also returned in JsonObjects from Facebook servers. The data feeds like user IDs also needed to be converted to strings (line 47 in Figure 28) for them to be easy to be parsed to plain text. This was necessary since we had to use Lucene index to index the data that we were getting from Facebook servers.

To parse textual data feeds from Facebook servers JavaParser class was used. To call this class we declared JavaParser variable, line 49 Figure 28. The data feeds that we were getting from Facebook servers were in the JsonObjects arrays format and JavaParser class was developed to parse and return the textual data from those JsonObjects. Line 51 in Figure 28 shows the declaration of the variable which was used to parse the statuses data feeds we were getting from Facebook. Line 54 in Figure 28 shows the declaration of the variable which was used to parse the comments which were found associated with such a specific status.

```
47              String json1 = connect.toString(2);
48
49              JavaParser user1 = mapper.readValue(json1, JavaParser.class);
50              try {
51                  ArrayList<JavaParser.Data> projects = (ArrayList<JavaParser.Data>) user1.getData();
52                  for (JavaParser.Data p : projects) {
53                      addDoc(w, "Statuses", p.getMessage());
54                      ArrayList<JavaParser.Data.Comments.Data1> project = (ArrayList<JavaParser.Data.Comments.Data1>)
55                              p.getComments().getData();
56                      for (JavaParser.Data.Comments.Data1 c : project) {
57                          addDoc(w, "Comments", c.getMessage());
```

Figure 28: call JavaParser class

Line 50 in Figure 28, shows how to declare the variable of JavaParser class. The JavaParser class was developed in a manner that would enable it to parse the contents of the JsonObject array. It was developed on top of Jackson Library. This class is called every time there is new or next data feed to be parsed. The JsonObjects data feeds were containing the status and the comments

associated with such specific status, but the comments were also in another nested JsonObject array inside the actual JsonObject array containing the data feed. Lines 52 and 53 show how the statuses were parsed from JsonObject data feed and indexed to Lucene index using "addDoc" method. Lines 54, 55, 56 and 57 show how the comments were read from the JsonObject data feed array parsed to plain textual data and also indexed to Lucene index using the same method "addDoc".

The snipped code in Figure 29 is JavaParser class. We could not include the entire code due to its length. The length of the this class was as a result of the objects we had to map when we were parsing the JsonObject data feeds, as the class developed in reference to the structure of the data feeds we were getting from Facebook servers.

```
10  public class JavaParser {
11
12      private ArrayList<JavaParser.Data> data;
13
14      public static class Data {
15
16          private String created_time;
17          private String id;
18          private String message;
19
20          public String getCreated_time() {
21              return this.created_time;
22          }
23
24          public void setCreated_time(String created_time) {
25              this.created_time = created_time;
26          }
27
28          public String getId() {
29              return this.id;
30          }
31
32          public void setId(String id) {
33              this.id = id;
34          }
35
36          public String getMessage() {
37              return this.message;
38          }
39
40          public void setMessage(String message) {
```

Figure 29: JavaParser Class

Figure 30 and Figure 31 show how Lucene was used to create and index the data respectively. Figure 30 shows Lucene classes that were used to preprocess textual data, create the index and specify the directory in which the index will be created on, the class to configure index writer and lastly the index writer class respectively. The last line, Line 33 in Figure 30 is the

IndexWriter method described the in previous chapter that is used to delete the contents of the index. This method is used every time the Facebook crawler module is being executed and new content is to be added. The usage of this method is to avoid duplications of statuses in cases where Facebook users did not update new statuses if this module had been ran before as their old statuses in our index and on Facebook will be the same.

```
29          StandardAnalyzer analyzer = new StandardAnalyzer(Version.LUCENE_42);
30          FSDirectory index = FSDirectory.open(new File("/home/mfenyanasi/Documents/index3"));
31          IndexWriterConfig config = new IndexWriterConfig(Version.LUCENE_42, analyzer);
32          IndexWriter w = new IndexWriter(index, config);
33          w.deleteAll();
34
```

Figure 30: Lucene Index classes

The following figure, Figure 31 is the method used to index text to Lucene index. This method accepts three parameters IndexWriter variable, the tittle name to be used to identify different fields in the index once created and the actual content to be indexed. Referring to Figure 28 and Figure 25 where this method is used the fields names used are “Statuses” and “Comments” and the actual data to be indexed are “p.getMessage” and “c.getMessage” to be stored under fields statuses and comments respectively.

```
66
67      private static void addDoc(IndexWriter w, String title, String cont) throws IOException {
68          Document doc = new Document();
69          doc.add(new TextField("title", title, Field.Store.YES));
70          doc.add(new TextField("content", cont, Field.Store.YES));
71          w.addDocument(doc);
72      }
73  }
```

Figure 31: Method for Text Indexing

In this section we described the development of Facebook crawling that is gathering data from Facebook servers, text extraction that is text parsing and text indexing. The next section will discuss the development of index searching modules which are used to handle end-user queries and perform data analysis.

Index Searching

This section discusses the four index searching sub-modules which were developed to search the created index in the previous described module Facebook Crawling, Text Extraction and Text Indexing system module. These sub-modules differ in the way they search the index, use the

data they retrieve from the index and the way they visualize and/or display the results to the end-user. To start with, we will describe the keyword searching, content matching and frequency analysis modules. When discussing each module we will start by its middleware component, the component that performs the controlling of the searching process by passing the messages to and from different components of the module, and then proceed to discuss the search manager module with their different expected results and lastly discuss how the results are visualized or displayed in the user-interface. These modules were developed in different sub-packages and having different JSPs which were created as the different modules' UIs.

Keyword Searching, Content Matching and Frequency Analysis

The snipped code is the middleware for this module. This middleware accepts the user query through line 28 as shown in Figure 32 below. The search manager class is then called through line 29 and also supplies it with the keyword searched. RequestDispatcher is used to receive and forward the messages between the JSP and the search manager class as shown in line 31 of Figure 32.

```
20  @WebServlet(name = "SControler", urlPatterns = {"/SControler"})
21  public class SControler extends HttpServlet {
22
23      private static final long serialVersionUID = 1L;
24
25      @Override
26      public void doPost(HttpServletRequest request, HttpServletResponse response)
27              throws IOException, ServletException {
28                  String searchWord = request.getParameter("searchWord");
29          SMnager searchManager = new SMnager(searchWord);
30          List searchResult = searchManager.search();
31          RequestDispatcher dispatcher = request.getRequestDispatcher("Searching.jsp");
32          request.setAttribute("searchResult", searchResult);
33          dispatcher.forward(request, response);
34      }
35
36      @Override
37      public void doGet(HttpServletRequest request, HttpServletResponse response)
38              throws IOException, ServletException {
39          doPost(request, response);
40      }
41  }
```

Figure 32: Keyword Searching, Content Matching and Frequency Analysis search controller component

The search manager class is initiated through the constructor that accepts the actual query issued by the end-user. The constructor method used by the middleware above Figure 32 is shown in Figure 33. The constructor also initiates the class which sets the path to the created index. The

name on the class that sets the path to the directory that stores the index is called in line 38, Figure 33.

```
37      public SMnager(String searchWord) throws IOException {
38          this.indexManager = new MangeIndex();
39          this.searchWord = searchWord;
40      }
```

Figure 33: Constructor Method

The following diagram Figure 34 is the actual class that keeps the path to the directory where the index is created and stored by the system module described in the first section of this Chapter.

```
12  public class MangeIndex {
13    private final FSDirectory indexDir;
14      public  MangeIndex() throws IOException {
15          this.indexDir = FSDirectory.open(new File("/home/mfenyanasi/Documents/index3"));
16      }
17      public FSDirectory getIndexDir() {
18          return this.indexDir;
19      }
20  }
```

Figure 34: Index Manager

When the index and the keyword are all initialized, the "search()" method in Figure 35 is then initiated. This method returns the list of an actual data which will be visualized to the end user. As stated in Chapter 3, this module starts by searching the SUMO ontology, retrieves the superclasses, sub-classes and instances that match the searched keyword, preprocess data and temporarily stores it in the array for later usage.

```
139     public List search() {
140         List searchResult = new ArrayList();
141         List<String> stopWords = searchOntology(searchWord);
142         stopWords.add(searchWord);
143         List<String> words = new ArrayList<String>();
144         ArrayList<String> wrds = new ArrayList<String>();
145         String delims = ", . ";
146         try {
147             IndexReader reader = DirectoryReader.open(indexManager.getIndexDir());
148             IndexSearcher indexSearcher = new IndexSearcher(reader);
149             Query query = new MatchAllDocsQuery();
150             TopDocs docs = indexSearcher.search(query, reader.maxDoc());
151
152             for (ScoreDoc scoredoc : docs.scoreDocs) {
153                 Document doc = indexSearcher.doc(scoredoc.doc);
154                 try {
155                     StringTokenizer st = new StringTokenizer(doc.get("content"), delims);
156                     String str = (String) st.nextElement();
157                     words.add(str);
158                 } catch (Exception e1) {
159                 }
160             }
161             reader.close();
```

Figure 35: Search Method

This function is executed by the snipped codes illustrated in Figure 36, Figure 37, and Figure 38. Line 141 in Figure 35 reads all the data retrieved from the ontology into the search method which it then uses for content matching and frequency analysis of the matching words. In line 147 in Figure 35, search method opens the index enabling it to be read. Line 149, in Figure 35 shows the Lucene query used to retrieve all data that is stored in the index. The data is then tokenized; the delimiters which were used to tokenize the data are defined in line 145, Figure 35. The delimiters are then used in line 155, in Figure 35 to tokenize the data that is retrieved from the index. The data is then converted to string data type and stored in the array in line 156 and 157 respectively for later use in content matching and frequency analysis, as shown in Figure 35

The following three figures are the snippet codes for the method of reading SUMO ontology. In Figure 36, line 75 the path to the SUMO ontology directory is specified. Line 76 is assigning the keyword to be used when searching the ontology as this method accepts the string parameter which is the searched keyword.

```
74      public List<String> searchOntology(String searchWord) {
75          String inputFileName = "/home/mfenyanasi/Documents/Library/SUMO.owl";
76          String j = searchWord;
77          List<String> sihle = new ArrayList<String>();
78          try {
79              OntModel inf = ModelFactory.createOntologyModel();
80              InputStream in = FileManager.get().open(inputFileName);
81              inf.read(in, "");
82              ExtendedIterator classes = inf.listClasses();
83              while (classes.hasNext()) {
84                  OntClass obj = (OntClass) classes.next();
85                  String className = obj.getLocalName().toString();
86                  boolean found = false;
87                  if (j.equalsIgnoreCase(className)) {
88                      if (obj.hasSubClass()) {
89                          for (Iterator i = obj.listSubClasses(true); i.hasNext();) {
90                              OntClass c = (OntClass) i.next();
91                              String st = c.getLocalName().toString();
92                              String s = st;
93                              String[] r = s.split("(?=\\p{Upper})");
94                               for (String ss : r) {
95                              if(sihle.contains(ss)) {
96                              } else {
97                                  sihle.addAll(Arrays.asList(ss));
98                              }
99                          }
100                         }
101                     }
```

Figure 36: Search SUMO Ontology Method

As earlier mentioned this method reads the superclasses, sub-classes and the instances of the searched keyword which is treated as the class when searching the ontology as declared in line 82, Figure 36. Line 88, in Figure 36 checks if the searched keyword has the sub-classes if it does

then iterates over them and retrieves all the sub-classes from SUMO which match the searched keyword. In cases where the sub-classes where found, they were returned as joined words but it could be observed that they were two or three words in one. There was therefore a need to separate those words because we needed those words for content matching as there is no one who writes like that on Facebook. It was challenging or practically impossible to find any matching words between the content from the created index and the content from the ontology. The splitting of the words was done using the Upper case delimiter as the words were differentiated by the use of the upper case latters and this was done in line 93, Figure 36. The words were then added to an array which was to be returned to the search method mentioned in Figure 35.

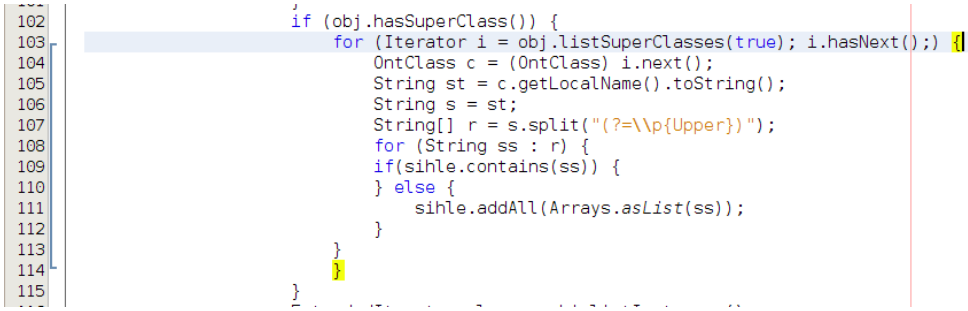

Figure 37: Search SUMO Ontology Method

The above snipped code in Figure 37 is also part of the search ontology method. The code reads the superclasses if the searched keyword has them. The superclasses were returned as the joined words, and were also separated using the upper case delimiter as they were joined together with the start of the other word shown by the upper case letter.

The following snipped code is the code for retrieving the instances of the searched keyword. It starts by checking if the searched keyword has the instances, if does it then it starts retrieving them. The delimiters which were used in processing the instances were different from those which were used for superclasses and sub-classes. The delimiters to be used were decided based on the ones which were noticed during the implementation of this method. The delimiters which were used are shown in Figure 38, line 120. The instances, sub-classes and superclasses were all temporarily saved in one array, line 125, the array which was used to store the retrieved data from the ontology.

The line codes in Figure 36, line 86 and Figure 38, line 129 and line 130 were used to control the searching of the ontology, in such a manner that when the searched data was found and retrieved, the searching of the ontology should be stop. These lines of codes where needed because the searching processes would continue till the end of the SUMO OWL file.

```
116                     ExtendedIterator clase = obj.listInstances();
117                     while (clase.hasNext()) {
118                         Individual inst = (Individual) clase.next();
119                         String s = inst.getLocalName().toString();
120                         String delimis = "[- _ ]+";
121                         String[] arr = s.split(delimis);
122                         for (String ss : arr) {
123                             if(sihle.contains(ss)) {
124                             } else {
125                                 sihle.addAll(Arrays.asList(ss));
126                             }
127                         }
128                     }
129                     found = true;
130                     break;
131                 }
132             }
133         } catch (Exception e) {
134             System.out.println(e.getMessage());
135         }
136         return sihle;
137     }
```

Figure 38: Search SUMO Ontology Method

The created array is then returned when the search ontology method is called, line 136 Figure 37.

The following snipped code in Figure 39, is the part of the search() method in Figure 34 which is used for content matching and for storing the words which are found matching. The created array is then used for frequency analysis, which is done in Figure 42.

```
163             for (int i = 0; i < words.size(); i++) {
164                 boolean found = false;
165                 for (int j = 0; j < stopWords.size(); j++) {
166                     if (words.get(i).equalsIgnoreCase(stopWords.get(j))) {
167                         found = true;
168                         break;
169                     }
170                 }
171                 if (found) {
172                     wrds.add(words.get(i));
173                 }
174             }
```

Figure 39: Content Matching

The following snipped code is the very last part of the search method in Figure 35 which was implemented for word frequency analysis and saving data to the array which will later be returned by the search method for results visualization.

```
176             Word[] frequency = new SMnager(searchWord).getFrequentWords(wrds);
177             for (int i = 0; i < frequency.length && i < 10; i++) {
178                 Word w = frequency[i];
179                 ResultsBean resultBean = new ResultsBean();
180                 resultBean.setTitle(w.word);
181                 resultBean.setContent(w.count);
182                 searchResult.add(resultBean);
183             }
184         } catch (IOException ex) {
185         }
186         return searchResult;
187     }
188 }
```

Figure 40: Search Method Adding results to the final array and returning it

This snipped code uses the "Word class" for keeping track of the words in the array, by making sure that the words get their actual frequency values. This works in conjunction with method described in Figure 42.

```
42  public class Word implements Comparable<Word> {
43      String word;
44      int count;
45      public Word(String word, int count) {
46          this.word = word;
47          this.count = count;
48      }
49      @Override
51      public int compareTo(Word otherWord) {
52          if (this.count == otherWord.count) {
53              return this.word.compareTo(otherWord.word);
54          }
55          return otherWord.count - this.count;
56      }
    }
```

Figure 41: Word Class for Comparing Words in the array

The word class in Figure 41 helps the method in Figure 42 by keeping track of the already calculated frequency and their corresponding words. This class is called in line 59 Figure 42 and used to keep track of the words and their corresponding frequencies in line 61 Figure 42. The frequencies of the words are then calculated and the results are then returned to the searched method line 176, Figure 40.

```
58      public Word[] getFrequentWords(ArrayList<String> words) {
59          HashMap<String, Word> map = new HashMap<String, Word>();
60          for (String s : words) {
61              Word w = map.get(s);
62              if (w == null) {
63                  w = new Word(s, 1);
64              } else {
65                  w.count++;
66              }
67              map.put(s, w);
68          }
69          Word[] list = map.values().toArray(new Word[]{});
70          Arrays.sort(list);
71          return list;
72      }
```

Figure 42: Frequency Analysis method

The array is sorted in line 70, Figure 42; this feature of this method helped in limiting the number of words we wanted to visualize to the end-user because results are sorted in a descending order with the word with the higher frequency occurring as the first element in the array. The array in line 71 Figure 42 is returned whenever this method is called. Line 177 Figure 40 shows that we were only interested in the ten most occurring words with higher frequencies.

The array list that was returned from the frequency analysis method in Figure 42 and contained two different data types, strings which were the actual words and the values of their frequencies as integers.

```
11  public class ResultsBean {
12      private String title;
13
14      private int content;
15
16      public String getTitle() {
17          return title;
18      }
19
20      public void setTitle(String title) {
21          this.title = title;
22      }
23
24      public int getContent() {
25          return content;
26      }
27
28      public void setContent(int content) {
29          this.content = content;
30      }
31  }
```

Figure 43: Java Beans for Frequency Analysis Results

When putting and retrieving the data on the final array list the above class in Figure 43 of java beans was used. The array was then returned to the middleware, the search controller as shown in

Figure 32, line 32. The following snipped JSP code shows how we were reading the results from middleware as described in Figure 32.

```
154                         <%
155                             List searchResult = (List) request.getAttribute("searchResult");
156                             ArrayList<Integer> val = new ArrayList<Integer>();
157                             ArrayList<String> terms = new ArrayList<String>();
158                             int resultCount = 0;
159                             if (null != searchResult) {
160                                 resultCount = searchResult.size();
161                             }
162                             for (int i = 0; i < resultCount; i++) {
163                                 ResultsBean resultBean = (ResultsBean) searchResult.get(i);
164                                 String title = resultBean.getTitle();
165                                 int cont = resultBean.getContent();
166                                 terms.add(title);
167                                 val.add(cont);
168                         %>
```

Figure 44: Read Results from Search Controller Component

The results from the middleware were then read into two different arrays, one for integers and one for strings in lines 156 and 157 respectively. The elements of theses arrays were then used to create the bar graph for data visualization. The snipped code in Figure 45 was used to create a bar graph on top of Jfreechart java library.

```
173                         <%
174                             if (resultCount >= 10) {
175                                 try {
176                                     OutputStream out2 = response.getOutputStream();
177                                     DefaultCategoryDataset dataset = new DefaultCategoryDataset();
178
179                                     dataset.setValue(val.get(0), "Most Occuring Words", terms.get(0));
180                                     dataset.setValue(val.get(1), "Most Occuring Words", terms.get(1));
181                                     dataset.setValue(val.get(2), "Most Occuring Words", terms.get(2));
182                                     dataset.setValue(val.get(3), "Most Occuring Words", terms.get(3));
183                                     dataset.setValue(val.get(4), "Most Occuring Words", terms.get(4));
184                                     dataset.setValue(val.get(5), "Most Occuring Words", terms.get(5));
185                                     dataset.setValue(val.get(6), "Most Occuring Words", terms.get(6));
186                                     dataset.setValue(val.get(7), "Most Occuring Words", terms.get(7));
187                                     dataset.setValue(val.get(8), "Most Occuring Words", terms.get(8));
188                                     dataset.setValue(val.get(9), "Most Occuring Words", terms.get(9));
189
190                                     JFreeChart chart = ChartFactory.createBarChart("Keyword searching, Content Matching and Freque
191                                             "Found Matching Words", " Exact number of thier Frequency",
192                                             dataset, PlotOrientation.VERTICAL, true, true, true);
193                                     chart.setBackgroundPaint(Color.white);
194                                     response.setContentType("image/png");
195                                     ChartUtilities.writeChartAsPNG(out2, chart, 1500, 450);
196                                     out2.close();
197                                 } catch (Exception e) {
198                                 }
199
200                             }
201                         %>
```

Figure 45: Create Bar Graph for Frequency Analysis Results

In this sub-section we have discussed the internal development of the keyword searching, content matching and frequency analysis sub-module. Its outer interface will be discussed in the UI implementation section. In the following sub-section we will discuss the internal development of keyword searching, content matching and correlation analysis.

Keyword Searching, Content Matching and Correlation Analysis

This module like the previously described index searching module accepts the keyword to be searched and starts the searching process. The difference with this module is that it does not use SUMO for content matching and frequency analysis. This module as aforementioned accepts the searched keyword and searches the index and retrieves all the content that matches the searched keyword.

The code in Figure 46 is the middleware which controls the searching of this sub-module like the one described in Figure 32. This also accepts the end-user query and passes it to the search manager class which is the class that does most of the business logic of this module.

```
17  @WebServlet(name = "SearchController", urlPatterns = {"/SearchController"})
18  public class SearchController extends HttpServlet {
19
20      private static final long serialVersionUID = 1L;
21
22      @Override
23      public void doPost(HttpServletRequest request, HttpServletResponse response)
24              throws IOException, ServletException {
25          String searchWord = request.getParameter("searchWord");
26          SearchManager searchManager = new SearchManager(searchWord);
27          List searchResult = searchManager.search();
28          RequestDispatcher dispatcher = request.getRequestDispatcher("Search.jsp");
29          request.setAttribute("searchResult", searchResult);
30          dispatcher.forward(request, response);
31      }
32
33      @Override
34      public void doGet(HttpServletRequest request, HttpServletResponse response)
35              throws IOException, ServletException {
36          doPost(request, response);
37      }
38  }
```

Figure 46: Keyword Searching, Content Matching and Correlation Analysis Search Controller Class

The snipped code in Figure 47 is the constructor used to initiate the search manger class. This constructor is then used in line 26 Figure 46 to initiate the search manager class. After the search manager class has been initiated, the search() method which is used to search and perform the business logic of this module is also initiated in line 27, Figure 46.

```
32
33      public SearchManager(String searchWord) throws IOException {
34          this.searchWord = searchWord;
35          this.indexManager = new IndexManager();
36          this.analyzer = new StandardAnalyzer(Version.LUCENE_42);
37      }
38
```

Figure 47: Constructor Method

The constructor also initiates the index manager class which is used to define and manage the path to the index created by the Facebook Crawling, Text Extraction and Text Indexing module. The following class in Figure 48 is used to define and keep the path to the created index.

```
11  public class IndexManager {
12      private final FSDirectory indexDir;
13      public IndexManager() throws IOException {
14          this.indexDir = FSDirectory.open(new File("/home/mfenyanasi/Documents/index3"));
15      }
16      public FSDirectory getIndexDir() {
17          return this.indexDir;
18      }
19  }
```

Figure 48: Index Manager Class

After the keyword has been accepted by the search manager class and the index manager class has been accepted the search() method in Figure 49 is then called to perform the actual index searching and the data processing and analysis.

This method is like a list data type method that returns results on a list array. In The snipped code in Figure 49 line 44 is used to open the index, using the method defined in index manager class defined in Figure 48 "getIndexDir" method. The snipped code in Figure 49 also defines the QueryParser method which is used to define the fields to be searched from the index in line 49. The search() method is then assigned the keyword to be used for searching the index in line 53, Figure 49.

```
39      public List search() {
40          List searchResult = new ArrayList();
41          IndexSearcher indexSearcher = null;
42          IndexReader reader = null;
43          try {
44              reader = DirectoryReader.open(indexManager.getIndexDir());
45              indexSearcher = new IndexSearcher(reader);
46          } catch (IOException ioe) {
47              System.out.println(ioe.getMessage());
48          }
49          QueryParser queryParser = new MultiFieldQueryParser(Version.LUCENE_42, new String[]{"content", "title"}, analyzer);
50
51          Query query = null;
52          try {
53              query = queryParser.parse(searchWord);
54          } catch (ParseException e) {
55              System.out.println(e.getMessage());
56          }
```

Figure 49: Search Method

The keyword is then used to search the index in line 75, Figure 50. Line 50 also specifies that the query should be searched against the entire index and should retrieve all the content that is found matching the searched keyword from the index.

The retrieved data is then tokenized in line 79 using the delimiters which are defined in line 70 Figure 50. The delimiters where chosen based from the few that were noticed from the data that we obtained from the Facebook servers.

```
69          ArrayList<String> wrds = new ArrayList<String>();
70          String dil = "[- _ . * '' @ : ; ,]+";
71          List<String> words = new ArrayList<String>();
72          if (null != query && null != indexSearcher) {
73              try {
74                  indexSearcher.search(query, collector);
                    TopDocs docs = indexSearcher.search(query, reader.maxDoc());
76                  for (ScoreDoc scoredoc : docs.scoreDocs) {
77                      Document doc = indexSearcher.doc(scoredoc.doc);
78                      try {
79                          StringTokenizer st = new StringTokenizer(doc.get("content"), dil
80                          System.out.println(doc.get("content"));
81                          String str = (String) st.nextElement();
82                          words.add(str);
83                      } catch (Exception e1) {
84                      }
85                  }
86                  reader.close();
87              } catch (IOException e) {
88                  System.out.println(e.getMessage());
89              }
90          }
```

Figure 50: Index Searching and Data retrieval

The tokens are then stored in an array in array line 82, which is declared in line 69 in the above figure. As previously stated in Chapter three, there was a need to remove stop-words as these are the words which almost occur in every statement and because the correlation in this thesis is analyzed based on the frequencies of the words that were found reoccurring from the content retrieved matching the searched keyword. Figure 51 are the few stop-words which were used as train dataset for testing the functionality of this module.

```
61          List<String> stopWords = Arrays.asList("can't", "cannot", "couldn't", "didn't", "do", "doesn't",
62                  "doing", "don't", "during", "had", "having", "he", "he'd", "did", "he's", "her", "here", "here's", "hers",
63                  "herself", "him", "himself", "his", "how", "i'd", "i'll", "if", "in", "into", "itself", "me", "most",
64                  "my", "That's", "ought", "the", "themselves", "then", "there's", "this", "we", "hy", "were", "weren't", "what",
65                  "what's", "is", "when", "If", "we", "mh", "A", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "m", "n", "m", "o
66                  "p", "r", "s", "t", "u", "w", "v", "x", "Y", "z","i'm");
67
```

Figure 51: Stop - Words

The following snipped code Figure 52 is used for content matching and the removal of the found stop-words. The words that are not found as the stop-words are then stored in an array in Line 100, Figure 52, which will be used to calculate the frequencies of the words from the content retrieved because it is matching the searched keyword.

```
91          for (int i = 0; i < words.size(); i++) {
92              boolean found = true;
93              for (int j = 0; j < stopWords.size(); j++) {
94                  if (words.get(i).equalsIgnoreCase(stopWords.get(j))) {
95                      found = false;
96                      break;
97                  }
98              }
99              if (found) {
100                 wrds.add(words.get(i));
101             }
102         }
```

Figure 52: Content Matching

The following snipped code in Figure 53 is the very last part of the search() method in Figure 49 and is used for calculating the frequencies of the words from the array created in the code described in Figure 52 line 100. The code in Figure 53 also puts the results of word frequencies and the actual words to the final array that will be returned by the search() method.

```
103         Set<String> unique = new HashSet<String>(wrds);
104         for (String key : unique) {
105             SearchResultBean resultBean = new SearchResultBean();
106             resultBean.setTitle(Collections.frequency(wrds, key));
107             resultBean.setContent(key);
108             searchResult.add(resultBean);
109         }
110         return searchResult;
111     }
```

Figure 53: frequency Analysis

The adding of the frequencies of the words is done in line 106 and the adding of the words is done in line 107, Figure 53. This is all done using the Java Beans class described in Figure 54. This class is also used for reading the created array for data visualization. The search() method then returns the results in line 110 Figure 53 to the middleware defined in Figure 46.

```
7   public class SearchResultBean {
8       private int title;
9
10      private String content;
11
12      public int getTitle() {
13          return title;
14      }
15
16      public void setTitle(int title) {
17          this.title = title;
18      }
19
20      public String getContent() {
21          return content;
22      }
23
24      public void setContent(String content) {
25          this.content = content;
26      }
27
28  }
```

Figure 54: Class for Putting and Retrieving Data the Final array

The following snipped code in Figure 55, is used to read data from the middleware component in Figure 46. This code reads data into two different arrays, one for the actual frequencies which is of integer data type and the other for the actual words which as of string data type as shown in lines 33 and line 34 respectively.

```
30                  <%
31                      List searchResult = (List) request.getAttribute("searchResult");
32                      int resultCount = 0;
33                      ArrayList<Integer> val = new ArrayList<Integer>();
34                      ArrayList<String> terms = new ArrayList<String>();
35                      if (null != searchResult) {
36                          resultCount = searchResult.size();
37
38                      }
39
40                      int[] rand = new int[resultCount];
41
42                      for (int i = 0; i < resultCount; i++) {
43                          SearchResultBean resultBean = (SearchResultBean) searchResult.get(i);
44                          int title = resultBean.getTitle();
45                          String cont = resultBean.getContent();
46                          terms.add(cont);
47                          val.add(title);
48                          rand[i] = (int) (Math.random() * resultCount);
49
50                  %>
51                  <%                      }
52                  %>
```

Figure 55: Read Results from Search Controller Component

To visualize the data from these arrays Jfreechart scatter plot was used. The scatter plot was used because it was impossible to predict the number of words that could be correlated to the searched keywords and had the same values of frequencies. The creation of the scatter plot requires the XY dataset, and from our data we only had frequency values which in this case are Y dependent

values. For X, the independent values we generated were random numbers that are in a range of the size of the array from the middleware component, in line 40 Figure 55.

```
52                  %>
53                  <%
54                      if (resultCount >= 1) {
55                          try {
56                              OutputStream out2 = response.getOutputStream();
57                              XYSeries series = null;
58                              XYSeriesCollection xySeriesCollection = new XYSeriesCollection();
59                              for (int i = 0; i < terms.size(); i++) {
60                                  for (int x = 0; x < val.size(); x++) {
61                                      series = new XYSeries(terms.get(i));
62                                  }
63                                  series.add(rand[i], val.get(i));
64                                  xySeriesCollection.addSeries(series);
65                              }
66                              JFreeChart chart = ChartFactory.createScatterPlot("Most Occuring Words Search Results",
67                                      "Most Occuring Words", " Exact number of Occurances", xySeriesCollection,
68                                      PlotOrientation.VERTICAL, true, true, true);
69                              XYPlot plot = (XYPlot) chart.getPlot();
70                              plot.setDomainCrosshairVisible(true);
71                              plot.setRangeCrosshairVisible(true);
72                              chart.setBackgroundPaint(Color.white);
73                              response.setContentType("image/png");
74                              ChartUtilities.writeChartAsPNG(out2, chart, 1000, 350);
75                              out2.close();
76                          } catch (Exception e) {
77                          }
78                      }
79                  %>
```

Figure 56: Create Scatter Plot Graph for Correlation Analysis Results

The above snipped code in Figure 56 is used for creating the scatter plot for visualizing the correlation results. The values for creating the plot graph are assigned in line 62 and add to XYSeries in line 63. The scatter plot is then created in line 65 using the Jfreechart java library. The following sub-module to be discussed is the keyword searching module.

Keyword Searching

This sub-module is basically used for searching the actual content that is indexed from Facebook severs. This module is different from the two previously discussed sub-modules for index searching module in the sense that it only retrieves the top ten statuses and/or comments and does not perform any text preprocessing and data analysis. The similarity is its structure, as it also has the search controller described in Figure 57, which is also meant for accepting the keyword searched from the end-user and passes it to the search manager class which does all the business logic of this module.

```
21  @WebServlet(name = "Controller", urlPatterns = {"/Controller"})
22  public class Controller extends HttpServlet {
23
24      @Override
25      public void doPost(HttpServletRequest request, HttpServletResponse response)
26              throws IOException, ServletException {
27          String searchWord = request.getParameter("searchWord");
28          ManageSearching searchManager = new ManageSearching(searchWord);
29          List searchResult = searchManager.search();
30          RequestDispatcher dispatcher = request.getRequestDispatcher("KeyWordSearch.jsp");
31          request.setAttribute("searchResult", searchResult);
32          dispatcher.forward(request, response);
33      }
34
35      @Override
36      public void doGet(HttpServletRequest request, HttpServletResponse response)
37              throws IOException, ServletException {
38          doPost(request, response);
39      }
40  }
```

Figure 57: Search Controller

The constructor is used for accepting the searched keyword, initialize the index manager class and also initiate the search manger class by in invoking the search() method which does all the searching process and data processing. The following snipped code in Figure 58 is the constructor method used by this sub-module.

```
29      public ManageSearching(String searchWord) throws IOException {
30          this.searchWord = searchWord;
31          this.indexManager = new ManagerIndex();
32          this.analyzer = new StandardAnalyzer(Version.LUCENE_42);
33      }
```

Figure 58: Constructor Method

The following class in Figure 59 is the index manager class that is used to define and manage the path to the directory where the index is created and stored.

```
13  public class ManagerIndex {
14    private final FSDirectory indexDir;
15      public ManagerIndex() throws IOException {
16          this.indexDir = FSDirectory.open(new File("/home/mfenyanasi/Documents/index3"));
17      }
18      public FSDirectory getIndexDir() {
19          return this.indexDir;
20      }
21  }
```

Figure 59: Index Manager Class

The following snipped code in Figure 60, is the first part of the search(). Line 45 is used to open the index and assign the actual keyword to be searched as a query to be used to search the index which is done in line 49.

```
37      public List search() {
38          List searchResult = new ArrayList();
39          IndexSearcher indexSearcher = null;
40          try {
41              IndexReader reader = DirectoryReader.open(indexManager.getIndexDir());
42              indexSearcher = new IndexSearcher(reader);
43          } catch (IOException ioe) {
44              System.out.println(ioe.getMessage());
45          }
46          QueryParser queryParser = new QueryParser(Version.LUCENE_42, "content", analyzer);
47          Query query = null;
48          try {
49              query = queryParser.parse(searchWord);
50          } catch (ParseException e) {
51              System.out.println(e.getMessage());
52          }
```

Figure 60: Search Method

The following snipped code in Figure 61, is the last part of the search() method for keyword searching sub-module. This snippet code is basically used to retrieve the top ten statuses and/or comments that match the searched keyword from Lucene index using TopScoreDocCollector method. Line 52 in Figure 61 defines the number of pages to be retrieved and TopScoreDocCollector class is used in line 53. This class retrieves the documents with top scoring hits and limit the number of the retrieved documents based on the number specified in line 52 Figure 61.

```
52          int hitsPerPage = 10;
53          TopScoreDocCollector collector = TopScoreDocCollector.create(hitsPerPage, true);
54
55          if (null != query && null != indexSearcher) {
56              try {
57
58                  indexSearcher.search(query, collector);
59                  ScoreDoc[] hits = collector.topDocs().scoreDocs;
60                  for (ScoreDoc hit : hits) {
61                      int docId = hit.doc;
62                      Document d = indexSearcher.doc(docId);
63                      Beans resultBean = new Beans();
64                      resultBean.setTitle(d.get("title"));
65                      resultBean.setContent(d.get("content"));
66                      searchResult.add(resultBean);
67                  }
68              } catch (IOException e) {
69                  System.out.println(e.getMessage());
70              }
71          }
72          return searchResult;
73      }
74  }
```

Figure 61: Search Method

The top ten documents which are retrieved are then stored in array which will be returned by the search() method of this sub-module to the search controller component described in Figure 57. The above snippet code used the class defined in Figure 62, to add the data to the final array that will be returned to the search controller component in Figure 57. This snippet code returns the actual retrieved content to search controller component class using line 72 in Figure 62.

The class in Figure 62 is also used to retrieve the data that is stored in the array create Figure 61.

```
13  public class Beans {
14      private String title;
15
16      private String content;
17
18      public String getTitle() {
19          return title;
20      }
21
22      public void setTitle(String title) {
23          this.title = title;
24      }
25
26      public String getContent() {
27          return content;
28      }
29
30      public void setContent(String content) {
31          this.content = content;
32      }
33
34  }
```

Figure 62: Java Beans

The data that is retrieved from the index is also needed to be displayed to the user-interface so that it can be seen and read by the end-users. The following snipped code in Figure 63 is used to retrieve data from the search controller method using line 112. The data is then displayed to the end-user using line 123 Figure 63.

```
111                     <%
112                         List searchResult = (List) request.getAttribute("searchResult");
113                         int resultCount = 0;
114                         if (null != searchResult) {
115                             resultCount = searchResult.size();
116                         }
117                         for (int i = 0; i < resultCount; i++) {
118                             Beans resultBean = (Beans) searchResult.get(i);
119                             String title = resultBean.getTitle();
120                             String cont = resultBean.getContent();
121                     %>
122                     <TR>
123                         <TD class="title"><h3><A href="<%=title%>"><%=cont%></A></h3></TD>
124
125                     </TR>
126                     <tr><td><hr /></td></tr>
127                             <%
128                                 }
129                             %>
```

Figure 63: Read Results from Search Controller and Display the Results

In this sub-section we have discussed the internal implementation of the keyword searching sub-module. This module as it has been observed was just meant to retrieve the actual data that is stored in the created index. This was developed so that the end-users could read and get the sentiments of the statuses and/or comments written by Facebook users in regard to any subject

they would be talking about. The following sub-section will discuss the content matching and frequency analysis.

Content Matching and Frequency Analysis

This is the last sub-module for indexing searching modules. This sub-module is different from the other three sub-modules which have been described in previous sub-sections above. This sub-module does not require the keyword to search the index. It only requires the end-user to press a button to initiate it. The development of this module was to add a feature to our system which will enable the end-users to be able to just check the trending words from Facebook without having to think of what to search.

The discussion of its internal development will follow the same precedence of starting with the search controller component shown in Figure 64. This sub-module unlike the other three sub-modules does not require a keyword to search the index. Line 28 in Figure 64 shows the constructor which is used to initiate the search manager class. The constructor does not require the keyword to be searched.

```
22  @WebServlet(name = "Controllller", urlPatterns = {"/Controllller"})
23  public class Controllller extends HttpServlet {
24    private static final long serialVersionUID = 1L;
25      @Override
26      public void doPost(HttpServletRequest request, HttpServletResponse response)
27              throws IOException, ServletException {
28          DManager searchManager = new DManager();
29          List searchResult = searchManager.search();
30          RequestDispatcher dispatcher = request.getRequestDispatcher("Trends.jsp");
31          request.setAttribute("searchResult", searchResult);
32          dispatcher.forward(request, response);
33      }
34      @Override
35      public void doGet(HttpServletRequest request, HttpServletResponse response)
36              throws IOException, ServletException {
37          doPost(request, response);
38      }
39  }
```

Figure 64: Search controller

The following snipped code in Figure 65 is the constructor which is used to initiate the index manger class and also invoke the search() method for this sub-module. This does not accept the keyword to be searched as it is an empty constructor.

```
30      public DManager() throws IOException {
31          this.indexManager = new MIndex();
32      }
```

Figure 65: constructor Method

The class in Figure 66 is the class which used for defining and managing the path of the index for this module.

```
17  public class MIndex {
18      private final FSDirectory indexDir;
19      public  MIndex() throws IOException {
20          this.indexDir = FSDirectory.open(new File("/home/mfenyanasi/Documents/index3"));
21      }
22      public FSDirectory getIndexDir() {
23          return this.indexDir;
24      }
25
26  }
```

Figure 66: Index Manger

The following snipped code in Figure 67 is the first part of the search() method for this sub-module. The array in line 69 is the train dataset which was used for system functional testing. These were the stop-words which were used as train dataset for removing stop-words that were to be found from the content which was retrieved from the index. The reason for removing stop-words is because they do not have sematic meaning and usually occur on every statement or document.

```
66      public List search() {
67          List searchResult = new ArrayList();
68
69          List<String> stopWords = Arrays.asList("can't", "cannot","lmao","mxm","and","ahh","aw","awuu","couldn't",
70                  "didn't", "do", "doesn't", "doing", "don't", "during","had", "having", "he", "he'd", "did", "he's",
71                  "her", "here", "here's", "hers","herself","him", "himself", "his", "how", "i'd", "i'll", "if","in",
72                  "into", "itself", "they","me", "more", "most", "my", "That's",  "ought","the", "themselves", "then",
73                  "therethere's",  "this", "we",  "were", "has","oh","weren't", "what", "what's", "is", "when","lol","i",
74                  "why","i'm","hey");
```

Figure 67: Search Method

The following snipped code in Figure 68, is part of the search() method for this sub-module. It is used for retrieving the entire content stored in the index to be processed by removing the stop-words and storing it to an array which will later be used for content matching and the selection of the words which would not be the stop-words. To retrieve all the data found in the index MatchAllDocsQuery of Lucene is used in line 85 in Figure 68. The data is then tokenized in line

90 using the delimiters defined in line 80. The data is then added to an array in line 92 which will be matched against the stop-words in Figure 67.

```
77
78          List<String> words = new ArrayList<String>();
79          ArrayList<String> wrds = new ArrayList<String>();
80          String delims = ", . ";
81
82          try {
83              IndexReader reader = DirectoryReader.open(indexManager.getIndexDir());
84              IndexSearcher indexSearcher = new IndexSearcher(reader);
85              Query query = new MatchAllDocsQuery();
86              TopDocs docs = indexSearcher.search(query, reader.maxDoc());
87              for (ScoreDoc scoredoc : docs.scoreDocs) {
88                  Document doc = indexSearcher.doc(scoredoc.doc);
89                  try {
90                      StringTokenizer st = new StringTokenizer(doc.get("content"), delims);
91                      String str = (String) st.nextElement();
92                      words.add(str);
93                  } catch (Exception e1) {
94                  }
95              }
96              reader.close();
```

Figure 68: Text Retrieval from Lucene Index

The following snipped code in Figure 69 is also part of the search() method of this sub-module. This snippet code is used for content matching and the selection of words that are not the stop-words and adds them to the new array which will be used to calculate the frequencies of the words found not to be the stop-words from the index content.

```
96              reader.close();
97              for (int i = 0; i < words.size(); i++) {
98                  boolean found = true;
99                  for (int j = 0; j < stopWords.size(); j++) {
100                     if (words.get(i).equalsIgnoreCase(stopWords.get(j))) {
101                         found = false;
102                         break;
103                     }
104                 }
105                 if (found) {
106                     wrds.add(words.get(i));
107                 }
108             }
```

Figure 69: Content Matching

The following snippet code in Figure 70 is the very last part of the search() method for this sub-module. This code is used to calculate the frequencies of the words from the array created in Figure 69. It also returns the results of the word frequencies and the actual words to the search controller component described in Figure 64.

```
110             Word[] frequency = new DManager().getFrequentWords(wrds);
111             for (int i = 0; i < frequency.length && i < 10; i++) {
112                 Word w = frequency[i];
113                 RBeans resultBean = new RBeans();
114                 resultBean.setTitle(w.word);
115                 resultBean.setContent(w.count);
116                 searchResult.add(resultBean);
117             }
118         } catch (IOException ex) {
119         }
120         return searchResult;
121     }
122 }
```

Figure 70: Adding Results to the Final Array and returning it to Search Controller

The following method in Figure 71 is the method used by the search() method in line 110 Figure 70 to calculate frequencies of the words from the array created in Figure 69. This method is used in conjunction with the class in Figure 72.

```
49
50      public Word[] getFrequentWords(ArrayList<String> words) {
51          HashMap<String, Word> map = new HashMap<String, Word>();
52          for (String s : words) {
53              Word w = map.get(s);
54              if (w == null) {
55                  w = new Word(s, 1);
56              } else {
57                  w.count++;
58              }
59              map.put(s, w);
60          }
61          Word[] list = map.values().toArray(new Word[]{});
62          Arrays.sort(list);
63          return list;
64      }
```

Figure 71: Frequency Analysis

The method described in Figure 71 accepts the array list of strings and then calculates their frequencies. The class in Figure 72 is used to keep track of the words which are already in the array and their corresponding frequencies. The frequencies calculated by the method in Figure 71 are then sorted in line 62 with the word with the highest frequency being the first element of the array that is being returned to the method in Figure 71.

```
34  public class Word implements Comparable<Word> {
35      String word;
36      int count;
37      public Word(String word, int count) {
38          this.word = word;
39          this.count = count;
40      }
41      @Override
43      public int compareTo(Word otherWord) {
44          if (this.count == otherWord.count) {
45              return this.word.compareTo(otherWord.word);
46          }
47          return otherWord.count - this.count;
48      }
    }
```

Figure 72: Word Class for Comparing words on an Array

When the calculation and the sorting of the words frequencies is returned to the search(), the search() method then adds the frequencies and the actual words in the final array that will be later returned to the search controller component described in Figure 64.

```
13  public class RBeans {
14      private String title;
15
16      private int content;
17
18      public String getTitle() {
19          return title;
20      }
21
22      public void setTitle(String title) {
23          this.title = title;
24      }
25
26      public int getContent() {
27          return content;
28      }
29
30      public void setContent( int content) {
31          this.content = content;
32      }
33
34  }
```

Figure 73: Java Beans

The class described in Figure 73 is also used for retrieving the data stored in the final array, for data visualization. The data in the search controller component is then read using the snippet code in Figure 74 line 15. The data is then read into two different arrays that are then used for data visualization by this sub-module. The data is read into two arrays one which is of string data in line 17 and the other which is the of the integer data type in line 15 Figure 74. The strings are the actual words from the index content and the integers are the actual frequencies of the words.

```
<%
    List searchResult = (List) request.getAttribute("searchResult");
    ArrayList<Integer> val = new ArrayList<Integer>();
    ArrayList<String> terms = new ArrayList<String>();
    int resultCount = 0;
    if (null != searchResult) {
        resultCount = searchResult.size();
    }
    for (int i = 0; i < resultCount; i++) {
        RBeans resultBean = (RBeans) searchResult.get(i);
        String title = resultBean.getTitle();
        int cont = resultBean.getContent();
        terms.add(title);
        val.add(cont);
%>
```

Figure 74: Reading the Results from the Search Controller Component

The data is then visualized using the Jfreechart bar graph. The snippet code in Figure 75 is used to create the bar graph. On the development of this module we decided to only visualize the top ten trending words with their actual frequency values. The bar graph for this sub-module is created using line 43 in Figure 75.

```
<%
    try {
        OutputStream out2 = response.getOutputStream();
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.setValue(val.get(0), "Most Occuring Words", terms.get(0));
        dataset.setValue(val.get(1), "Most Occuring Words", terms.get(1));
        dataset.setValue(val.get(2), "Most Occuring Words", terms.get(2));
        dataset.setValue(val.get(3), "Most Occuring Words", terms.get(3));
        dataset.setValue(val.get(4), "Most Occuring Words", terms.get(4));
        dataset.setValue(val.get(5), "Most Occuring Words", terms.get(5));
        dataset.setValue(val.get(6), "Most Occuring Words", terms.get(6));
        dataset.setValue(val.get(7), "Most Occuring Words", terms.get(7));
        dataset.setValue(val.get(8), "Most Occuring Words", terms.get(8));
        dataset.setValue(val.get(9), "Most Occuring Words", terms.get(9));
        JFreeChart chart = ChartFactory.createBarChart("Most Occuring Words Search Results",
                "Most Occuring Words", " Exact number of Occurances", dataset, PlotOrientation.VERTICAL, true, true, true);
        chart.setBackgroundPaint(Color.white);
        response.setContentType("image/png");
        ChartUtilities.writeChartAsPNG(out2, chart, 1000, 350);
        out2.close();
    } catch (Exception e) {
    }
%>
<%
    }
%>
```

Figure 75: Create the Bar Graph for Content Matching and Frequency Analysis

In this sub-section we have discussed the internal implementation of the content matching and frequency analysis sub-module. This was the last sub-module which was developed for index searching module and data analysis. The following section discusses the development of the system UI that is used to join the four sub-modules of the index searching module to form one system package.

User Interface

In this section we will discuss the development of the UI in Figure 24, in chapter three under high level system design. The index searching modules discussed in previous sub-sections have different UIs and search controller components. The interfaces are then called in one interface to form one system package.

The following snipped code in Figure 76 is used to call the Keyword searching, Content Matching and Correlation Analysis Interface. The search controller component is called in line 118. This interface also accepts the keyword to be searched in a form of text in line 123.

```
118  <FORM id=searchForm action=SearchController>
119      <TABLE>
120          <TBODY>
121              <TR>
122                  <TD colspan="3">
123                      <INPUT name=searchWord id=searchWord type=text size="40">
124                      <INPUT id=doSearch type=submit value= SearchTermsCorrelation>
125                  </TD>
126              </TR>
127          </TBODY>
128      </TABLE>
129  </FORM>
```

Figure 76: Keyword searching, Content Matching and Correlation Analysis Interface

The following snipped code in Figure 77 is used to call the Keyword searching, Content Matching and Correlation Analysis Interface. The search controller component is called in line 115. This interface also accepts the keyword to be searched in a form of text in line 110.

```
104
105  <FORM id=searchForm action=SControler>
106      <TABLE>
107          <TBODY>
108              <TR>
109                  <TD colspan="3">
110                      <INPUT name=searchWord id=searchWord type=text size="40">
111                      <INPUT id=doSearch type=submit value=SearchTrendingTerms>
112                  </TD>
113              </TR>
114          </TBODY>
115      </TABLE>
116  </FORM>
```

Figure 77: Keyword searching, Content Matching and Frequency Analysis Interface

The following snipped code in Figure 78 is used to call the Keyword searching interface. The search controller component is called in line 131. This interface also accepts the keyword to be searched in a form of text in line 136.

```
131  <FORM id=searchForm action=Controller>
132      <TABLE>
133          <TBODY>
134              <TR>
135                  <TD colspan="3">
136                      <INPUT name=searchWord id=searchWord type=text size="40">
137                      <INPUT id=doSearch type=submit value=KeyWordSearch>
138                  </TD>
139              </TR>
140          </TBODY>
141      </TABLE>
142  </FORM>
```

Figure 78: keyword Searching Interface

The following snipped code in Figure 79 is used to call the Content Matching and Frequency Analysis Interface. The search controller component is called in line 144. This interface also does not accept the keyword to be searched like the three previously described interfaces.

```
144  <FORM id=searchForm action=Controllller>
145      <TABLE>
146          <TBODY>
147              <TR>
148                  <TD colspan="3">
149                      <INPUT id=doSearch type=submit value=CheckTrendingTerms>
150                  </TD>
151              </TR>
152          </TBODY>
153      </TABLE>
154  </FORM>
```

Figure 79: Content Matching and Frequency Analysis Interface

Conclusion

The chapter started by introducing and discussing the implementation of the Facebook crawler, text extraction and the text indexing using Lucene java library. It continued by discussing the implementation of the index searching and modules and concluded by discussing the implementation of the user-interface. The next chapter will discuss the system testing process and present the experimental results.

Chapter Five: System Testing and Experimental Results

Introduction

The Chapter discusses the system functional testing and the experimental results. It starts by describing the testing of the system back-end that is the Facebook Crawling, Text Extraction and Text Indexing and also provides the results which were found. It then moves on to the testing of

the system's front end index searching modules and also gives a brief explanation of the system outputs.

Due to the limited data we could collect, these results serve only as a proof that the system does pass functional tests. We have used the following examples to illustrate to the reader that the system can be used in targeted opinion mining as well as general trend tracking. We discuss the difficulties encountered in gathering extensive data in chapter 6.

System Functional testing

Facebook Crawling, Text Extraction and Text Indexing Results

In this research project we were supposed to develop a system prototype for opinion monitoring and trend analysis. For this system prototype to work we needed to gather data from social networks, we chose to limit the scope of the project by only focusing on Facebook. The following diagram in Figure 80 is the structure and data format which we obtained from Facebook. The data as aforementioned in the previous chapter was in the JSON array and the comments were also in the nested array as can be observed in Figure 80.


```
{
 "data": [{
  "comments": {
   "data": [{
    "can_remove": false,
    "created_time": "2013-12-07T17:32:42+0000",
    "from": {
     "id": "100004387262990",
     "name": "Thobeka Charlotte"
    },
    "id": "1379152532336712_5686",
    "like_count": 0,
    "message": "Gud9yt u 2,we also lv u.",
    "user_likes": false
   }],
   "paging": {"cursors": {
    "after": "MQ==",
    "before": "MQ=="
   }}
  },
  "created_time": "2013-12-07T14:39:46+0000",
  "id": "100007259341154_1379152532336712",
  "message": "Good night people, just for you to know I love you all like serious........."
 }],
 "paging": {
  "next": "https://graph.facebook.com/100007259341154/feed?fields=message,created_time,comments.
  "previous": "https://graph.facebook.com/100007259341154/feed?fields=message,created_time,comme
 }
}
```


Figure 80: Facebook Data in JSON Format

The data in the in Figure 80 is the status and there is one comment which was posted on that status. There was need to parse the data to parsed to be a plain textual data for it to be able to be

indexed using Lucene java library. The following diagram Figure 81 is the textual data parsed from the data JSON data presented in Figure 80.

```
 Status   Good night people, just for you to know I love you all like serious.....
Comment   Gud9yt u 2,we also lv u.
```

Figure 81: Parsed Textual Data

The textual data was then indexed using Lucene index. The actual time this module takes to crawl, parse text and index textual data depends on the actual size of the content that is retrieved from the facebook database servers. The testing of the index functionality was then done through performing searching through the index searching modules described in the following section, Index Section.

Index Searching

In this section we discuss the visual results for our work, the results that the end-users will come in contact with. However, there are also a lot of technical results we achieved during this research project to make the system work as it should.

Keyword Searching, Content Matching and Frequency Analysis

Testing this module had some limitations such the data format from SUMO was totally different to the data that is likely to be found from Facebook. This was because of the writing styles and wrong spellings which are normally found from facebook and while SUMO uses correct spellings of the words.

Figure 82: Keyword Searching, Content Matching and Frequency Analysis

The other drawback which was noticed in using SUMO was that it was time consuming to try and found out about all text delimiters that they used in defining the words on SUMO. The last

drawback which was noticed in using SUMO is that it was taking time to search the ontology which jeopardized the response time of the system. In order to test the functionality of this module, we needed three people with Facebook user accounts to update their statuses and write some comments on them with some few words we were getting from the ontology. We then initiated our system to crawl Facebook, preprocess the data we were getting from Facebook and then index. We then performed the searching through this module to see the functionality of the system. We searched the word "Church" and the results from Figure 82 were found. The results suggest that it possible to use the ontologies for opinion monitoring and the trend analysis from the SNSs data.

Keyword Searching, Content Matching and Correlation Analysis

After the death of one of the greatest world icon, the former South Africa first black President Nelson Rolihlahla Mandela was announced on the 5th December 2013, it filtered all over the SNSs and the Internet. That in turn prompted us to know what people on the network from Facebook which we were using to test our system were saying about the death of Mr. Mandela.

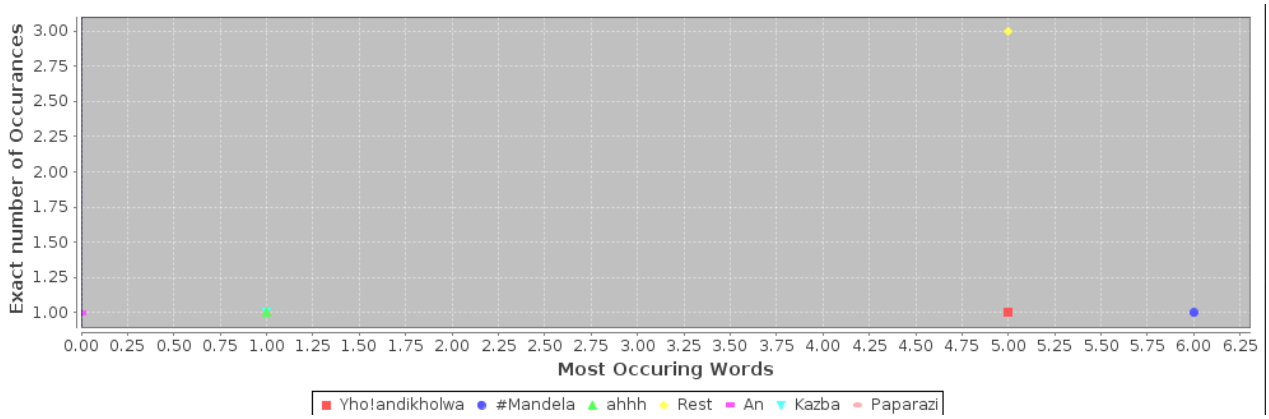

Figure 83: Keyword Searching, Content Matching and Correlation Analysis Results

On the 6th of December 2013, the day after the tragic of passing the South African president we crawled Facebook. We collected statuses and comments of the Facebook users which are in the network with the Facebook user account we were using in this research project for testing purposes. The reason gathering data from Facebook on such day, we wanted to see the trends about the passing of our former first black president. We then searched his surname "Mandela" to see the words which were mostly used in the statuses and comments that matched the searched keyword. The results of the searched keyword are visualized in Figure 83. The word "Rest" was

the word with the highest frequency as can be observed in Figure 83. This word was most occurring on the statuses and comments of Facebook users which were mourning about the passing of our former president.

Keyword Searching

The section discusses the keyword searching module. As aforementioned in Chapter 3 and 4, this module retrieves statuses and/or comments that match the searched keyword. It also limits the maximum number of the retrieved statuses and/or comments to ten based on their hits.

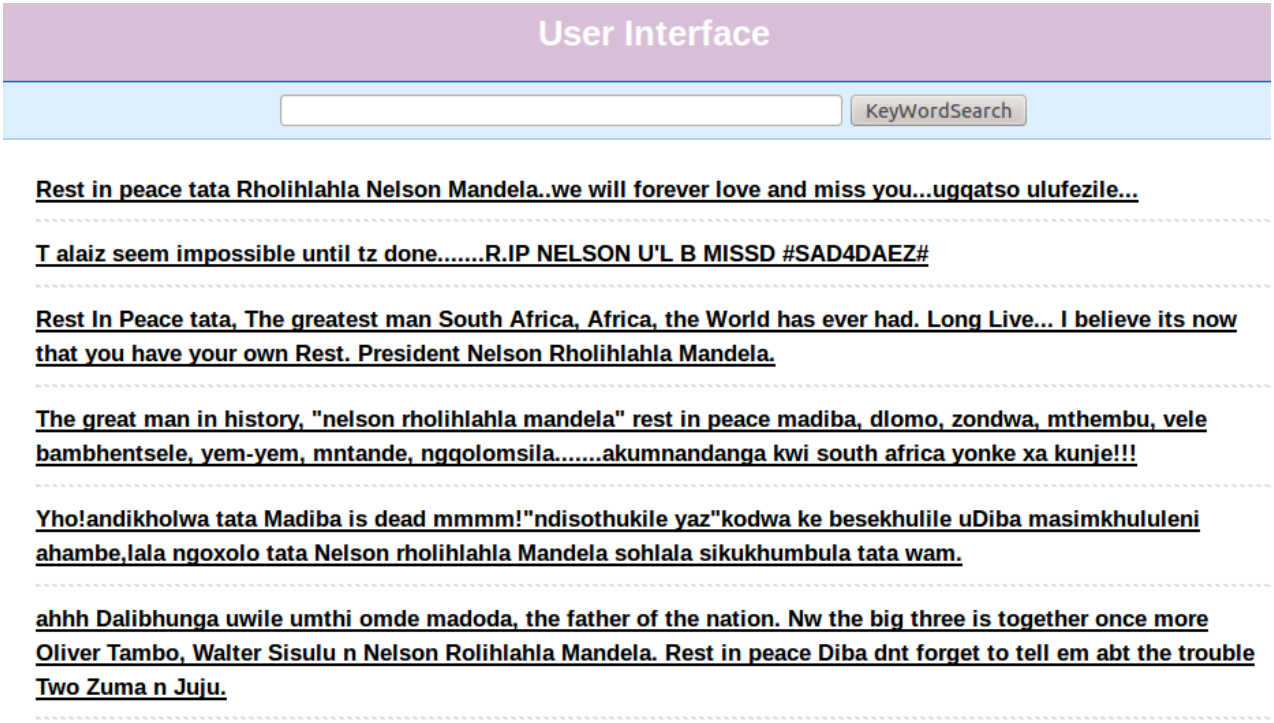

Figure 84: Keyword Searching Results

Figure 84 above presents the results which are the statuses and/or comments which were retrieved after we searched the keyword "Nelson" from our system. The retrieval of the actual statuses and comments feature in our system was developed so that end-users could get the actual sentiments of the content which is found matching the searched keyword. With the results displayed of the keyword searched, an end-user who knows IsiXhosa and English could simply understand that the people who wrote the statuses and comments in Figure 84 were actually expressing their condolences to the Mandela family. The data as aforementioned was collected on the 6th December 2013.

Content Matching and Frequency Analysis

The section presents the results of the last module in index searching modules. This module as aforementioned in Chapter four, the system implementation chapter, does not require the keyword search input for it to start searching the index. This module starts retrieving the data from the index and starts calculating the frequencies of the words found from the index. That implies that the trending words are not calculated based on the keyword searched but on the whole index.

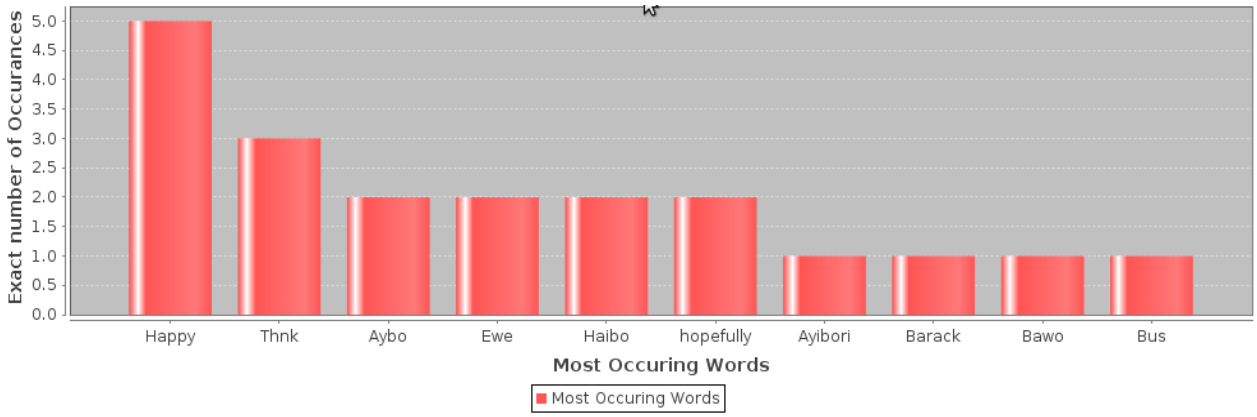

Figure 85: Content Matching and Frequency Analysis Results

The results in Figure 85 were the results of the content matching and frequency analysis found after we collected and indexed on 12 December 2013. These results were the words with higher frequencies, even though there were probably more words with frequencies of one but we limited our results to ten.

System Performance testing

According to [73] performance testing is the technical investigation done to determine the responsiveness, speed scalability characteristics of product under test. In this research we were mainly focusing on testing the responsiveness of our developed system. This was done so that we could determine if the end-users would be satisfied with the performance characteristics of the system.

As it has been noticed from the previous chapters, chapter 3 and chapter 4 the design and the implementation of the Index searching sub-modules were different. Their response times are also different. The response times of the three Index Searching sub-modules Keyword Searching,

Content Matching and Frequency Analysis, Keyword Searching and Content Matching and Correlation Analysis were all equal. The reason for their response times to be same is that they only search the index and analyze and visualize the data. Figure 86 shows the response time which was found after test modules were tested. The three sub-modules have the response time of second and that suggested that the end-users will not have to wait too long for the system to give back the feedback.

```
kwi zizwe ubungumzekelo
Nxeba elisentliziyweni lelona libuhlungu kuba kaloku alibethwa ngu moya
Idandathekile imiphefumlo yabantwana base Africa, zophukile intliziyo
Sithi thina nto zaziyo, sithi bekufanele, sithi engqondweni bekumele
Ifikile kaloku inqwelo yokufa nangelilixa, ayahamba ze nangoku
Kodwa isithathele ingqondi, intsika yesizwe

It's upon us to live and keep the legacy you fought and plawed upon us, alive

Aahh!!! DAlibhunga, Zondwa zintshaba, Yemyem, Sopitsho
Phumala ngo xolo Tata Mandela ulifezi olwakho ugqatso kulomhlaba umagada ahlabayo

Aahhhh Dalibhunga!!!!!

nde graaam!!!!
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 86: Index Searching Response Time

The fourth Index Searching sub-module Keyword Searching and Content Matching and Frequency Analysis as it was described in the previous chapters, chapter 3 and chapter 4 its design and the implementation is different to the rest of the other three Indexing Searching sub-modules. This sub-module start by searching the SUMO retrieve, preprocess data before the data could be used for content matching and frequency analysis. The searching of the SUMO takes little long when compared to searching the index. The response times when searching the SUMO is not constant as compared when searching the index. As it has been noticed it only takes one second to search and retrieve data from Lucene index. Figure 87 and Figure 88 Shows the response times we were getting when searching the SUMO.

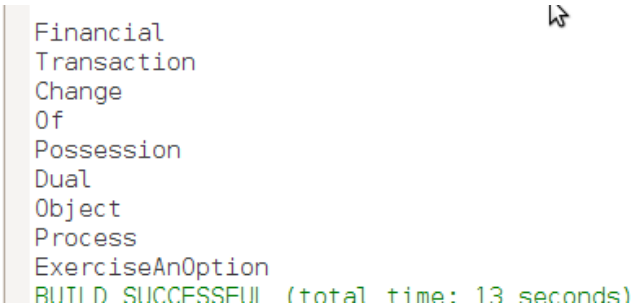

Figure 87: SUMO Response Time

In Figure 87 above is the response time we got when we were searching the "Transaction" from the SUMO.

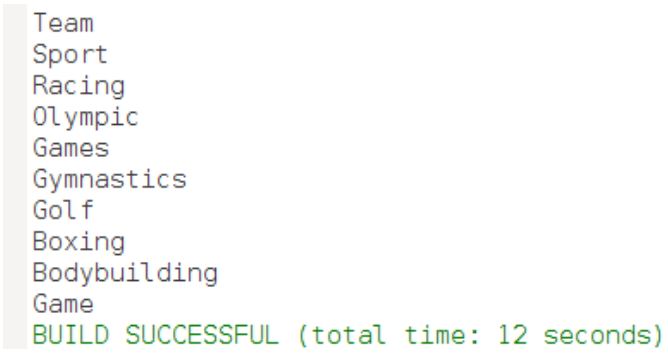

Figure 88: SUMO Response Time

In Figure 88 above is the response time we got when we were searching the word "Sport" from the SUMO.

The average response time when searching SUMO was 12.5 seconds when calculated from the response times we were getting during the time we were testing the our developed system. This response time compared to the response times of the other three Index Searching sub-modules is a bit too slow. This in turn means that end-users will have to wait for some time for the system to get the feedback when using this Index Searching sub-module. The overall system response time is less than one minute that means the response of time of the system will be good for end-users.

System Usability Testing

The usability testing of the system was performed at computer science masters lab. There were five participants which were involved in the usability system testing. Three participants were the

computer sciences masters students and the other two students where from other departments. The questions which were asked were ranked from strongly agree to strong disagree, strongly agree referring to positive response and strongly referring to negative answer while undecided referring to neutral response. The ranking of the questions was done so that we could find the sentiments of the participants towards the system developed in this research project.

The system was explained to the participants and the results each sub-module should return were explain. Each participant was then given time to use the system. When the participant was finished using the system were then asked to answer the questionnaire which was to be used to get their sentiments towards the system. The questions and the responses from the participants are shown in following figure, Figure 89.

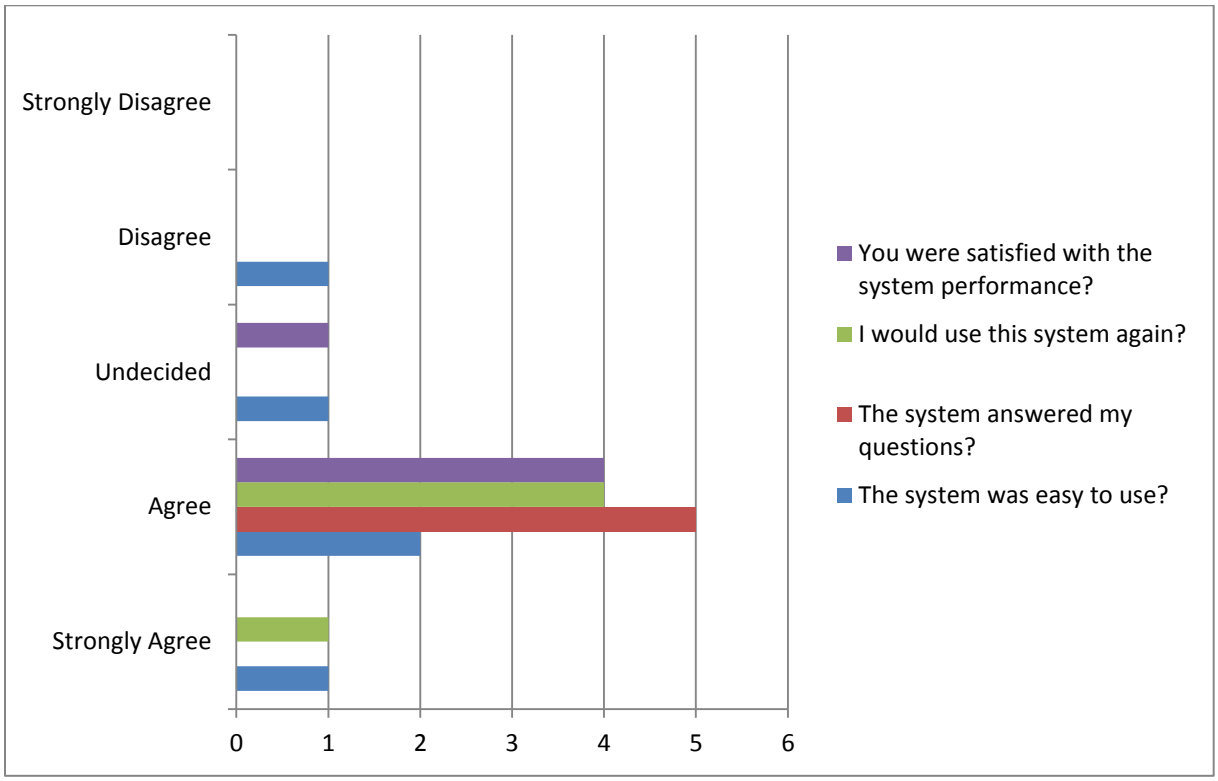

Figure 89: System Usability Testing Results

There was one participant who was not satisfied with the performance of the system and the other four participants agreed that they were satisfied with the performance of the system. All the participants agreed that they would love to use the system in future. They were also asked if the system was able to answer the queries they issuing and all the participants agreed that the system

was able to answer their queries. Participants were also asked if it was easy to use the system and four participants agreed that it was easy to use the system. There was one participant which could not decide about the easiness of the system. Based from the results which we got after the system usability testing we can assume that the system after deployment will be usable by the people of Dwesa.

Conclusion

In this chapter we discussed the system functional testing on which we were mainly focusing on the system outputs the system end-users will come across when they are using the system. Chapter then went to discuss the system performance testing and there we were mainly focusing on the system response time. The chapter concluded by the discussion of the usability which was meant to get the sentiments of the participants who were involved in this system usability testing.

Chapter: Discussion and Conclusion

Introduction

This chapter gives a summary of the thesis, it then go to discuss objectives achieved. The problems which were encountered during the course of this research are also discussed in this chapter. The chapter concludes by giving the recommendations of the possible extensions that can be added on the system.

Thesis Summary

There are many traditional ways of collecting data such as questionnaires, observations, formal and informal interviews etc. These, when used, have some drawbacks such as travel costs, data collection and data analysis which both require some expert knowledge and are also time consuming. The data in normal cases is collected and analyzed to answer some specific questions or to find some insights about some specific aspects about a certain population. It is a time consuming and costly process if for example, manufacturing companies can use these traditional methods to collect data from public with the objectives of finding out:

1. What people think about the product they produce;
2. What the public sentiments towards the product they are producing; and
3. What improvements could possibly be made to try and meet their clients' needs?

This research project proposed and developed a system prototype which could be used to mitigate the problems which normally come with the traditional ways of data collection and data analysis. Nowadays SNSs have become a popular communication tools among Internet users, where millions of users share opinions, views, thoughts, and ideas on different aspects of life almost every day. People communicate through these SNSs by positing content that makes more data about the public to be available on these SNSs. The developed system prototype is developed to be used by the Dwesa community local government officials for them to easily monitor opinions of the public and see the trending topics about the services they provide. The system is designed in such a manner that it could be used for many different applications, but in this research it was meant to be used for the government service delivery case study.

Objectives Achieved and Discussion

Based on the system implementation discussed in chapter 4 and the system testing discussed in chapter 4 perspectives, the developed system meets and satisfy the research objectives specified in chapter 1. The following were the specific objectives which were proposed in this research project:

Investigate and identify the key requirements for a Facebook Crawler that can be used for opinion monitoring and trend analysis purposes.

Through thorough literature review which was conducted on the research key requirements and the tools which could be used for developing the system of this nature were identified. Through literature we also able to best select the tools which used to develop the system that was proposed in this research.

Design, develop, implement and test a Facebook Crawler for Opinion Monitoring and Trend Analysis Purposes.

After extensive literature review were then able to design, develop and implement the proposed as that can be witnessed from chapter 3 and chapter 4.

Investigate general problems faced by rural communities when communicating with their local government.

This objective was achieved through data collection and data analysis. The findings of the data collected were discussed in chapter three, under feasibility Study.

Investigate the methods they are currently using to communicate with their government.

This objective was also addressed through the data collection and data analysis. The findings were discussed in chapter 3 under feasibility study.

In overall conclusion, all the objectives which were proposed in this research were fully satisfied. The following section will discuss the problems which were encountered during the course of this research.

Problems Encountered

From the onset of this research, there were some limitations which we faced. This section is meant to discuss some few limitations which were experienced during this research project.

University of Fort Hare's Network Security

This restricted us from using the university proxy server whenever we wanted to access and gather data from Facebook through Graph API. There was then a need for us to use the USB (Universal Serial Bus) Modem to access and gather data from Facebook. The use of a Modem led to a situation in which there was need to buy data bundles which was costly. However, the problem will not occur in Dwesa where the system will be deployed, because the network security system in the SLL is different to the one which is being used at the University of Fort Hare as the system was also tested there.

System Development

During the system development phase there were some few problems which were encountered.

Facebook Crawler

Not only was a problem of network security that was encountered during the development of the Facebook crawler which was to be used gather data from Facebook. There was also a problem of using the RestFB java library which was used to communicate with Facebook. There was a time where we planned to crawl user data feeds in for week, the idea was to crawl data feeds of week per each user. The idea was practically impossible to implement the system would crawl Facebook pass the date parameters we were setting. That would require more data to be used when crawling data feeds and it would take to finish. That led to situation where we had to change our system to only crawl only the latest statuses of the Facebook who were in the network of the count which was used in this research for test purposes. The other problem was experience when parsing text from JSON data to plain text using Jackson java library. This needed to be updated every time when there was a new feature added on Facebook. If the code that was used to parse text was not updated in accordance with updates Facebook the system would not run and reported error of undefined object. That suggested that system administrator would need training of how to keep updating the script that is used to parse text from JSON data format to plain. The problem only occurs when there is a new feature which is added on Facebook. The last problem on the development of the Facebook crawler, even though we were

specifying the parameters which were to specify the fields of JSON that wished to get, JSON would be returned with lot of that we did not need. That will results in situation where we had to write a very long script to parse textual data from JSON format.

Data Filtering

After the network security and text parsing problems were resolved and we were able to crawl Facebook and collect data, we began to experience problems of determining the delimiters which could be used to preprocess the textual data which was to be then used for term frequency analysis and correlation analysis.

The reason being there are many different stop-words used on Facebook and the people could write one in many different ways. An example would be the case where we wanted to remove the "lol", where it could be written as "looool" or "lool". The other problem was that people would keyboard special keys when writing the statuses or comments along with the text they would be posting. A practical could be observed from Figure 82 from the "Yho!andikholwa", the phrase in English means "I cannot believe" and "Yho" being just an expression of being startled.

Additionally, while we settled on the idea of using stop-words, we realize that encoding the exact words is a weakness in our development. In the future we would like to employ an intelligent algorithm which will study facebook status words and come up with a collection to frequently used stop words and their most common spelling. We also believe that perhaps instead of searching for stop-words, we could use target words. For example, given a user that is looking up "service delivery" we would look for words such as roads, water, schools, etc. We believe that this method can also be implemented through the use of Ontologies or other expert systems that use Semantic Webs.

SUMO Usage

The limitations were on the usage of the SUMO. The words on SUMO are mostly written in correct spellings while in Facebook, users mostly write words in different styles. This led to a situation in which it was difficult to find matching words from Facebook and SUMO which could be used to calculate term frequencies. The other problem was determining delimiters which could be used to process data from SUMO as it contained a lot of data and was time

consuming to try to learn all delimiters which could be possibly used for preprocessing the data. The other minor problem which was experienced on using the SUMO was the response time where searching it and the SUMO contains words in English. That in turn suggests that it will be difficult for non-English speakers to use this system sub-module. The last problem could be observed from Figure 81 the Index searching sub-module results. The problem with the results is the letter "A" which does not convey any meaning. The problem occurred because there was a word from SUMO which was having consecutive upper letters. When the data from SOMO was being the consecutive preprocessed that results that letter to being treated as of the words which were used for content matching and frequency analysis. As mentioned in previous chapters that upper case delimiter was also used to separate the joined words from SUMO, that means words with the consecutive upper letters, letters will be regarded as the stand alone words which be used for frequency analysis.

Data collection

During the period of collecting data, the following were the drawbacks from getting larger survey sample:

1. **Distance between University of Fort Hare (UFH) and research site (Dwesa)** – the distance between the UFH and Dwesa community restricted the author from conducting constant visits to the research site. The researcher would only visit the research site when there was a computer literacy training to be conducted during a specific week of every month. The literacy trainings were sometimes not conducted due to transport problems and the conditions of the Dwesa roads after some weather conditions which would restrict cars from travelling.
2. **Transport** – the transport did not only affect the researcher from constantly visiting the research site but also affected the Dwesa residents from reaching the schools where the trainings were usually conducted (Mpume and Nqabarha). The people who were attending the computer literacy trainings were from different regions of Dwesa and their regions were identified by the names of the schools from their regions. These are the schools which are involved in SLL initiative: Badi SS, Hlabizulu JSS, Kunene SPS, Lower Ngwane JPS, Lurwayizo JSS, Lurwayizo SS, Mevana JSS, Mpume JSS, Mthokwane JSS, Ngoma JSS, Ngqeza JSS, Ngwane JSS, Nondobo JSS, Nqabara SS, Nquba JSS, Ntubeni JSS and Zwelidumile SS. People were attending trainings at one of the two schools where the

trainings were usually held based on the distance they would travel; that meant the closest school in their region. The problem for some of them would be getting transport, a factor which resulted in some of them failing to attend the trainings due to the long distances which they had to travel. This was a significant problem for both the residents and the researcher during the data collection process.

3. **Weather conditions** – as prior stated, weather conditions were a challenge especially after heavy and/or steady rain falls because transport for the SLL research team and the residents who travelled with the cars used to restrict them from attending the trainings. The situation was worse for people who travelled on foot to and from the trainings, because it would be very cold.

Future Work

This section discusses the possible extensions which could be added on the developed system in this research. The following are some of the extension suggestions which could be added on the system presented on this thesis:

1. **Collect data from Twitter** – this research was mainly focusing on gathering data from one SNS, Facebook. In future, data could also be collected from Twitter as it is one of the SNSs in use today. This would allow the system to gather more data from these two different SNSs as it was noticed from the literature that Twitter is the most popular microblogging platform.
2. **Add a feature on our system interface which will allow the system to continuously display the content from the created index**- This feature will allow the end-users to easily read the index content without having to search, but if they find something which catches their interests, they can then search it through the same developed system in this research.
3. In literature there is a lot of work that has been focusing on sentiment analysis from Twitter textual, sentiment analysis could also add a good value in future on our work.

Overall Conclusion

In this thesis we have presented the research done and outlined the development of the proposed system that will probably improve the way the government and local people communicate. The

implementation of the facebook crawler for opinion monitoring and trend is a success story, although its deployment at Dwesa where the system is intended to be used still remains a future proposal and/or development. The success of the system when performing its actual main duty, which is to provide a cheap way of communication between government officials and the local people, will be determined by the reviews of the targeted end-users, after the system has been tested and deployed at Dwesa.

References

[1] "Project Activities | Siyakhula Living Lab." [Online]. Available: http://siyakhulall.org/?q=activities. [Accessed: 20-Sep-2013].

[2] P. Chiu, C. M. K. Cheung, and M. K. O. Lee, "Online Social Networks: Why Do 'We' Use Facebook?," *J. Mark. Res.*, vol. 19, pp. 67–74, 2008.

[3] C. M. K. Cheung, P. Chiu, and M. K. O. Lee, "Computers in Human Behavior Online social networks : Why do students use facebook ?," *Comput. Human Behav.*, vol. 27, no. 4, pp. 1337–1343, 2011.

[4] A. Nadkarni and S. G. Hofmann, "Why do people use Facebook?," *Pers. Individ. Dif.*, vol. 52, no. 3, pp. 243–249, 2012.

[5] M. Srivastava, "Social Media and Its Use by the Government," *J. Public Adm. Gov.*, vol. 3, no. 2, pp. 161–172, Jul. 2013.

[6] R. Bassett, T. Chamberlain, S. Cunningham, and G. Vidmar, "Data Mining and Social Networking Sites: Protecting Business Infrastructure and Beyond," *Data Min. Soc. Netw. Sites*, vol. Volume XI,, no. 1, pp. 352–357, 2010.

[7] E. Costa, R. Ferreira, P. Brito, I. I. Bittencourt, O. Holanda, and A. Machado, "Expert Systems with Applications A framework for building web mining applications in the world of blogs : A case study in product sentiment analysis," *Expert Syst. Appl.*, vol. 39, no. 5, pp. 4813–4834, 2012.

[8] "Batho Pele: Improving government service." [Online]. Available: http://www.etu.org.za/toolbox/docs/govern/bathopele.html. [Accessed: 18-Sep-2013].

[9] M. Thinyane, H. Slay, A. Terzoli, and P. Clayton, "A Preliminary Investigation into the Implementation of ICTs in Marginalized Communities," in *SATNAC*, 2006, p. No 213.

[10] B. T. Jakachira, "Implementing an integrated e-Government functionality for a marginalized community in the Eastern Cape , South Africa Master of Science in Computer Science University of Fort Hare by," no. November, 2009.

[11] S. Batsakis, E. G. M. Petrakis, and E. Milios, "Improving the performance of focused web crawlers," *Data Knowl. Eng.*, vol. 68, no. 10, pp. 1001–1013, 2009.

[12] C. House, Z. Streets, and I. Services, *Report on the Evaluation of the Implementation of the Batho Pele Principle of Consultation*, no. October. THE PUBLIC SERVICE COMMISSION (PSC) Commission House Cnr. Hamilton & Ziervogel Streets Arcadia 0083 Private Bag X121 Pretoria 0001, 2007.

[13] “Municipal Service Delivery.” [Online]. Available: http://www.etu.org.za/toolbox/docs/localgov/munservice.html. [Accessed: 18-Sep-2013].

[14] Farai Makombe, “Developing a help-desk system for a multi-purpose ICT platform in a marginalised setting Master of Science Computer Science University of Fort Hare By Farai Makombe,” no. January, 2011.

[15] N. B. Ellison, “Social Network Sites: Definition, History, and Scholarship,” *J. Comput. Commun.*

[16] K. G. Provan, a. Fish, and J. Sydow, “Interorganizational Networks at the Network Level: A Review of the Empirical Literature on Whole Networks,” *J. Manage.*, vol. 33, no. 3, pp. 479–516, Jun. 2007.

[17] K. Differences, G. Cormode, and B. Krishnamurthy, “Key Differences between Web1.0 and Web2.0,” pp. 1–30, 2008.

[18] Eb. Rank, “Top 15 Most Popular Social Networking Sites.” [Online]. Available: http://www.ebizmba.com/articles/social-networking-websites. [Accessed: 01-Oct-2013].

[19] F. G. C. Hwang Kai, Dongarra Jack, *Cloud Computing Over Cluster , Grid Computing : a Comparative Analysis*, vol. 1, no. 1. Morgan Kaufmann, 2011.

[20] C. Wilson, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao, “Beyond Social Graphs,” *ACM Trans. Web*, vol. 6, no. 4, pp. 1–31, Nov. 2012.

[21] E. Protalinski, “Facebook has over 845 million users | ZDNet.” [Online]. Available: http://www.zdnet.com/blog/facebook/facebook-has-over-845-million-users/8332. [Accessed: 01-Oct-2013].

[22] E. Kontopoulos, C. Berberidis, T. Dergiades, and N. Bassiliades, “Expert Systems with Applications Ontology-based sentiment analysis of twitter posts,” *Expert Syst. Appl.*, vol. 40, no. 10, pp. 4065–4074, 2013.

[23] K. Ivana and M. Mihić, “Business Intelligence: The Role of the Internet in Marketing Research and Business Decision-Making,” *UDC 65.012.34004.738.5*, pp. 69–86, 2010.

[24] Pamela Lewis Dolan, “10 tips to using LinkedIn - amednews.com,” 2011. [Online]. Available: http://www.amednews.com/article/20110725/business/307259969/4/. [Accessed: 01-Oct-2013].

[25] “The Value of LinkedIn – LinkedIn Overview Video | LinkedIn Help Center.” [Online]. Available: http://help.linkedin.com/app/answers/detail/a_id/45. [Accessed: 01-Oct-2013].

[26] P. Bhattacharyya and S. F. Wu, “Social Network Model based on Keyword Categorization,” in *International Conference on Advances in Social Network Analysis and Mining*, 2009.

[27] S. S. Dhenakaran and K. T. Sambanthan, “Web Crawler - An Overview,” vol. 2, no. 1, pp. 265–267, 2011.

[28] B. Pontes, S. Rocha, R. Luis, and T. Santos, “Example-Driven, Content-Based, Distributed Crawling and Extraction for Online Social Network Analysis,” 2006.

[29] J. L. Hao Wang, Richard Frost, “Social Network Crawling and Properties Analyzing Using Different Sampling Methods,” pp. 1–23, 2012.

[30] “Graph API.” [Online]. Available: https://developers.facebook.com/docs/reference/api/. [Accessed: 01-Oct-2013].

[31] P. Bedi, A. Thukral, and H. Banati, “Focused crawling of tagged web resources using ontology,” *Comput. Electr. Eng.*, Oct. 2012.

[32] J. P. G. Sharma Shruti, A.K.Sharma, “A Novel Architecture of a Parallel Web Crawler,” *Int. J. Comput. Appl.*, vol. 14, no. 4, pp. 38–42, 2011.

[33] G. Pant, P. Srinivasan, and F. Menczer, “Crawling the Web,” *springer*, pp. 153–177, 2004.

[34] U. Pisa, D. Informatica, and A. Signorini, “The Indexable Web is More than 11 . 5 billion pages Categories and Subject Descriptors,” pp. 5–6, 2005.

[35] A. Micarelli and F. Gasparetti, “Adaptive Focused Crawling,” pp. 231–262, 2007.

[36] F. Van De Maele, “Ontology-based Crawler for the Semantic Web Felix Van de Maele Ontology-based Crawler for the Semantic Web Felix Van de Maele,” no. May, 2006.

[37] M. M. Mostafa, “More than words : Social networks ’ text mining for consumer brand Avenida das Forças Armadas Lisbon , Portugal,” *Expert Syst. Appl.*, 2013.

[38] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, “Sourcerer: mining and searching internet-scale software repositories,” *Data Min. Knowl. Discov.*, vol. 18, no. 2, pp. 300–336, Oct. 2008.

[39] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*, Online edi., no. c. 2009.

[40] W. B. Croft, *Knowledge-based and statistical approaches to text retrieval*, vol. 8, no. 2. IEEE Computer Society, 1993, pp. 8–12.

[41] D. Hiemstra, *Using language models for information retrieval*. Copyright c 2000 Djoerd Hiemstra, Enschede, The Netherlands.

[42] D. Yalamanchi, "SIDEFFECTIVE - SYSTEM TO MINE PATIENT REVIEWS : SENTIMENT ANALYSIS Abstract of the Thesis Sideffective - System to Mine Patient Reviews : Sentiment Analysis," 2011.

[43] M. Mccandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action*, Second Edi. Manning Publications Co. 180 Broad St. Suite 1323 Stamford, CT 06901.

[44] G. Brajnik, S. Mizzaro, C. Tasso, and L. Rizzi, "Evaluating User Interfaces to Information Retrieval Systems : A Case Study on User Support," *19th Int. Conf. Res. Dev. Inf. Retr.*, no. August, pp. 128–136, 1996.

[45] G. Ingersoll, "Apache Lucene - Scoring," *Copyr. © 2006 Apache Softw. Found. All rights Reserv.*, pp. 1–6, 2006.

[46] "Yahoo News ZA - Latest World News South Africa News Headlines." [Online]. Available: http://za.news.yahoo.com/. [Accessed: 20-Sep-2013].

[47] M. Mathioudakis and N. Koudas, "TwitterMonitor : Trend Detection over the Twitter Stream," pp. 5–7.

[48] P. Kadam, "Trend Detection and Visualization and Custom Search."

[49] M. Spiliopoulou, B. Mobasher, O. Nasraoui, and O. Zaiane, "Guest editorial: special issue on a decade of mining the Web," *Data Min. Knowl. Discov.*, vol. 24, no. 3, pp. 473–477, Mar. 2012.

[50] E. Boldrini, A. Balahur, P. Martínez-Barco, and A. Montoyo, "Using EmotiBlog to annotate and analyse subjectivity in the new textual genres," *Data Min. Knowl. Discov.*, vol. 25, no. 3, pp. 603–634, Mar. 2012.

[51] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, "Community detection in Social Media," *Data Min. Knowl. Discov.*, vol. 24, no. 3, pp. 515–554, Jun. 2011.

[52] A. Tuzhilin, "Customer relationship management and Web mining: the next frontier," *Data Min. Knowl. Discov.*, vol. 24, no. 3, pp. 584–612, Feb. 2012.

[53] J. Hagberg, "Development of a News Reading System," 2012.

[54] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, "Parallel Crawling for Online Social Networks," pp. 1283–1284, 2007.

[55] J. Cho and H. Garcia-Molina, "Parallel crawlers," *Proc. Elev. Int. Conf. World Wide Web - WWW '02*, p. 124, 2002.

[56] S. Catanese, P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Crawling Facebook for Social Network Analysis Purposes," *Methodology*, vol. abs/1105.6, pp. 0–7, 2011.

[57] D. Kim, S. Rho, and E. Hwang, "Detecting Trend and Bursty Keywords Using Characteristics of Twitter Stream Data," *Int. J. Smart Home*, vol. 7, no. 1, pp. 209–220, 2013.

[58] S. K. G. Nancy Burns, *The Practice of Nursing Research: Conduct, Critique and Utilization*, 5th ed. Elsevier/Saunders, 2005, p. 780.

[59] S. Sukamolson, "Fundamentals of quantitative research," *Lang. Inst. Chulalongkorn Univ.*, 2007.

[60] N. Fox and N. M. Hunn, Amanda, "Sampling and Sample Size Calculation Authors," *NIHR RDS East Midlands / NIHR RDS Yorksh. Humber*, 2009.

[61] "WhatsApp :: Home." [Online]. Available: http://www.whatsapp.com/.

[62] A. Cockburn, "Using Both Incremental and Iterative Development," *Technol. Softw. Eng.*, no. May, pp. 27–30, 2008.

[63] S. I. Mfenyana, N. Moroosi, M. Thinyane, and S. M. Scott, "Development of a Facebook Crawler for Opinion Trend Monitoring and Analysis Purposes : Case Study of Government Service Delivery in Dwesa," *Int. J. Comput. Appl. (0975 – 8887)*, vol. 79, no. 17, pp. 32–39, 2013.

[64] M. Carr and J. Verner, "Prototyping and Software Development Approaches," *City Univ. Hong Kong. Retrieved January*, vol. 8, p. 2008, 1997.

[65] X. W. Jun Cai, Vladimir Eske, "Semantic Web & Ontologies," pp. 1–30.

[66] M. Najork and J. L. Wiener, "Breadth-First Search Crawling Yields High-Quality Pages," pp. 114–118, 2001.

[67] "Chapter 1 Web Crawling." [Online]. Available: http://grupoweb.upf.es/WRG/course/slides/crawling.pdf.

[68] M. Najork, "Web crawler architecture," *Microsoft Res. Mt. View, CA, USA*, pp. 3–5, 2009.

[69] "RestFB - A Lightweight Java Facebook Graph API and Old REST API Client." [Online]. Available: http://www.restfb.com/. [Accessed: 20-Sep-2013].

[70] "FacebookClient (RestFB)." [Online]. Available: http://restfb.com/javadoc/com/restfb/FacebookClient.html. [Accessed: 20-Sep-2013].

[71] "JSON." [Online]. Available: http://www.json.org/. [Accessed: 20-Sep-2013].

[72] “The Suggested Upper Merged Ontology (SUMO) - Ontology Portal.” [Online]. Available: http://www.ontologyportal.org/.

[73] D. R. J.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, “Chapter 2 – Types of Performance Testing, Performance Testing Guidance for Web Applications,” 2007. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb924357.aspx. [Accessed: 10-Jan-2014].

Appendix – Questionnaire which was used for Data Collection

In the section is the actual question which was used during the period of collecting the data at Dwesa community. The results which were found after analysis the data that was collected are discuss in Chapter 3 under the Feasibility Study section.

Questionnaire for Data collection in Dwesa

The main objective of the questionnaire is to validate the problems which were noticed during the visits while the research was being carried out. It also seeks to find out how people of Dwesa are currently reporting the issues related to government service deliveries that they are facing. These objectives in mind, we will be able to determine the significance of our research study. This research on its perspective is not intended to take over any current methods which are currently being used, but to optimize the way of doing things with efforts of improving communication between people and their local government official. The questionnaire seeks to find out: Social Networking Sites currently being used, Methods Currently being used to communicate problems with current local government, and the possibility of the improvements the developed system could bring.

Thank you for taking the time to fill in this questionnaire; it should only take 20 minutes. Your answers will be treated with complete confidentiality. If you have any questions about this questionnaire, please free to ask any question.

Section A

1. Which SNSs do you know among the following: (please tick all that apply)
   Facebook ☐

   Twitter ☐

   LinkedIn ☐

Other	

2. Which SNSs do you have accounts among the following: (please tick all that apply)

   Facebook ☐

Twitter ☐

LinkedIn ☐

Other	

3. Which one do you use most: (please tick one)
   Facebook ☐

   Twitter ☐

   LinkedIn ☐

Other	

4. Do you use the SNS select from number 3, on average: (please tick one)
   Less than once a month ☐
   once a month ☐
   once every two weeks ☐
   once a week ☐
   two or three times a week ☐
   Daily ☐

5. What are your main uses of the SNS? (please tick all that apply)
   Update Statuses ☐
   Read other people's statuses ☐
   Upload pictures ☐
CD Send messages ☐
CDI Instant messaging ☐
   Advertising ☐

Other	

If you DO NOT use any SNSs please go to question 11 (Section B)
If you DO use the SNSs, please continue with question 6 below.

6. a. Which application do you use most? (please tick one)
   Update Statuses ☐

Read other people's statuses ☐
Upload pictures ☐
CDII Send messages ☐
CDIII Instant messaging ☐
Advertising ☐

Other	

b. What is your main reason for using SNSs? (Please tick one)

to support course of study ☐
leisure/general enjoyment ☐
independent learning/research ☐
looking for jobs ☐

Keeping in touch with family/friends ☐

Other(please say how)	

7. How do access internet and/or SNSs?
   Mobile Phone ☐
   Personal computer ☐

   Only through Siyakhula Living Labs ☐

Other(please say how)	

8. If you have access to use SNS elsewhere, please say where:
   (please tick all that apply)
   home ☐
   school/college/university ☐
   cybercafé ☐
   work ☐

Other(please say where)	

9. Do you consider yourself in using SNSs:
   a beginner ☐
   an intermediate user ☐
   a fairly experienced user ☐
   a very experienced user ☐
10. Do you see the importance of using SNSs as communication tools?
   Yes ☐

   No ☐

If yes, how important is it SNSs providing communication platform in your life?
Very important ☐
Quite important ☐
Not very important ☐
Not at all important ☐

10. Which of these statements most reflects your view of the SNSs services?
   a vital service ☐
   an add-on service, secondary to other library services ☐
   an unnecessary expense ☐

   Please give a reason for your view:

Section B

11. If you do not use the SNSs, could you please say why: (please tick one)
   didn't know they were there ☐
   no interested on using them ☐
   don't know how to used them ☐
   don't have access to them ☐
   no-one to help/reluctant to ask for help ☐

Other(please say what)	

12. Would you be interested in training on how to use the SNSs?
   Yes ☐
   No ☐

Please now continue with sections C and D

Section C. (All respondents)

13. Are you a resident of Dwesa?

Yes ☐

If yes, please indicate your postcode below:

No ☐

If no, are you visiting from:

Residence	postcode

14. Do you have access to Siyakhula Living Labs?

Yes ☐

No ☐

15. Are you:

male ☐

female ☐

16. Are you:

full-time employed ☐

part-time employed ☐

self-employed ☐

not in paid employment ☐

student ☐

student and working ☐

retired ☐

Other(please say what)	

17. Are you:

under 16 ☐

16-25 ☐

26-35 ☐

36-45 ☐

46-55 ☐

56-65 ☐

over 65 ☐

18. How would you describe your ethnic background?

White ☐

Black African ☐

Other(please say what)	

19. Do you consider yourself to have a disability?

No ☐

Yes ☐

please specify	

Section D. (All respondents)

20. How do you communicate with your local government?

Call meetings ☐

Phone call ☐

E-mail ☐

Other, please specify	

21. Current way of communication, do you find it easy and effective when expressing your views in regard to services delivery issues or any other issues you as community are facing to local government?

Yes ☐

No ☐

If no, why say please specify:

22. Are there any local government offices close to Dwesa?

Yes ☐

No ☐

If yes, do you find it easy to reach them?

No ☐

Yes ☐

If no, where are the closest offices that you usual report your problems?

Please specify:

23. Have you ever feel like there is a need to optimize communication between people of Dwesa and Local government?
    Yes ☐
    No ☐

24. Are there any ways the developed system could improve communication between you and the local government?
    Yes ☐
    No ☐

25. Can you mention few problems Dwesa as a community is facing in terms government services delivery, state below:

26. If there any problems, in your opinion what are the reasons those problem are not attended by government?

Thank you very much for taking the time to complete this questionnaire.
Please hand it back to a member of staff, or put it in the box provided.

If you have any other comments in regard with the research conducted, please add them below:

Thank You, Your Contribution on our research much appreciated

University of Fort Hare
Together in Excellence

Telkom Centre of Excellence
ICTD