ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

COLLEGE OF NATURAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

Design of Local Web Content Observatory System

GASHAW TSEGAYE SHAWO

A Thesis Submitted to the School of Graduate Studies of Addis Ababa University in Partial Fulfillment for the Degree of Master of Science in Computer Science

March, 2015

Addis Ababa University

School of Graduate Studies

College of Natural Sciences

Department of Computer Science

Design of Local Web Content Observatory System

GASHAW TSEGAYE SHAWO

Advisor: SOLOMON ATNAFU (PhD)

MEMBERS OF THE EXAMINING BOARD:

1. Solomon Atnafu (PhD.), Advisor _____

2. _____ _____

3. _____ _____

Dedication

I would like to dedicate this paper to my God and family.

Acknowledgments

First of all, I would like to thank my advisor, Dr. Solomon Atnafu, whose vast knowledge and experience, giving this research proposal title, patience and encouragement has made this work possible. I would also like to thank Addis Ababa University data center staffs which support me providing Internet service.

I would like to thank my almighty God and my family for their love and guidance and who encouraged me to embark on the journey of the Masters that this thesis is the culmination of.

I would also like to extend my gratitude to the Addis Ababa University Computer Science department staffs for their contribution in one way or another in the process of my study. Finally, thanks to all my classmates.

You all have my sincerest gratitude.

Table of contents

Acknowledgments iii
List of Figures vii
List of Tables viii
Lists of Algorithms ix
Abbreviations x
Abstract xi
Chapter One: Introduction 1
1.1 Background 1
1.2 Motivation 3
1.3 Statement of the Problem 3
1.4 Objectives 4
1.5 Methods 5
1.6 Scope and Limitations 6
1.7 Application of Results 6
1.8 Thesis Organization 6
Chapter Two: Literature Review 7
2.1 Information Retrieval on the Web 7
2.2 Web IR Algorithms 9
2.2.1 HITS 9
2.2.2 Page Ranking 10
2.3 Crawling 11
2.3.1 Focused Crawlers 12
2.3.2 Intelligent Crawlers 14
2.3.3 Learning Crawlers 14
2.3.4 Semantic Crawlers 15
2.4 Classification 15
2.4.1 Naïve Bayes 15
2.4.2 Decision Trees 16
2.4.3 Artificial Neural Networks 16
2.4.4 Support Vector Machines 17

2.5 Indexing 17
2.6 Information Retrieval model 21
2.7 Summary 24
Chapter Three: Related Work 26
3.1 Search Engine 26
3.2 Web Content Observatory 27
3.3. Web Document Language Identification 28
3.4. Summary 30
Chapter Four: Design of a Local Web Content Observatory System 31
4.1 Introduction 31
4.2 Architecture of Local Web Content Observatory System 31
4.2.1 The Crawler Component 33
4.2.2 The Statistics Tracker 38
4.2.3 The Content Extractor 40
4.2.4 The Language Identifier 41
4.2.5 Report Generator 44
2.4.6 Web Document Categorizer 44
Chapter Five: Implementation 47
5.1 The Crawler 47
5.1.1 Heritrix 47
5.1.2 Methanol 47
5.1.3 Grub 48
5.1.4 The Choice for Relevant Crawler 48
5.2 Indexers 49
5.2.1 Lemur 49
5.2.2 Lucene 49
5.2.3 Xapian 50
5.2.4 Nutch 50
5.3 The Development Environment 51
5.4 The Development Tools 51
5.5 The Crawler Component 52

5.6 Statistical Tracker 54
5.7 The Content Extractor 56
5.8 The Language Identifier 57
5.9 The Report Generator 59
5.10 Web Document Categorizer 61
Chapter Six: Experimental Result 62
6.1 The Crawler of the Local Web Content Observatory System 62
6.2 The Language Identifier 63
6.3 Web Document Categorizer 67
Chapter Seven: Conclusion and Recommendations 69
7.1 Conclusion 69
7.2 Recommendations 71
References 72
Appendix-1: New crawling job Configuration 78
Appendix-2: Source code for Web document language detection 85

List of Figures

Figure 4-1: General Architecture of the Web Content Observatory System 32
Figure 4-2: Basic Heritrix Architecture 35
Figure 4-3: Process Chain 37
Figure 4-4: The Statistics Tracker 38
Figure 4-5: Architecture of Language Identification 42
Figure 4-6 : Architecture of Web Document Categorizer 46
Figure 5-1: The Crawler Consol 54
Figure 5-2: Sample Crawler Report 55
Figure 5-3: Sample Frontier Report 55
Figure 5-4: Language Identifier User Interface 57
Figure 5-5: A Script to Generate Statistical Report of Identified Language 59
Figure 5-6: Report Generated using Pareto Chart 60
Figure 5-7: Web Content Categorization User Interface 61
Figure 6-1: Identified Web Documents Per Language 66

List of Tables

Table 2-1: IR Model Properties 24
Table 6-1: Status of Domains under the .et ccTLD 63
Table 6-2: Training Corpus Performance Result 65
Table 6-3: Performance Evaluation between LIM and Adopted G2LI 65
Table 6-4: Identified Web Documents Per Language 66
Table 6-5: Evaluation of Categorized English Web documents 67
Table 6-6: Evaluation of Categorized Amharic Web Documents 68

Lists of Algorithms

Algorithm 4-1: Algorithm to identify the status of each domain..... 40
Algorithm 4-2: Algorithm for Archived Document Extractor..... 41
Algorithm 4-3: Algorithm for Language Identifier 43
Algorithm 4-4: Algorithm to Generate Statistical Report of Identified Language..... 44
Algorithm 4-5: Web Document Categorization Algorithm..... 46

Abbreviations

ASP	Active Server Page
ccTLD	Country Code Top Level Domain
CERN	European Organization for Nuclear Research
DNS	Domain Name System
G2LI	Global Information Infrastructure Laboratory's Language Identifier
HITS	Hyperlink-Induced Topic Search
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IR	Information Retrieval
ISP	Information Service Provider
JDK	Java Development Kit
JMX	Java Management Extension
JSP	Java Server Page
JVM	Java Virtual Machine
LIM	Language Identification Module
LSC	Language Specific Crawler
MIMI	Multipurpose Internet Mail Extension
PHP	Hypertext Preprocessor
SDK	Software Development Kit
UNESCO	United Nations Educational, Scientific and Cultural Organization
URL	Uniform Resource Locator

Abstract

The amount of information on the Web as well as the number of Internet users on the Web is growing rapidly. The Web contents are becoming more multilingual and on diverse subjects. Considering a particular group or country, it is very difficult to know how much Web contents are published and which are in what language and on what specific subject. Knowing the status of local Web content of a country or a culture is of critical importance for making an informal decision on policy and strategy design for the development of the multi-lingual and multi-cultural Web.

This research work is therefore aimed to design a local Web content observatory system that measures and reports periodically the qualitative and quantitative content of different domains. The local Web content observatory system mainly consists of four components – the crawler, content extractor, statistical tracker, language identifier, Web document categorizer and report generator.

The crawler downloads documents and then the language identifier detects the language of each crawled Web document and inserts detected language into a database. The statistical tracker monitors the crawler and records statistical data. The Web document categorizer categorizes the collected documents into the selected type of subject. The report generator provides statistical information about the detected language and distribution of Web document per language across the selected sets of domains.

To test and evaluate the system, we have selected all domains hosted under the .et domain. Accordingly about two thousand seed URLs under the .et domain are used and the crawler collected around 263,031 Web documents. According to the accuracy rate measures employed to the language identifier, accuracy rate of 98.67% obtained. To demonstrate the effectiveness of the local Web content categorizer precision, recall and F-measures test were conducted and average precision of 91.7%, recall of 97.2% and F-measures of 94.25% obtained for English document and precision of 91.7%, recall of 87.85% and F-measures of 86.65% obtained for Amharic document. The average accuracy rate of the statistical tracker is 98.72%.

Key words: Information Retrieval, Crawler, Language Identification, Web Document Categorization, Local Web content Observatory System

Chapter One: Introduction

1.1 Background

The World Wide Web ("WWW" or simply the "Web") is a global information medium which users can read and write about via computers connected to the Internet [1]. The history of the Internet dates back significantly further than that of the World Wide Web. The Web is often mistakenly used as a synonym for the Internet itself, but the Web is a service that operates over the Internet, just as e-mail also does.

The content on the Web is usually more important to people when it is typically in their own language and is more relevant to the communities in which they live and work. These communities may be defined by their location, culture, language, religion, ethnicity or area of interest. This relevant content is often referred to as “local content” [1]. The term community is used in a broad way to include not only local professional communities (public and private), but also non-professional content creators and users. This idea of relevant content in the speaker’s own language is called local content. The subset of information that is relevant to an individual is often closely related to knowledge within any of the communities where she or he resides. The term “Local Content” is described or defined differently in different sources. However, a UNESCO definition states “local content” as an expression and communication of a community’s locally generated, owned and adapted knowledge and experience that is relevant to the community’s situation [1].

There are two observable trends with respect to local content across the various measures. First, local content is growing very fast in volume across the world, often at astonishingly high rates. Second, its composition is changing as developing countries and non-English speaking areas are much better represented in terms of content production than before. Because local content cannot be measured directly, a set of measures must be used to indirectly infer its size. To facilitate classification of local content, the work in [2] first divided potential measures into (i) measures which are associated with a particular country and (ii) measures tied to a particular language. The aim of the Design of Local Web Content Observatory System is thus to provide a central authority to *observe and understand* the dynamics of local Web content, structure and usage. The

local Web content observatory helps to check the status, the language and the content of each domain.

With the enormous growth of the Web, search engines play a critical role in retrieving information from the borderless Web. Although many search engines can search for content in numerous major languages, they are not capable or not efficient in searching pages of less-resourced languages (such as languages in Ethiopia). Since the Web is a distributed, dynamic and rapidly growing information resource, a normal Web crawler cannot download and identify all pages in different language. For a language specific search engine, Language Specific Crawler (LSC) is needed to collect targeted pages. In the Ethiopian Context there are efforts made by different researchers to design language specific crawlers in Amharic language [3, 4], Oromiffa language [5] and Tigrigna language [6].

The Web is a vast collection of completely uncontrolled heterogeneous documents. Today though Web search engines provide the easiest way to reach information resources that are available on the Web, the amount of information available via networks and databases has increased and is still rapidly increasing. Existing search and retrieval engines provide limited assistance to users in locating the relevant information they need. However language specific programs are used by search engines to gather information on Web pages. Such programs, called crawlers, also follow links from one Web page to another and work with indexing code to store data for later searching [7]. Thus, language specific crawlers are very important to identify local content.

There have been many efforts by citizens, organizations and some researchers to promote Ethiopic local Web content on the Internet but there is no work that estimates the size and extent of local content in the context of the country. Internet access has a significant impact to a country's economic development. The work in [2] shows identifying and analyzing the status of Internet content in the Ethiopian context and how much Internet access impacts the economy with some statistical figures. Currently, ethio telecom [8] provides domain name services to about 2000 organizations under the .et domain.

The goal of this research is thus to design and implement Local Web Content Observatory system. It deals with the analysis of local Web content in Amharic, Oromiffa, Tigrigna and

English languages. The main significance of this research is to help to observe and understand the status and dynamics of Web content and provide qualitative and quantitative data that might help its services and development. For this study Web documents ware collected from the .et domain using language focused crawlers then different techniques are applied to identify the different content and the language of the Web documents.

1.2 Motivation

The work in [2] presents the status of the local Web Internet content and the environment for local Web content development in the Ethiopian context. It has selected a metric of measuring local content availability and used the same to present what is available as local content for Ethiopia as much as possible. However, absence of certain tools to quantify the available local Web content had been a challenge. Currently ethio telecom provides domain name services for 2000 organizations under the .et domain but there is no tool that provides qualitative and quantitative information. After organizations or individuals host their domain there is no way for the hosting organization to check the status of each domain. It is, therefore, the need for conducting this work to come up with a system that provides qualitative and quantitative data about local Web content under the .et domain. This will help to know the availability of local content, and to investigate areas to be promoted online and to identify the gap in the local Web content representation of local language based content.

1.3 Statement of the Problem

The amount of information about the local Web content as well as the number of new users on the Web is growing rapidly. Absent of tools to quantify qualitative and quantitative data found under certain domain is one of the challenges for policy makers to promote local Web content on the Internet. The language identifier [54] didn't correctly identify the language of multilingual Web documents.

The Language Observatory project [9, 10] planned primarily to provide means for assessing the usage level of each language in cyberspace. Using UbiCrawler [11], a scalable fully distributed Web crawler used in the project, the project considers only 100 of domains hosted under the .et domain and the language identifier used has the following problems:

- It doesn't correctly detect the language of multilingual Web pages.
- At a time it identifies only one file which is difficult to identify millions of Web pages.
- Not well trained.
- It returns a Japanese language for a Web page that have images and text content.

The crawler used by the work in [10] used broad scope that is not limited in to the host, this result to crawl Web pages outside the host scope. In addition to this so far research work regarded the local Web content didn't address the language of multilingual Web documents, the status of each domain and the number of document presented per language under the .et domain.

1.4 Objectives

General objective

The general objective of this research is to design a Local Web Content Observatory system.

Specific objectives

Specifically, the objectives of this research are to:

- Review related works on local content crawling.
- Review existing Amharic, Oromiffa, Tigrigna and English crawler algorithms.
- Identify the requirements of creating a local Web content observatory system.
- Design a local Web content observatory system that fits the requirements identified.
- Identify the language of multilingual Web content.
- Develop a system that dynamically checks the local Web contents and status.
- Identify the criteria of categorizing local Web content.
- Categorize the crawled documents into selected fields of topic by language.
- Generate a statistical data on the status and the dynamics of available local Web content.
- Testing and evaluate the system.

1.5 Methods

In order to accomplish the objectives of the research, the following methods and procedures will be used.

i. Literature reviews will be done on related fields to this research work. Information retrieval on the Web will be studied with respect to its feature that affects the information retrieval and Web IR algorithms. Information retrieval techniques, local search engine, indexing and crawler will be reviewed in detail. In order to learn lessons on how to categorize Web document different techniques of classification techniques will be reviewed.

   Web language identification methods will be reviewed to find appropriate tools and techniques that can be applied in our work.

ii. Different open source tools and programming languages will be selected to implement the system. Java programming language will be used primarily to implement the system because of the following reasons:-

   - It has good networking capability
   - It is platform independent
   - Many reusable components are available on the Web

   The crawler downloads Web documents hosted under the .et domain, which is limited to crawling within each host scope. After crawling Web documents different pre processing operations will be done to enable for report generation, language identification and categorization.

iii. Experiments will be conducted to test the local Web content observatory system. The performance of the crawler is evaluated in terms of the documents crawled and ability to recovery after crash. Other components of the system will be evaluated in terms of Accuracy rate, Precision, Recall and F-measure.

1.6 Scope and Limitations

This work deals with designing a general architecture for Web content observatory and developing a working system based on the design. The crawled local Web contents are also categorized into selected topics by language. This research work considers only Amharic, Oromiffa, Tigrigna and English language for the seed URLs hosted under the .et domain.

1.7 Application of Results

The possible applications at the end of this research are listed below.

- It helps to identify the language of multilingual Web contents.
- Improves the performance of local searching and retrieval of local Web document.
- To create policies and strategies for local Web content development.
- Provides a central authority to observe and understand the dynamics of Web content.
- It helps us to know the status of the local content and categorize Web documents based on subject.
- Provide information about in what language local contents are presented.

1.8 Thesis Organization

The remaining part of this thesis is organized as follows. Chapter Two presents the review of literature in the domain. Chapter Three presents related works done on Web content observatory system and related areas. The fourth Chapter deals with the design of local Web content observatory system. Chapter Five presents the implementation of the local Web content observatory system under the .et domain. Chapter Six presents experimental results. Finally, conclusions and recommendations are given in Chapter Seven.

Chapter Two: Literature Review

Examining the techniques and technologies for the generation and observation of content under certain domain cannot be addressed without first exploring IR on the World Wide Web. The purpose of this Chapter is thus to explore the techniques for discovering, classifying and harvesting content of the Web for the topic and language specific repositories. The Chapter will first describe about information retrieval on the Web and some of the prominent Web IR algorithms. That will be followed by Section 2.2 which discuses how information retrieval is made from the Web. Section 2.3 describes Web content crawling, and in particular focused crawling, which allows the discovery of topic and language specific contents. The generation of specific content is a means of determining the relevancy of a downloaded document, document classification will be discussed in Section 2.4. The content of downloaded documents needs to be represented in a way to allow, in conjunction with a user query, identifying the location and ranking of relevant content. The process of generating a document representative to facilitate search and retrieval (also known as indexing) and models for retrieval will be discussed in Section 2.5. Finally Section 2.6 will discuss about information retrieval mode.

2.1 Information Retrieval on the Web

Web information retrieval is a technology for helping users to accurately, quickly, and easily find information on the Web. The World Wide Web is a very large distributed digital information space. From its origins in 1991 as an organization-wide collaborative environment at CERN [66] for sharing research documents in nuclear physics, the Web has grown to encompass diverse information resources: personal home pages; online digital libraries; virtual museums; product and service catalogs; government information for public dissemination; research publications; and mail servers. The ability to search and retrieve information from the Web efficiently and effectively is an enabling technology for realizing its full potential. With powerful workstations, high network bandwidth and parallel processing technology, efficiency is no more a major bottleneck. In fact, some existing search tools sift through gigabyte-size precompiled Web indexes in a fraction of a second. But retrieval effectiveness is a different matter. Current search tools retrieve too many documents, of which only a small fraction are relevant to the user query. Furthermore, the most relevant documents do not necessarily appear at the top of the query

output order. Writing about the Web is a challenging task for several reasons [12]. First, its dynamic nature guarantees that at least some portions of any document on the subject will be out-of date before it reaches the intended audience, particularly URLs that are referenced. Second, a comprehensive coverage of all of the important topics is impossible, because so many new ideas are constantly being proposed and are either quickly accepted into the Internet mainstream or rejected. Finally, as with any review paper, there is a strong bias in presenting topics closely related to the authors' background, and giving only quick treatment to those of which they are relatively ignorant.

One way to find relevant documents on the Web is to launch a Web robot (also called a wanderer, worm, walker, spider, or knowbot). These software programs receive a user query, and then systematically explore the Web to locate documents, evaluate their relevance, and return a rank-ordered list of documents to the user. The vastness and exponential growth of the Web make this approach impractical for every user query. An alternative is to search a precompiled index built and updated periodically by Web robots. The index is a searchable archive that gives reference pointers to Web documents. This is obviously more practical, and many existing search tools are based on this approach. Generating a comprehensive index requires systematic traversal of the Web to locate all documents. The Web's structure is similar to that of a directed graph, so it can be traversed using graph-traversal algorithms. Because Web servers and clients use the client-server paradigm to communicate, it is possible for a robot executing on a single computer to traverse the entire Web. There are currently three traversal methods [13]:

- Providing the robot "seed URLs" to initiate exploration. The robot indexes the seed document, extracts URLs pointing to other documents, and then examines each of these URLs recursively in a breadth-first or depth-first fashion.
- Starting with a set of URLs determined on the basis of a Web site's popularity and searching recursively. Intuitively, we can expect a popular site's home page to contain URLs that point to the most frequently sought information on the local and other Web servers.
- Partitioning the Web space based on Internet names or country codes and assigning one or more robots to explore the space exhaustively. This method is more widely used than the first two.

2.2 Web IR Algorithms

The effectiveness of IR is often measured using two terms: precision and recall. Precision is the number of relevant documents retrieved divided by the total number of documents retrieved. Recall is the number of relevant documents retrieved divided by the total number of relevant documents (whether they were retrieved or not). Both of these measures can be said to be subjective, as determining if a document is relevant requires knowledge of the user's intent when they made the query in the first place. As the Web is constantly growing with new content being generated all the time, this cannot be known. Web-based IR algorithms typically employ some sort of link analysis algorithm to effectively rank Web pages. Garfield's impact factor [14], which defines a measure of how impactful an academic or scientific journal can be determined by computing the average amount of citations papers published in the journal have received in the past two years. This concept is a key foundation of the Web page ranking algorithms in use today. Pinski and Narin [15] suggested that a citation from a more influential paper should be weighted more; this concept can also be extended to Web page ranking algorithms.

2.2.1 HITS

Hyperlink-Induced Topic Search (HITS) is a link analysis algorithm for rating Web pages, and uses the Hubs and Authorities [16] theory. Hubs are Web pages that serve as directories and link directly to authoritative pages. A good Hub will link to many Authorities, and a good Authority will be linked to by many different hubs. The HITS algorithm operates as follows. A focused sub graph of the Web is generated from the results of a text based search engine for a given query. The sub graph is then enlarged by adding the links on each page (up to certain threshold) of the sub graph. The sub graph is then pruned by removal of intrinsic links, which are links within the same domain name (as they are often purely navigational), thus leaving the transverse links, that is the links across different domain names. Next the Hub and Authority values are computed for each page in the sub graph.

To begin each page is given a Hub value and an Authority value of 1. Then the following 4 steps are followed:

1. Each page's Authority value is refreshed to be equal to the sum of the Hub values of all the pages linking to it.
2. Each page's Hub value is refreshed to be equal to the sum of the Authority values of each page it links to.
3. The Hub and Authority values are normalized by dividing each Hub value by the sum of the squares of all the Hub values and dividing each Authority value by the sum of the squares of all the Authority values.
4. The 3 previous steps are repeated as necessary.

2.2.2 Page Ranking

Unlike HITS, the Page Rank [17] algorithm is run at index time, and extends the Hubs and Authorities concept. A link from page A to page B is counted as a vote for page B by page A. The importance of the page casting the vote is also taken into account; a vote by a more important page is given more weight. Page Rank forms the basis of the Google search engine, and as time has gone on various other factors have been incorporated into the algorithm, but Google does not reveal specific details in order to preserve their market lead and also to prevent manipulation of the search results. The Page Rank algorithm is a probability distribution which represents how likely it is that a person surfing the Web randomly traversing links will arrive at any particular page. The Page Rank for a given page is a probability value between 0 and 1 indicating how likely it is to arrive at the page via a random link. Any page which links to a given page contributes their Page Rank value divided by the normalized number of links on each linking page. The contributions from all linking pages are summed to determine a Page Rank for a given page. The general form can be specified as:

$$PR(U) = \sum_{v \in B_u} \frac{PR(V)}{L(V)}$$

The Page Rank for page U is related to the Page Ranks of each page V in the set Bu divided by the number of L(V) from links from page V. Bu is the set of all pages that link to page U. The simplified form of the algorithm above was enhanced with the addition of a damping factor. This models the fact that the random surfer will eventually get bored and stop traversing links. An acceptable value for the damping factor is a probability of 0.85.

The revised formula can be seen below:

$$PR(pi) = \frac{1-d}{N} + d \sum_{pj \in M(pi)} \frac{PR(pj)}{L(pj)}$$

where pi…… pN are the pages being considered. The set M(pi) contains all pages linking to pi. L(pj) is the number of out-links (outward hyperlinks) on page pj and N is the total number of pages.

From the formula above one can see that the damping factor d has the effect of decreasing a given page's Page Rank. It should be noted that pages with no out-links are assumed to link out to all other pages in the collection, their Page Rank values shared equally among all other pages.

2.3 Crawling

A crawler also popularly known as spider or robot is a program which visits Web servers spread across the world irrespective of their geographical location, downloads and stores Web document on a local machine mostly on behalf of the Web search engine. [18, 19, 20] The functionality of a Web crawler is given below:

- The crawler starts crawling with a set of URLs fed into it, known as seed URLs.
- The crawler downloads the page.
- It extracts the URLs from the downloaded page and inserts them in a queue. From the queue the crawler again retrieves the URLs for downloading the next page.
- The downloaded page is saved in the repository.
- The process continues until the crawler stops.

Web search engines such as Google attempt to do a breadth-first crawl, i.e., traverse the entire Web, download each page and index it for use in later queries. However there are parts of the deep Web that are not accessible, and cannot be indexed by search engines. In 2004 it was estimated that the deep Web is on the scale of 307,000 sites, 450,000 databases, and 1,258,000 interfaces [21], but currently there is no exact figure that indicate the number of deep Webs

found. About a third of this was indexed by the major search engines. Research work in [22] reveals the facet of the deep Web[1], comprising in the vicinity of trillions pages of information. A crawler may be configured to download a fixed number of pages, or it can continue crawling until storage or CPU resources have been consumed. Determining which page to retrieve next in the crawl frontier has been the focus of a huge amount of research in this area. Search engines typically employ a breadth-first crawl as they can potentially be invoked with any query. The aim of focused crawling is to crawl using a best-first ordering of the pages in the crawl frontier. The following subsections will discuss focused, intelligent, learning and semantic crawlers.

2.3.1 Focused Crawlers

Focused crawlers are programs which are used to selectively retrieve Web pages based on predefined criteria or topics. Because focused crawlers target a narrower slice of the Web, and can crawl this slice to a greater depth, they can potentially retrieve information that was missed by a traditional breadth-first crawler. Focused crawlers make use of the phenomenon of Topical Locality [23], i.e., the content of a linked page is likely to be related to the content of the page on which it was linked. Links that appear closer together on a Web page tend to be more related to each other and the anchor text can provide useful information about the relevancy of the linked content. Before a focused crawl is conducted a set of locations from which to begin the crawl are required, these are known as seed pages. The relevancy of the returned result is highly dependent on the selection of good seed pages. Seed pages should either contain highly relevant content, or else link to highly relevant content in a minimum amount of link traversals.

Methods of Focused Crawling

Link prioritization is used by both Fish Search [24] and Shark Search [25] approaches. Fish Search employs keyword matching and assigns a binary priority value to potential downloaded pages; 1 if the page is relevant to the search, 0 if not. The Shark Search approach expands on this by employing a Vector Space Model to assign priority values more precisely than the binary

[1] The deep Web is content which is invisible to search engines. There are various reasons this can be the case: pages are dynamically generated, pages contain no in-links, page access requires registration and login, page content varies with context, page access requires links generated by a scripting language or pages contain textual content in specific file formats not handled by search engines.

values assigned by Fish Search. When assigning a priority value, Shark Search considers the content of the page, the priority value of the parent page, the anchor text, and the text in close proximity to the links.

In the Web Topic Management System (WTMS) [26] a user supplies seed pages or keywords from which seed pages can be generated. The crawler creates a Representative Document Vector (RDV) derived from keywords which occur frequently in the downloaded seed pages. The links on the seed pages are downloaded and a Vector Space Model is used to determine their similarity to the RDV. Pages with a similarity in excess of a certain set value are indexed and their links added to the crawl frontier. These out-links are then retrieved until the crawl frontier is exhausted. The in-links, that is the pages linking to a particular page, can be obtained from the Google search engine. The crawler downloads the in-links of the seed pages and indexes them if they are sufficiently similar to the RDV. Then their in-links are added to the crawl frontier and so on. In order to reduce the number of irrelevant downloaded pages the crawler uses 2 heuristics. The first specifies that a linked page will only be downloaded if it is sufficiently near in the directory hierarchy. Pages in parent or child directories will be downloaded, as well as sections of sibling directories. The second heuristic specifies that if more than 25 pages have been downloaded from a directory and 90% discarded (because of insufficient similarity to the RDV) then no further pages will be downloaded from that directory.

Taxonomic Crawling was suggested by Chakrabarti1 *et al.* [27] as an improvement to focused crawling. This involves the addition of the classifier and the distiller programs to provide guidance to the crawler. The classifier's responsibility is evaluating the relevance of a retrieved document, while the distiller seeks to identify hypertext nodes which would lead to many relevant pages within a minimum of link traversals. The system is set up with an initial coarse grained canonical classification tree, such as the Open Directory Project (ODP) [28]. The user imports some example URLs, and the system proposes classes that fit the examples best. The user can accept these proposals or if the taxonomy is too coarse, refine categories and reassign examples to the new categories. The system will also propose some additional URLs that appear similar to the given examples. The classifier then integrates the selections of the user into the statistical class models. A crawl can now be initiated. During the crawl the distiller identifies hubs and the (re)visit priorities of these pages and their immediate neighbors are increased. From

time to time the user can give feedback on resource lists which is fed back to the classifier and distiller.

Genetic Algorithms can be applied to IR, whereby the process of natural selection to be applied to solutions, effective solutions thrive, and ineffective solutions go extinct. Info Spiders [29] is an example of this, a collection of crawler agents traverse the Web searching for relevant content, starting at a randomly chosen seed page. They are given a random behavior and a supply of energy. Agents which are successful have their energy replenished and survive; those that do not return relevant content die off. Agents can change their behavior by determining the best strategy for having their energy replenished, and agents can produce offspring's where their behaviors are combined if two agents reach a page at the same time.

Context Graph: One method to improve the priority value given to a page is by using a context model. This models the context within which the relevant content is usually found on the Web. The model or context graph [68] is generated using an existing search engine to find pages which link to a set of seed pages. The found pages are assigned a relational value based on the least amount of link traversals it takes to reach one of the seed pages. Then during a crawl the link distance between an unknown page and a relevant page can be estimated. Thus the crawler can determine if it is worthwhile to crawl deeper, given the page's context. Links which are closer to the relevant pages are prioritized.

2.3.2 Intelligent Crawlers

An intelligent crawler does not begin with seed pages, instead predicates are defined. The predicates can be in-linking Web page content, candidate URLs structure, or other behaviors of the in-linking Web pages or siblings [67]. They are then used to estimate the probability a given page will satisfy a query on a particular topic. The Web's link structure is learned and so the classifier is trained as the crawl progresses. The crawl begins at general points on the Web, and then focuses on relevant content as it comes across pages which match the predicates.

2.3.3 Learning Crawlers

Instead of seed pages, a learning crawler is started with a training set. A training set will consist of example pages indicating which pages are deemed relevant to the topic of the crawl. It may

optionally include the links traversed leading to the relevant pages and also may include pages that are not relevant to crawl. Once the crawl has been initiated, a classifier analyses retrieved content and determines content relevancy, and assigns a priority. Naïve Bayes classifiers, decision trees, neural nets and Support Vector Machines (SVMs) have been used as classifiers for Learning crawlers [69] (see Section 2.4). The Context Graph method is extended using the Hidden Markov Model crawler. A user surfs the Web searching for pages stating if a visited Web page is relevant or not. The sequence of page visits is recorded and can then be used to train the crawler, helping it to recognize paths which lead to relevant content.

2.3.4 Semantic Crawlers

As far as best-first crawlers are concerned, content relevance equates to content similarity. Lexical term matching is used to determine if two documents are similar. This does not allow for terms that are semantically similar but lexically different. As a result of this traditional crawlers ignore content that is lexically dissimilar but semantically comparable to relevant content. Semantic crawlers [70] attempt to address this issue using term ontologies. These term ontologies allow similar terms to be related to each other using the various types of semantic links —Is-A, Is-Part-Of, etc.

2.4 Classification

Classification is used to assign a document to a particular category (also referred to as a class) or categories, based on its contents. Classifiers will typically require training in order to generate a classification model, which can then be used to categorize a document. The following subsections will discuss the Naïve Bayes, Decision Tree, Artificial Neural Network (ANN) and Support Vector Machine (SVM) classifiers, their training requirements and finally analyze their strengths and weaknesses.

2.4.1 Naïve Bayes

A Naïve Bayes classifier is a probabilistic classifier which applies Bayes theorem [30] with naïve independence assumptions and has been in use in IR for over 40 years. When determining if a document should belong to a certain class, each property necessary for conferring membership of

a class is treated as being independent of all the other properties, even if one property is related or depends on another. Each property independently contributes to the probability that the document is a member of the class. Because of the ease of implementation and speed of use, Naïve Bayes is often used as a baseline for classification. It requires minimal training data for estimating the parameters required for classification. However, it currently has a reputation for poor performance, but with some modifications it has been shown [31] to achieve better performance, close to SVM techniques.

2.4.2 Decision Trees

A classifier will use a decision tree (also known as a classification tree) to generate a model that predicts the class of an inputted document. The classification decision tree is constructed as follows; the root node of the tree contains all documents. An internal node contains a subset of the documents of its parent node, derived according to one attribute. The arc between a parent and a child node is labeled with a predicate to apply to the attribute at the parent node. A leaf node is labeled with a class. The tree is recursively partitioned from the root node. Documents are split into subsets according to an attribute, the attribute with the highest Information Gain [32] or using the Gini Index of Diversity [33] is chosen first. Trees can grow to be excessively large and so must be pruned to compensate for over fitting.

2.4.3 Artificial Neural Networks

ANNs are electronic representations of the neural structure of the brain. The network is trained by processing records sequentially and learning by comparing its classification with the known classification. Errors from the first classification are fed back [34] into the system and used to modify the algorithm for classification of the second record and so on. A neuron takes a set of inputs, each of which has an associated connection weight (initially a random value), it sums the weights and maps the result to an output. The network is composed of layers which are themselves composed of connected neurons. There is an input layer connected to one or more hidden layers which are connected to an output layer. The input layer simply takes the input values of a record and passes them to the connected hidden layer. The output layer has a separate output for each individual class. A record processed by an ANN results in a value at each output in the output layer, the record is then categorized as belonging to the class with the highest value.

Because the classes are known for the records in the training phase, the outputs can be assigned 1 for the correct class and 0 for all the others. The calculated values are compared to the correct values to generate an error value which can be fed back into the hidden layer so the input connection weights in the neurons can be adjusted to produce better results the next time a record is processed. The same training data can be processed several times to further refine the weights. It is possible that a neural network does not learn, this is the case if the correct output cannot be derived from the input data or there is not enough training data to sufficiently refine the network.

2.4.4 Support Vector Machines

Proposed in 1992 by Boser *et al.* [35], SVMs are used for separating data into one of two categories. An N-dimensional hyper plane model is built from a training set. The training set consists of examples, each one labeled as belonging to one of two categories. The examples are mapped as points in a hyper plane, such that examples in separate categories are separated by the widest possible margin in the space. Once trained a new example will be mapped to the space and estimated to belong to the category on whichever side of the margin they have been mapped to. The ability of an SVM to learn is independent of the dimensionality of the hyper plane. In the context of text classification a document will be reduced to feature vectors. Each feature vector will typically be a word stem (as a result of a prior stemming process), after all the stop words have been removed from the text. Document text that produces a high dimensional input space, few irrelevant features and document vectors are sparse and these properties are well suited to use with SVMs.

2.5 Indexing

With the continuous production of content on the Web, the importance of appropriate indexing of this content has never been greater. In the context of IR, indexing treats a document as an unordered set of words. Indexing attempts to do some term analysis of the document in order to decide on its subject matter. We can view effective Web searches as an information retrieval problem [36, 37]. IR problems are characterized by a collection of documents and a set of users who perform queries on the collection to find a particular subset of it. This differs from database problems, for example, where the search and retrieval terms are precisely structured. In the IR

context, indexing is the process of developing a document representation by assigning content descriptors or terms to the document. These terms are used in assessing the relevance of a document to a user query. They contribute directly to the retrieval effectiveness of an IR system. IR systems include two types of terms: objective and nonobjective. Objective terms are extrinsic to semantic content, and there is generally no disagreement about how to assign them. Examples include author name, document URLs, and date of publication. Nonobjective terms, on the other hand, are intended to reflect the information manifested in the document, and there is no agreement about the choice or degree of applicability of these terms. Thus, they are also known as content terms. Indexing in general is concerned with assigning nonobjective terms to documents. The assignment may optionally include a weight indicating the extent to which the term represents or reflects the information content. The effectiveness of an indexing system is controlled by two main parameters. Indexing exhaustively reflects the degree to which all the subject matter manifested in a document is actually recognized by the indexing system. When the indexing system is exhaustive, it generates a large number of terms to reflect all aspects of the subject matter present in the document; when it is no exhaustive, it generates fewer terms, corresponding to the major subjects in the document. Term specificity refers to the breadth of the terms used for indexing. Broad terms retrieve many useful documents along with a significant number of irrelevant ones whereas narrow terms retrieve fewer documents and may miss some relevant items.

The effect of indexing exhaustively and term specificity on retrieval effectiveness can be explained by two parameters used for many years in IR problems: recall and precision.

Indexing terms that are specific yields higher precision at the expense of recall. Indexing terms that are broad yields higher recall at the cost of precision [71]. For this reason, an IR system's effectiveness is measured by the precision parameter at various recall levels. Indexing can be performed either manually or automatically. The total size of the Web together with the diversity of subject matter make manual indexing impractical. Automatic indexing does not require the tightly controlled vocabularies that manual indexers use, and it offers the potential to represent many more aspects of a document.

It is necessary to reduce the input text to a document representative. A document representative will consist of a list of class names which can also be referred to as index terms. Each class name represents a class of words. So assigning a document representative to a particular index term implies that one of the original documents significant words is contained within a particular class of words. Achieving this will require three steps:

a. Removal of stop words
b. Stemming
c. Detection of equivalent stems

a. Removal of stop words

Zipf's Law [38] states that "the product of the frequency of use of words and the rank order is approximately constant". This essentially means that words which occur very commonly or very rarely in a document have little descriptive value. Common words will occur in too many documents to add any description of value and extremely rare words may possibly be misspellings or uncommon proper nouns. Words which occur with medium frequency are best used for differentiating between documents in a collection. The removal of high frequency or stop words is usually a process of comparison of the original document with a stop list of words to be removed.

b. Stemming

Suffix stripping or stemming is based on the assumption that words with the same stem are referring to the same concept and the index should reflect this. It is usually done using a suffix list and context rules. The context rules are devised in order to avoid mistakenly removing suffixes. But even with the best rules it is inevitable that errors will occur and terms will be conflated when they should not be. However the error rate has been shown to be on the order of less than 5% [39].

c. Index Term Weighting

Keen and Digger [40] have defined the terms indexing exhaustively and indexing specificity. The first is defined as the quantity of different topics indexed and the latter as the precision with

which a document is actually indexed. Quantifying these definitions proves to be very difficult. High levels of indexing exhaustively lead to high recall and low precision. Whereas low levels lead to low recall and high precision. On the other hand high levels of indexing specificity lead to high precision and low recall and low levels lead to low precision and high recall. Index term weighting schemes have been applied such that each index term is assigned a weight proportional to the frequency of its occurrence in the document. The term frequency (tf) can be defined as follows:

$$\mathrm{tfi;\ j} = \frac{\mathrm{ni;\ j}}{\sum_{\mathrm{k}} \mathrm{n_{k}, j}}$$

where ni; j equals the number of occurrences of a term ti in document dj divided by the sum of the number of occurrences of all terms n_k in document dj, k is number of documents.

Index term weighting schemes have also been applied such that each index term is assigned a weight proportional to the frequency of its occurrence in the entire collection of documents. The inverse document frequency (idf) can be defined as follows:

$$\mathrm{idfi} = \log \frac{|\mathrm{D}|}{|\{\mathrm{d} : \mathrm{ti} \in \mathrm{d}\}|}$$

where | D | is the total number of documents and$|\{\mathrm{d} : \mathrm{ti} \in \mathrm{d}\}|$ is the number of documents in which ti occurs.

Salton and Yang [41] combined the tf and idf thus taking into account both the inter and intra document frequencies, defined as term frequency–inverse document frequency (tf-idf) weighting which is equal to:

$$\text{tf -id fi; j} = \text{t fi; j \_id fi}$$

Using this weighting they were able to draw the following conclusions:

- High frequency index terms have little value in retrieval, whatever the distribution.

- Medium frequency index terms proved most valuable, especially with a skewed distribution.
- Infrequent terms with a skewed distribution have value, but were less valuable than medium frequency terms.
- Very infrequent terms have little value, being only more valuable than high frequency terms.

2.6 Information Retrieval model

An IR model describes a means of communicating an information need and how it may be refined. It specifies the format of a user query and also how that query will be satisfied, that is, how a document will be retrieved that fulfils the user's specific topic of interest. The Boolean, Vector Space, Probabilistic and Language models will be described and analyzed.

An IR model is characterized by four parameters:

1. representations for documents and queries
2. matching strategies for assessing the relevance of documents to a user query,
3. methods for ranking query output, and
4. mechanisms for acquiring user-relevance feedback

a. Boolean Model

The Boolean model is historically the oldest and most common IR model. It is based on boolean logic and classical sets theory. Documents in a collection are treated as a set of terms which have been extracted from all documents in the collection. Each term is assigned a binary weight, 1 denoting a term's presence in the document and 0 denoting its absence. Queries can then be represented as a Boolean expression composed of terms and boolean connectors AND, OR and NOT. In a search using the Boolean model the documents retrieved are those which are TRUE for a given query [42]. Boolean expressions can be quite limited and so the model was extended with the addition of term proximity operators. A proximity operator allows the ability to specify that two query terms must be present close to each other in a document. The closeness of terms

may be defined in terms of a structural unit (sentence or paragraph for example) or a number of intervening words.

b. Vector Space Model

In the Vector Space Model both document and queries are represented by vectors [72]. A term (typically a word or a phrase) that is present in the document or query is given a non-zero value in the document or query vector. The value corresponds to the term weight discussed in the previous section. For example a document could be represented by the vector:

$$\vec{d}_k = (w1,k, w2,k, \ldots\ldots\ldots wt,k)$$

were term wi($1 \leq i \leq k$) is non negative value denoting the single or multiple occurrence of term i in document d.

The query could be represented by the vector:

$$\underset{q}{\rightarrow} = (w1,q; w2,q, \ldots\ldots\ldots wt,q)$$

were term wi($1 \leq i \leq q$) is non negative value denoting the number of occurrences of wi in the query .

Each term is represented by a dimension in the vector. The angle between the two vectors is used to compare the user query and the document. The cosine of the angle can be used as it has the property such that identical vectors have a cosine of 1 and orthogonal vectors have a cosine of 0. If tf-idf weighting (as described in the previous section) is used, the document dk can be compared with query q using the cosine similarity function as shown below:

$$sim(d_{k,q}) = \cos(\overrightarrow{dk}, \vec{q}) = \frac{\overrightarrow{dk}}{|\overrightarrow{dk}|}\frac{\vec{q}}{|\vec{q}|} = \frac{\sum_{i=1}^{t} wi,k*}{\sqrt{\sum_{i=1}^{t} w^2,k*}}\frac{wi,q}{\sqrt{\sum_{i=1}^{t} w^2 i,q}}$$

c. Probabilistic Models

This is a model operating on the basis of the probabilities ranking principle. That is, the documents returned by a query should be ranked by decreasing probability of relevance to a

query. The model estimates the probability of relevance and there have been several methods developed for this process. The models require initial estimated assumptions of relevancy because there are no previously retrieved documents upon which to base the probabilities. The initial estimations are then recursively refined to obtain the final ranking of pages. The Bayes rule [43] can be applied so that odds rather than the probability that the retrieved pages are relevant can be used. The odd that a page is relevant is equal to the probability it is relevant divided by the probability than it is not relevant. The models make term independence assumptions as well as taking into account term presence and term absence [44].

d. Language Model

The query a user forms when searching for information will typically consist of keywords they expect to appear in a retrieved document. A language model attempts to model this. A probabilistic model is built for every document in the collection. The probability of a language model for a given document D generating an inputted query Q was defined by Ponte and Croft [45] as:

$$p(Q|D) = \prod_{q \in Q} P(q|D) = \prod_{q \in Q} \frac{D_q}{|D|}$$

This probability value is calculated for each candidate document which allows ranking of the results. A practical problem with the function above is that documents missing one or more of the query terms will be assigned a probability of zero. To combat this some smoothing to avoid zero frequencies must be done. Smoothing can be achieved either using discounting or interpolating methods. Discounting methods involve adding or subtracting a constant, to redistribute the probability mass. A summary of some of the comparable properties of the models present in the Table 2-1.

Table 2-1: IR Model Properties

	Boolean	**Vector space**	**Probabilistic**	**Language**
Mathematical basis	Set Theory	Algebraic	Probabilistic	Probabilistic
Ranking of Result	No	Yes	Yes	Yes
Binary term weight	Yes	No	Yes	No
Term frequency accounted for	No	Yes	No	Yes

2.7 Summary

Search engine indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. The effectiveness of IR is often measure using two terms precision and recall. Crawlers download and stores Web documents on a local machine mostly on behalf of the Web search engine. Naive Bayes classifiers are well suited for numeric and textual data, and are easy to implement when compared with the other classifiers as well as having low overhead. However the independence assumption does not fit with reality and it performs very poorly with correlated data. The advantages of using a Decision Tree classifier are that is is easy to generate rules and reduce the problem complexity. However the time spent training can be expensive, and once a mistake is made at a node, any nodes in the sub-tree are incorrect. It is possible the excessively large trees will be generated so a means of pruning must be devised. ANNs are capable of producing good results in complex domains. They are suitable in the discrete and continuous domain. Classification is fast but training quite slow. Learned results can be difficult to interpret compared with decision trees and like decision trees they may suffer from over fitting. SVMs are superior to ANNs in capturing inherent data characteristics. An SVM's ability to learn is independent of the dimensionality of the hyper plane. In tests SVMs tend to outperform the other classifiers.

The Boolean model is easily implemented and computationally efficient. Queries defined as Boolean expressions can be expressive and provide clarity, however they can be difficult to construct and are all or nothing. Results are not ranked and it provides no weighting of index or query terms. The Boolean model can be said to be more suited to data retrieval rather than

information retrieval. In comparison Vector Space models are not binary and allow ranking of results. However they have limited expressive power and are computationally more expensive than the Boolean model. There is the assumption that terms are independent. Because they produce low similarity values, long documents are inadequately represented by Vector Space models. Also the order of terms is lost in the Vector Space model representation. Probabilistic models also assume term independence. The probabilities are based on estimates so prior knowledge is required. Term frequencies within a document and document length are not taken into account. Unlike the Probabilistic models, Language models do not require initial estimates and term frequencies are factored into the results.

Chapter Three: Related Work

In this chapter we review papers that are particularly related to our work. Hence, the various techniques that are related to the local Web content observatory system and search engine are reviewed.

3.1 Search Engine

Search engines are special software tools that are designed to help people find information stored on the Web. The Web creates new challenges for information retrieval. The amount of information on the Web is growing rapidly, as well as the number of new users inexperienced in the art of Web search. People are likely to surf the Web using its link graph, often starting with high quality human maintained indices such as Yahoo or with other search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all important topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines. There are differences in the ways various search engines work, but they all perform three basic tasks:

- They search the Web based on important words using crawler component.
- They keep an index of the words they find using indexer component.
- They allow users to look for words or combinations of words found in that index using the query engine component.

There has been little research works conducted in Amharic, Oromiffa, and Tigrigna information retrieval in the past few years. The first Amharic search engine done by Tessema Mindaye [3] design and implement a search engine for Amharic Web documents that are written in Unicode based fonts. The other research done on the Amharic search engine was by Hassen Redwan Hussen [4] to re-design the Amharic search Engine in [3] in such a way that it is more speedy, considers the search of non-Unicode Amharic Web documents, and accommodates differently written Amharic words for the same meaning. The language identifier used in the previous

search engine was tested only on specific domains. In addition, these search engine are based on few seed URLs that are in the language of the search engine specification.

3.2 Web Content Observatory

Interactive multilingual on-line Language Observatory will serve as a reference tool regarding the practice of multilingualism for all project partners as well as for policy leaders and other stakeholders. The language observatory system identifies the type of language used on the Web. Dynamic content can be uploaded in any language and the Observatory will also act as a test-bed for multilingual (ML) tools.

An interactive on-line Language Observatory [46] has been developed which centralizes information on:

- The platform's policy recommendations;
- The motivators, scope and practice of multilingualism in various sectors of civil society;
- Best practice in the development of language policies;
- Best practice in the implementation of language policy;
- Multilingual tools;
- Allows for interaction among stake-holders.

The Language Observatory [46] is developed not to identify the language of the Web but to detect different tools that is used in teaching learning activity and language translation tool. The language is also identified manually after collecting online tools [47] that help for teaching learning process.

The work in [2] presents a good survey on identifying and promoting Local Internet Content in the Case of Ethiopia. The paper tries to investigate how much Internet access impacts an economy with some statically figure. The study shows that what makes Internet access more relevant and useful to a country is its local content. The measures of local content are classified in two categories in this paper: measure by particular economy and measure by particular language. Local content measurement by country includes: local content of indexed Web pages under country code, number of facebook subscribed per country, the number of online

newspapers, number of online radio stations, the number of online TV stations, Geo-tagged flicker photo and video uploaded on YouTube. Local content measurement by language indicates the number of pages per language. Measuring local content by the number of pages per language can be advantageous than using ccTLD contents, since this metric captures the .com, .net, .org, and other popular generic top-level domains which may contain local content. But there is no tool used in the work [2] that quantifies qualitative and quantitative data.

The Web is a medium for accessing a great variety of information stored in different parts of the world. Information is mostly in the form of unstructured data [48]. Web mining help in finding relevant information, extracting potentially useful knowledge and learning about consumers or individual users. Web content mining extracts information from Web page content. Data available on the Web is classified as structured data, semi structured data and unstructured data. The work in [49], the researchers used crawler to extract structured data from the Web. Base on this research work, the following are the problems in Web content mining.

- Data/information extraction
- Web information integration and schema matching
- Opinion extraction from online sources
- Knowledge synthesis and segmenting Web pages and detecting noise

The research work tries to address the problem of data extraction from the Web, but it didn't address the issue of Web language and the nature of data present on the Web.

3.3. Web Document Language Identification

Language identification is a classification task between a pre-defined model and a text in an unknown language. Most language identification systems are either based on short and frequent words or character n-grams.

It is obvious that short words are unique for a language and that they can be used for language identification. On the other hand, for efficiency reasons, not all words of a language can be used for language identification nor all words are known. All languages integrate new words into their vocabulary frequently. Many character sequences can be words in more than one language. Therefore, most approaches are based on common or frequent words [74, 75]. Typically, the

most frequent words in a trainings corpus of a known language are determined. The number of words used varies for example Souter et al. use 100 words [76]. Similar languages often share some common words and are therefore more difficult to be distinguished.

For short texts, word based language identification can easily fail, when a few words are present and these are not stored in the language model. Therefore, character n-grams have been used for identification as well and n varies between 2 and 5. This approach primarily focused on the occurrence of characters or n-grams unique for a specific language [77]. Current approaches store the frequency of the most frequent n-grams and compare them to the n-grams encountered in a text.

The work in [50] identifies the language of multilingual document over the Internet. The HTML and XML are standard encoding schemas used to create and form a Webpage. Both standards have attributes to specify the language content of the page. However, the reality remains that many Web pages do not make use of this attribute or, even worse, use it incorrectly and provide misleading information. The authors in [51] use byte sequence based n-gram algorithm to generate the language model. As previous studies are focusing on Latin-script based languages, most of them adopted a training corpus with a limited number of Latin-script based languages only. Thus, their research aims to improve language identification on a broader range of languages, especially for non-Latin-script and added support for Web page content. Language identification is done in two steps; creating a language model of the input text problem and comparing the created language model with language model of the training set problem. The algorithm used in this study adopted this general paradigm; however, it contains two new heuristics to properly handle Web pages. The first heuristic is to remove HTML tags in byte-sequence stream. The second heuristics is to translate HTML character entities to byte sequences of their Unicode code point. The target document's trigrams will then be compared to the list of byte-sequence based trigrams in every training language model. The problem in the work is translation of HTML character entities into byte sequence which is time consuming .It doesn't remove HTML tag data's which results in false positive output in language identification, and unable to determine correctly the language of multilingual Web documents.

The work in [52] investigates the language distribution of webpages in Asian languages. The number of pages found in each language is used to measure the presence of the language. The research work has three major objectives: presenting an overview of the Asian languages status on the Web, describing the state of multilingualism in the Asian country domain, and identifying the script and encoding issue of Asian languages. Unbicrawler [53] was used to download Web pages from 42 country domains with depth of 8. It took 14 days to complete crawling and gather around 107,141,679 Web pages. Language Identification Module (LIM) [54] was developed for language observatory projects, which is based on n-gram model, used as a language identification module in [55]. The survey reveals that the digital language divide exists at a serious level in the region. The state of multilingualism and the dominating presence of cross-border languages, English in particular, are analyzed. In order to promote language resource collection and sharing, authors have a vision of creating an observation-collection instrument for Asian language resources on the Web. The results of the survey show the feasibility of this vision, and provide them with a better idea of the steps needed to realize that vision. The experimental result indicates 55 Asian languages on the Web. Most of the Web content is found in pdf file, but the authors excluded it because of a technical difficulty in handling language identification. In addition, the authors didn't consider a Webpage with multi language content and the scope of the crawler is not limited to host, which results in crawl Web pages found outside a given ccTLD.

3.4. Summary

Although there are many initiatives on local Web content language identification, there is no work on the generic language observatory that can apply to all Web languages. The number of Web content in growing dramatically from time to time in terms of size, language and content type. Because language does not have specific characteristics, dealing with a limited group of languages associated with a country or a group of countries has no choice. In the case of Ethiopia so far there are research works on search engines, but there is little work done on local Web content observatory system.

Chapter Four: Design of a Local Web Content Observatory System

4.1 Introduction

The core objective of Local Web Content Observatory System is to provide a tool for generation of the Web content observatory system.

These goals can be further broken down into a set of high level requirements:

- Define Web services for a Local Web Content Observatory System that will provide the functionality of:
  - ✓ Crawling
  - ✓ Statistical information
  - ✓ Generating report
  - ✓ Progress report
  - ✓ Language identification
  - ✓ Categorization
- The services should provide monitoring and management interfaces.
- The interfaces should be flexible, extensible, and easily allow the addition of services and operations.

The following sections provide a brief discussion on the architecture of the proposed Local Web Content Observatory System.

4.2 Architecture of Local Web Content Observatory System

The Local Web Content Observatory System has six components. These are: Crawler, Categorizer, Language identifier, Statistics tracker, Content extractor and Report generator.

General architecture of Local Web Content Observatory System is shown in the Figure 4-1.

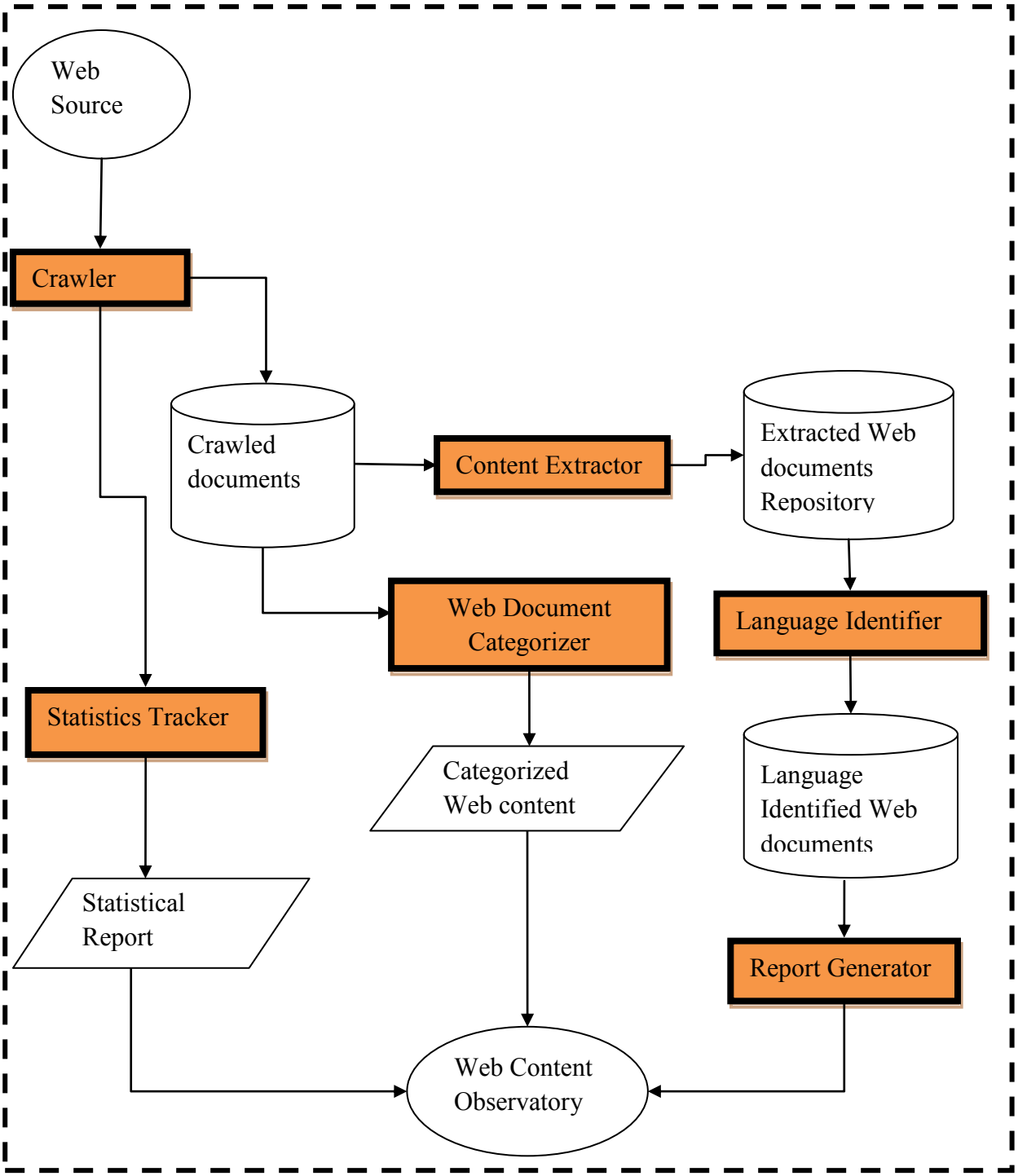


Figure 4-1: General Architecture of the Web Content Observatory System

4.2.1 The Crawler Component

The crawler component is composed of a multithreaded crawler (gatherer). A Web crawler systematically browses the World Wide Web, typically for the purpose of Web indexing.

For this research we modify and use open source Heritrix Web crawler [56]. The behavior of a Web crawler is the outcome of a combination of policies [57].

- a selection policy that states which pages to download,
- a re-visit policy that states when to check for changes to the pages, and
- a politeness policy that states how to avoid overloading Web sites

The Web Crawler is designed to be modular. Which module to use can be set at runtime from the user interface. The crawler consists of core class and pluggable modules. The core classes can be configured, but not replaced. The pluggable classes can be substituted by altering the configuration of the crawler. A set of basic pluggable classes is shipped with the crawler. The main reasons why we choose Heritrix Web crawler is because of:-

- Crawl the actual content of the Web document.
- Easy to develop new module and integrate it with the existing one.
- It is a Web user interface and easy to recover in case of a crash.

Crawl setup involves choosing and configuring a set of specific components to run. Executing a crawl repeats the following recursive process, common to all Web crawlers, with the specific components chosen:

1. Choose a URLs from among all those scheduled
2. A Domain Name System (DNS) resolution module that determines the Web server from which to obtain a URLs to be fetched.
3. Fetch the URLs
4. Analyze or archive the results
5. Select discovered URLs of interest, and add to those scheduled
6. Note that the URLs is done and repeat

The three most prominent components of the crawler are the Scope, the Frontier, and the Processor Chains, which together serve to define a crawl as shown in Figure 4-2 [65]. The Scope determines what URLs are ruled in or out of a certain crawl. The Scope includes the seed URLs used to start a crawl, plus the rules used in step 5 above to determine which discovered URLs are also to be scheduled for download.

The Frontier tracks which URL is scheduled to be collected, and those that have already been collected. It is responsible for selecting the next URLs to be tried (in step 1 above), and prevents the redundant rescheduling of already-scheduled URLs (in step 5 above).

The Processor Chains include modular Processors that perform specific, ordered actions on each URL in turn. These include fetching the URL (as in step 3 above), analyzing the returned results (as in step 4 above), and passing discovered URLs back to the Frontier (as in step 5 above).

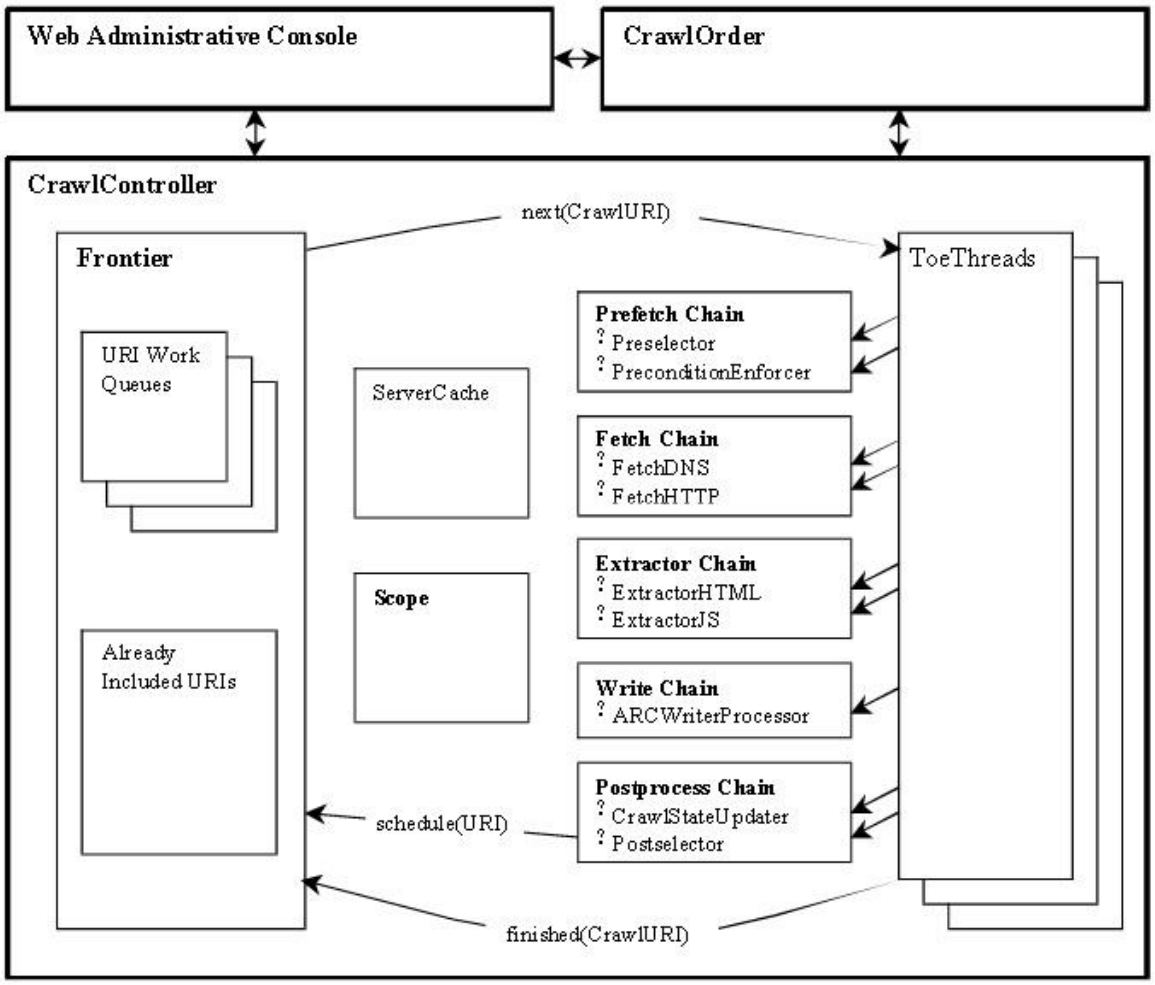


Figure 4-2: Basic Heritrix Architecture

The Web Administrative Console is a standalone Web application implemented by the embedded Jetty Java HTTP server. It is a Web user interface allowing the user to configure the crawler setting. The arrows in Figure 4-2 indicate the progress of a single schedule crawl URLs within one thread.

A crawl is started by passing the Crawl Order to the Crawl Controller, a component which instantiates and holds references to all configured crawl components. The Crawl Controller is the crawl's global context that all sub components can reach each other through it. The Web Administrative Console controls the crawl activity through the Crawl Controller.

The Crawl Order contains information about the scope of each instance crawling job and to create the Scope. The Scope provides initial seed URLs to the Frontier and cheek the new discovered URLs are within the scope.

The Frontier component selects URLs from the initial seeds and orders the URLs to be visited in a manner compatible with the configured politeness policy. The Frontier updates information periodically about the URLs that are visited and URLs that are going to be visited. The Frontier which keeps the state of the crawl includes, but is not limited to:

- What URLs have been discovered
- What URLs are being processed
- What URLs have been processed.

Each worker thread in Heritrix is called a ToeThread, The number of ToeThreads in a running crawler is adjustable to achieve maximum throughput given local resources. The number of ToeThreads usually ranges in the hundreds based on the performance of the computer.

The Server Cache Contains information about each server request response information such as IP addresses, robots exclusion policies, historical responsiveness, and per-host crawl statistics. The overall functionality of a crawler with respect to scheduled URLs is largely specified by the series of Processors configured to run. Each Processor in turn performs its tasks, makes up the Crawl URLs state, and returns. The tasks performed will often vary conditionally based on URLs type, history, or retrieved content. Certain Crawl URLs state also affects whether and which further processing occurs. For example, earlier Processors may cause later processing to be skipped.

Processors are grouped into processor chains, shown in Figure 4-3. Each chain does some processing on URLs. When a Processor is finished with URLs the ToeThread sends the URLs to the next Processor until the URLs have been processed by all the Processors. A processor has the option of telling the URLs to skip to a particular chain. Also if a processor throws a fatal error, the processing skips to the Post-processing chain.

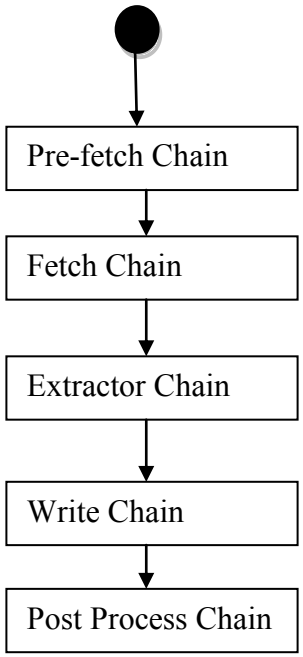


Figure 4-3: Process Chain

Processors are classified into five chains:

1. **Pre-fetch Chain** receives the Crawl URLs before any network activity to resolve or fetch the URLs. Such Processors typically delay, reorder, or veto the subsequent processing of a Crawl URLs; for example to ensure that robots exclusion policy rules are fetched and considered before URLs is processed.
2. **Fetch Chain** attempts network activity to acquire the resources referred-to by a Crawl URLs. In the typical case of an HTTP transaction, a Fetcher Processor will fill the request and response buffers of the Crawl URLs, or indicate whatever error condition prevented those buffers from being filled.
3. **Processors in the Extract Chain** performs follow-up processing on a Crawl URLs for which a fetch has already completed and extracting features of interest. Most commonly, these are new URLs that may also be eligible for visitation.
4. **Write Chain** stores the crawl results, returned content or extracted features to permanent storage.

5. Finally, **Post-process Chain** performs final crawl-maintenance actions on the Crawl URLs, such as testing discovered URLs against the Scope, scheduling them into the Frontier if necessary, and updating internal crawler information caches.

4.2.2 The Statistics Tracker

The Statistics Tracker is a module that monitors the crawl and records statistical data. Figure 4-4 shows the architecture of the statistical tracker.

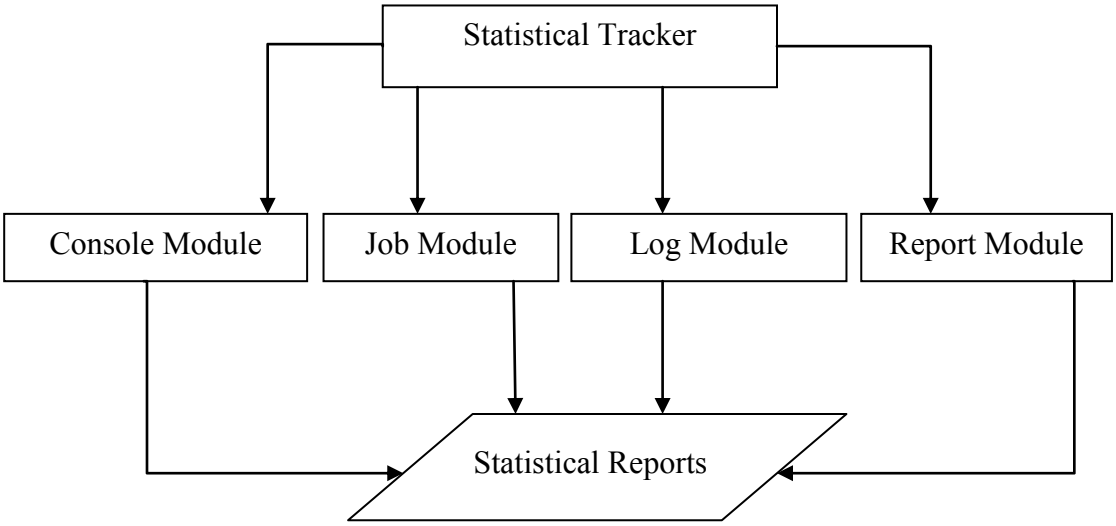


Figure 4-4: The Statistics Tracker

Generally statistics trackers gather information by either querying the data exposed by the Frontier or by listening for Crawl URLs disposition events and crawl status events. The Crawl Controller will start each statistics tracker once the crawl begins. If this facility is not needed in a statistics tracker (i.e., all information is gathered passively) simply implement the run () method as an empty method. Statistics Tracker is designed to write progress information to the progress-statistics.log as well as providing the Web user interface with information about ongoing and completed crawls. It also dumps various reports at the end of each crawl. The statistics Tracker has four modules: Console module, Job Module, Log Module and Report Module

The Console Module includes the following functionality:-

- It helps to submit seed URLs to be downloaded.
- Monitoring the progress of the crawler.
- Pause, checkpoint and terminate the current crawling job.
- Indicate how many URLs downloaded, queued and the number of hours and minutes taken to do the job.
- Alerts: - indicate if there is a problem to crawl URLs. On each crawling progress if there is a problem it indicates how many problems are found the type of problem happed.

The Job Module contains the configuration information about the crawler. We have made the change in the default configuration as presented in Appendix 1. The job module includes information about completed, pending and on the progress crawled jobs.

The Log Module provides information about the current progress logo report of the crawler. The main functionality of the Logs module includes:-

- Crawler logs: which URLs each trade is running
- Progress Statistics log: indicate the logs of discovered, queued at each time.

Reports Logs Module maintains information about on ongoing and current status of the crawler such as:-

- Crawl Report:-includes URLs, crawler, each domain status, MIMI type report.
- Seed Report: Seed Report provides information about which URLs the crawler is crawling, ignored and completed.
- Frontier Report: number of discovered and crawled URLs.

The status of each domain is determined by the HTTP reposed code the server return while the crawler downloads the pages. Algorithm 4-1 shows the algorithm to identify the status of each domain.

```
Do
        For each current crawling documents
        Request HTTP Responded Code
        Get status code
        Group each document based on the status code
        Update status code record
 While (end of crawling)
```

Algorithm 4-1: Algorithm to identify the status of each domain

4.2.3 The Content Extractor

The crawler downloads documents of the given seed URLs as archived files. The crawled content includes document other than the given domains, which biased the statistical data while identifying the language of a given domain. we adopted Heritrix content extractor [65] in a way to extract archived Web content that filter documents collected other than the given Country Code Top Level Domains (ccTLDs) and organize it in the respective domains. Algorithm 4-2 shows the algorithm to extract the content of archived crawled Web documents.

```
Input: Archived files
Do {
        Decompress archived file
        Read URL record
        if ( current record URL found in seed URL){
        create index file
        set a directory for each URL
        write the content into output directory}
        else
        break
} While (end of archived file)
Output: extracted Web documents
```

Algorithm 4-2: Algorithm for Archived Document Extractor

4.2.4 The Language Identifier

The language identifier module is used to identify the language of the crawled Web content. In this research work, n-gram model is used to detect the language of the crawled Web documents. The language identifier is aware of all proprietary encodings for Ethiopia Web pages. In this research, the technique of checking the META data of HTML pages for identification of Web pages is not applied. Ethiopia language Web pages use different encodings and cannot be specified by checking HTML META declaration or other language detection tools, such as Mozzilla charset detector.

The n-gram based language identifier is chosen for language identification due to its high accuracy in identification, its resilience to typographical errors, and its minimal data requirement for Training Corpus (TC). The n-gram language identifier adopted for this study is called Global Information Infrastructure Laboratory's Language Identifier (G2LI) [58]. The G2LI method first creates n-bytes sequences of each training text. Then it checks the n-bytes sequences of the collected page with those of the training text. The language having the highest matching rate is

considered to be the language of the Web page. The general paradigm of language identification can be divided into two stages. First, a set of language models is generated from a training corpus during the training phase. Second, the system constructs a language model from the target document and compares it to all trained language models, in order to identify the language of the target document during the identification phase. The architecture of language identification model is presented in Figure 4-5.

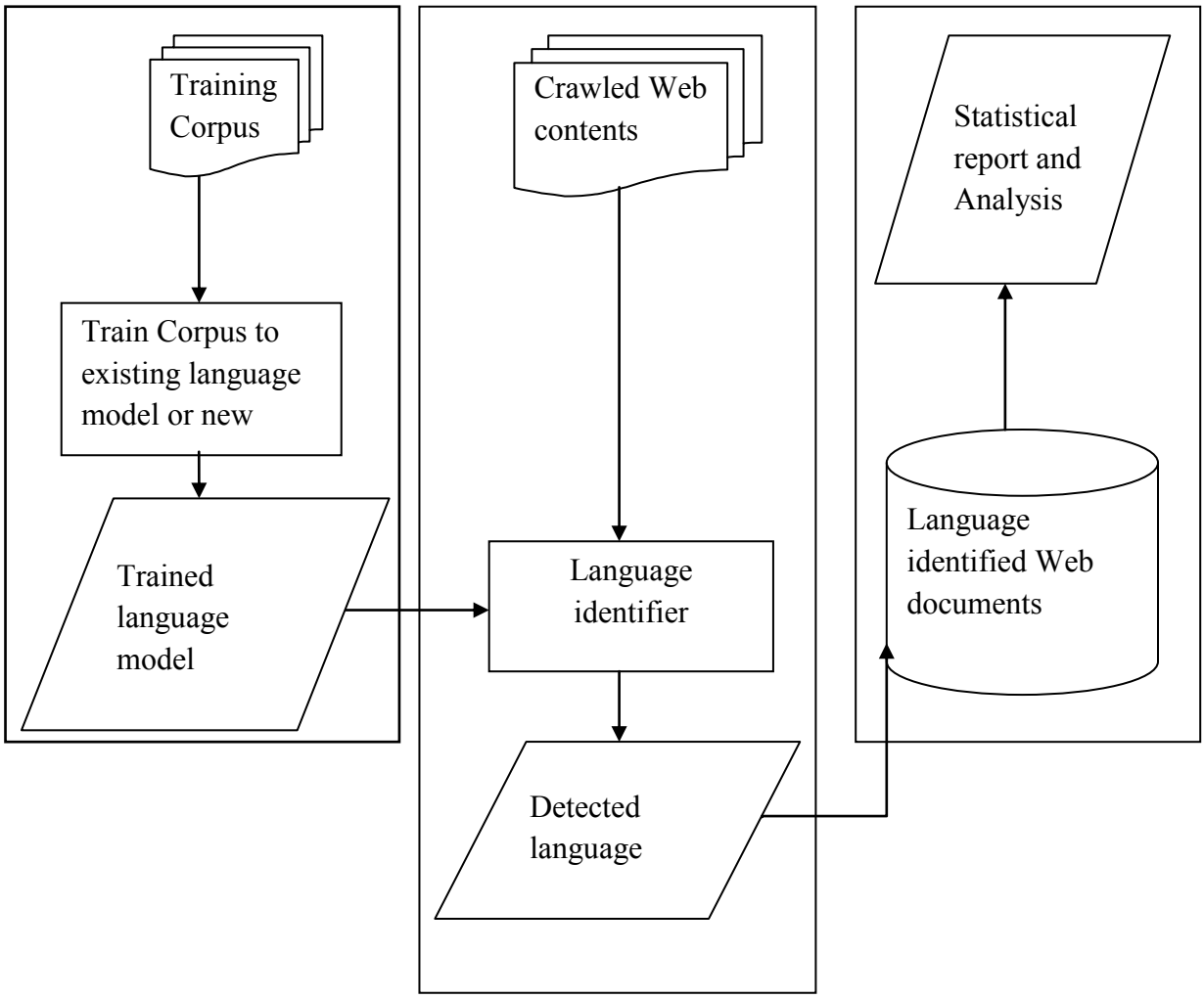


Figure 4-5: Architecture of Language Identification

The language identification method is based on n-gram frequency. The n-gram distribution vector of training document has no frequency information. Only the target document has a frequency-weighted vector. In order to detect the correct language of a target document, the algorithm will generates a list of trigrams from the target document, together with the frequency information of each n-gram. The target document's trigrams will then be compared to the list of

trigrams in every training language model. If a target's n-gram matches a n-gram in the training language model, its frequency value is added to the matching counter. After all trigrams from target document have been compared to trigrams in training language model, the matching rate is calculated by dividing the final matching counter by the total number of target's trigrams. If matching rate is equal for two or more language model it is considered as multilingual Web document. Finally detected language stored into the database for report generation and analysis.

1. Input : crawled Web directory
2. Count the number of Web documents and add to queue
3. Read from the queue
4. Identify the file type

   If (file type is gif, jpg, jpeg, png, bmp,mp3,mp4,css,js,asf)

   Break;

   Else

   Extract html tag if the file contains tags.

   generate a sequence of n-gram from the training corpus and the target document

   match the generated n-gram

   Identify the language of the file
5. Return the file name, the language detected, the number of match n-gram
6. Insert the result : file name and the detected language into the database
7. Repeat until all the documents are identified
8. Output: Identified Web documents

Algorithm 4-3: Algorithm for Language Identifier

After the language of each Web document is detected, the document name and the detected language were stored in the database for further processing. The algorithm for language identification model is presented in Algorithm 4-3.

4.2.5 Report Generator

The statistical information about identifying Web contents is available through the application user interface. The statistical report generator gathers information about the identified language from the language identifier component and provides summary information in the form of tables and different graphs. Algorithm 4-4 presents algorithms to generate statistical reports.

1. Input: language identified Web documents
2. Initialize record counter for identified language
3. Count the number of document identified per language
4. Get the number of count
5. Set the type of chart
6. Select random color for each language
7. Set animation frame time
8. Draw the chart
9. output: summery report of identified language

Algorithm 4-4: Algorithm to Generate Statistical Report of Identified Language

2.4.6 Web Document Categorizer

The simple crawl procedure would recursively crawl all outgoing links on processed pages and dump the results to the output file as raw data. Often, most of the crawled data have been totally irrelevant to the crawl subject (advertisements, unrelated pages, paid links, etc.). This makes manual detection of new interesting pages on the subject virtually impossible, as the user will have to go over all the raw data to find which pages yielded the most relevant results. This Component tries to solve this problem by categorizing a crawled Webpage and classifies the processed pages by subjects.

The Proposed Web document categorization architecture is shown in the Figure 4-6. Web content category is created from the cluster profile module. The cluster profile module is created based on the predefined class label (string). For this research work we identified the following class labels Education, News, Tourism, Business, Sport and Hacked. We implement Web document categorization based on the structure of the context of Web documents. Categorization by context exploits relevance hints that are present in the structure. Categorization by context is a technique for automatic Web page categorization based on the following hypotheses [73].

1. A Web page which refers to a document must contain enough hints about its content to induce someone to read it;
2. Such hints are sufficient to classify the document referred to it.

Indeed, a document would never be visited, except in casual browsing or through a direct referral, unless there were perceivable clues of its possible interest to potential readers. The categorization task in capable of identifying such hints. One obvious hint is just the anchor text of the link (i.e., the text between the <A> and </A> tags). But additional hints may be present elsewhere in a page: the page title, the section titles, list descriptions, etc. Our idea is to exploit the structure of HTML documents to extract such hints and to identify the number of hits (term frequency) present in each document. The algorithm to categorize Web document is show in the Algorithm 4-5.

After a cluster profile module has found class label keywords, it can use the keywords to generate queries and search for a similar document. Cluster profile module submits the queries to the search mechanism and gathers the document returned by the search, which are in turn reduced into a document vector. The new document can be used in a variety of ways. One option is to cluster the new Web content and the other option is to update the existing cluster by inserting a new document into the existing cluster. The query generator passes a query in the index component. A component called an indexer reads the repository, uncompressed stored Web documents and then parses the documents. Each document is then converted into a set of word-occurrences combination called hits. The hits record the word position in a document. In this instance a partially sorted forward index is created. The indexer also parses out all the links in every Web page and stores important information about the links in an anchors file.

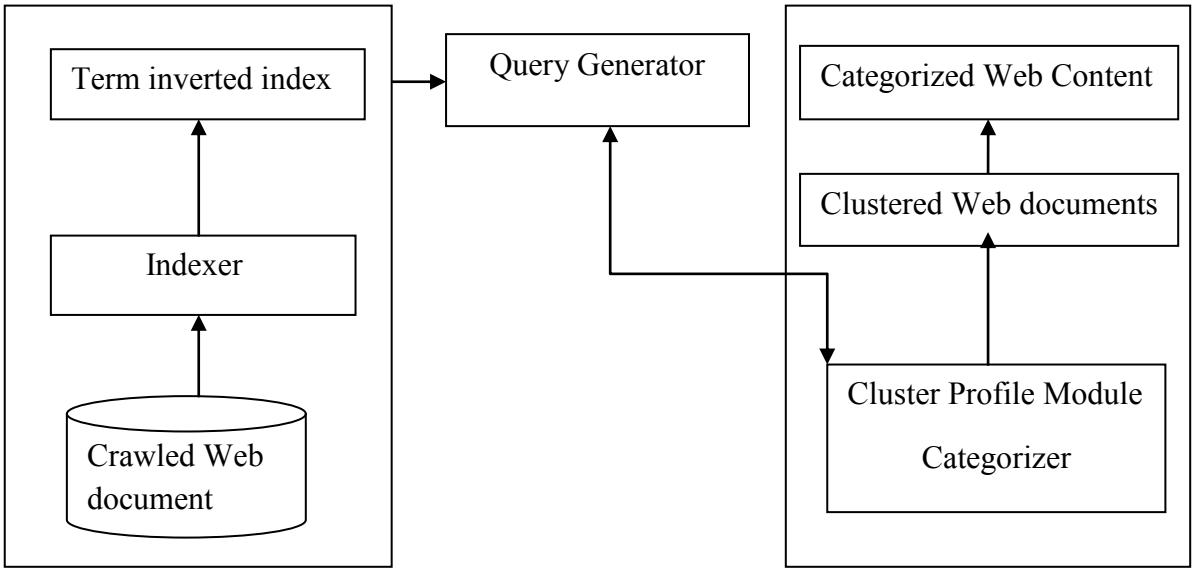


Figure 4-6 : Architecture of Web Document Categorizer

The crawled a Web contents parsed by the indexer and outputs terms-documents inverted index that will be used for clustering. The query generate search the inverted index for the given class label and return match result. Clustering Profile module receives terms-documents inverted index created by the indexer and creates k-frequent items sets. These are the most frequent term combinations in the index. Based on these combinations a clustering hierarchy of the crawled documents is created and each document will be classified to a best matching cluster.

1. Input : Class label
2. Read the crawled Web document
3. parse html page
4. Search Web pages and index terms that have match the class label

   do { if (Web document match class label)

   Categorize Web document into a class label

   else break

   while { end of crawled documents)
5. Output: Categorized Web documents and total matched number of pages

Algorithm 4-5: Web Document Categorization Algorithm

Chapter Five: Implementation

This Chapter details the implementation of the local Web content observatory system under the .et domain. Before describing our development environment we will first describe the open source crawlers, indexers and tool chains, which are used for the development of the Local Web content observatory system.

5.1 The Crawler

The following subsections discuss some of the open source crawlers that are available, and analyze their potential use as focused crawlers.

5.1.1 Heritrix

Heritrix [56] is the Internet Archive's Web crawler distributed under the GNU Lesser General Public License (LGPL). Written in Java, it has undergone continual development since 2004. It is widely used by several National Libraries. Heritrix stores the crawl data in ARC files. Each ARC file consisting of several URLs records along with some metadata, the Hypertext Transfer Protocol (HTTP) header and the response. The Local Web Content Observatory under the .et domain uses the Web crawler heritrix [56] version 14.4.4. To adopt it to Amharic, Oromiffa and Tigrigna Web document we configure a new job to crawl our list of URLs as indicated in Appendix 1. The configuration is performed on Web user interface and the file of the configuration located in the crawling directory as order.xml.

5.1.2 Methanol

The Methanol Web Crawling System [59] is a customizable Web crawler written in C developed by Emil Romanus distributed under the Internet Systems Consortium (ISC) license. It can handle HTML, Cascading Style Sheets (CSS), image, flash and video files. Methanol's primary component is its Web crawler Methabot which has been optimized for speed. The crawler has an extensible module system and supports user defined parsers written in JavaScript for additional file type support. A crawl can be configured for specific file types, each file type has an associated parser and potentially any user defined attributes. The parser is effectively a script or callback function that is invoked when the crawler encounters a specific file type. The parser is

responsible for extracting data from the file and can optionally set the user defined attributes. Methabot supports the Robots Exclusion Standard and is distributed and multi-threaded. Crawled information is stored in a MySQL database.

5.1.3 Grub

Grub [60] is a distributed Peer-to-peer (P2P) search crawler platform. Users can download a Grub client which runs when the user's computer is idle. The client crawls and indexes URLs and sends the information in a compressed format back to the central Grub server. Started in 2000, several versions were released under a closed license. It was released under the open source software license by Wikia Inc. in 2007, and was used for their open source Web search engine Wikia Search. It has achieved limited take up, and it is not a true peer to peer implementation because of the requirement for a central server which is a bottleneck.

5.1.4 The Choice for Relevant Crawler

Both Heritrix and Methanol provide architectures which support the plugging in of modules that could be used for focused crawling. They do not provide focused crawling functionality "out of the box". Grub does not support focused crawling and it is no longer being used by Wikia Search since 2009. Both the Heritrix and Methanol open source projects are active, but Heritrix has more usage and a more frequent release schedule. The latest version of Heritrix was released in 2014, but Methanol has not been updated in over a year. In addition Methanol has no recovery mechanism in case of system crash or network failure. For this research work we choose Heritrix as a Web crawler for the following reasons.

- Easy to monitor the progress of the crawling this came as Web user interface.
- Easy to integrate different plug-in
- Possible to configure the initial crawling setting after pausing
- Periodically report encountered error during crawling
- Possible to restore crawler state failed because of connection or power interruption.

Hence, we have chosen Heritrix as a crawler for our Web content observatory system.

5.2 Indexers

As discussed in Section 2.5, indexing refers to the process of representing the content to allow fast and accurate IR. The following subsections will discuss some of the open source indexing tools.

5.2.1 Lemur

Lemur [61] has been developed as a result of a partnership between Center for Intelligent Information Retrieval at the University of Massachusetts Amherst and the Language Technologies Institute at Carnegie Mellon University and is available under the Berkeley Software Distribution (BSD) license. Together they have developed the Lemur tool kit, an open source tool kit for the construction of language modeling and IR software. As part of this collaboration they have also built the INDRI search engine. The tool kit has been written in C and C++ and is designed to run in a UNIX environment but can also be run in Windows. It can index up to large-scale (terabyte) collections. It has built in support for English, Chinese and Arabic text. It supports Web, plain text, HTML, extensible Markup Language (XML), Portable Document Format (PDF), MBox (Unix mail box files), Microsoft Word, and Microsoft PowerPoint file formats. Indexing supports word stemming, retrieval can be achieved using language modeling approaches such as Indriand KL-divergence, as well as Vector Space, tf-idf, Okapi and InQuery.

5.2.2 Lucene

Lucene [62] is a text indexing and search library originally developed by Doug Cutting. It is currently available under the Apache License 2.0. It is written in Java and so has cross platform support, but it has also been ported to several other languages. It does not provide any HTML parsing functionality, so the text needs first to be parsed from the HTML or any other file format to be indexed before it can be processed using the Lucene API. Lucene provides ranked search results and allows querying by phrase, wildcard, proximity, range and more. Text analysis tools are included which facilitate text normalization, stop word removal and stemming. Lucene can support tf-idf term weighting and a combination of the Boolean and Vector Space model are used to compute query-document similarity.

5.2.3 Xapian

Xapian [63] is open source IR library which uses Boolean and Probabilistic modeling available under the GNU General Public License (GPL). Written in C++ with binding to several other languages, it also has cross platform support. Several large organizations are using Xapian including One Laptop per Child, Delicious and Debian. Xapian allows the easy addition of search facilities and indexing to applications by developers using its adaptable toolkit. It has the capability to scale to hundreds of millions of documents. It allows phrase and proximity searching, probabilistic ranking and relevance feedback. Boolean querying and wildcard search are supported. It can be combined with Omega to add search engine facilities to an Intranet or Web site.

5.2.4 Nutch

Nutch [64] is an open source Web search engine written in Java and released under the Apache License 2.0. The architecture allows components to be plugged in and out for activities such as file type parsing, querying, clustering and content retrieval. It provides a crawler for fetching and indexing pages and also a search for responding to user search requests. The crawler can be broken down into a fetcher for downloading content and extracting links, a Web database for storing URLs and harvested content and an indexer which generates an index made up of key-words for each downloaded page. Crawling on a scale of a local file system up to the World Wide Web is supported. Lucene [62] is used to provide indexing and search capabilities. Nutch designed for Windows and UNIX operating system.

For this research work we choose Lucene as indexer. The reason that we use Lucene for indexing and searching purpose is because of its performance, scalability and extensive adoption capabilities. Concerning performance, typical search durations are in milliseconds even for large collection. Lucene is selected for its data structures for the index (inverted index), its ability to consider partial matching (vector model), its support of different Boolean operators in queries, and its support of different kinds of queries.

5.3 The Development Environment

The development environment that is used in the development of the Local Web Content observatory system under the .et domain is described below. Our system is developed and tested on both personal and laptop computers.

- Personal Computer of Intel core i3 processor with 2.00GHz speed, 4.0GB of RAM, 500GB of hard disk capacity, with Windows/Ubuntu operating system.
- Laptop Computer of Intel core i5 processor with 2.43GH speed, 4.0GB or RAM, 600GB of hard disk capacity, with Ubuntu and Windows 7 operating system. We used a bandwidth of 20 Mbps.
- Other software components used to develop and test our system are different browser, Java 1.7.2, ASP, JSP, PHP, Nutch-0.9, Lucene-2.0, Apache Tomcat 6.0, Heritrix 3, G2LI, and Cygwin.

5.4 The Development Tools

During the development of Local Web content observatory under the .et domain, we select and utilize the following tools. These modifications were necessary because of software obsolescence and also because of the requirements (such as to fit for local language such as for Amharic, Oromiffa and Tigrigna.) of the Local Web content observatory under the .et domain. During the development of the Local Web Content Observatory system under the .et domain, we have selected and utilized the following tools.

Apache Tomcat 8.0

Apache Tomcat 8.0[2] is a Java based Web container that is used for handling HTTP requests and conveying responses from different components. For this work, we use the version 8.0 of Apache Tomcat. We used it as a container for our qualitative content report and to run PHP and ASP script.

[2] http://tomcat.apache.org

JDK

JDK is an implementation of either one of Java SE, Java EE or Java ME platforms. It includes a private JVM and a few other resources to finish the recipe to a Java application. We installed jdk1.7.0_45 (Java Development Kit version 1.7.0_46)[3] on a Windows environment. Then we used Java programming language for implementing various components of our system.

Netbeans7.3

Netbeans[4] is an integrated development environment for developing primarily for Java, but also other language, in particular PHP, C/C++ and HTML5. For this research work we used Netbeans 7.3 version to implement the language identifier and the crawler components.

Macromedia Dreamweaver 8

Macromedia Dreamweaver 8[5] provides a combination of visual layout tools, application development features, and code editing support, enabling developers and designers at every skill level to create visually appealing, standards-based sites and applications quickly. We have used Dreamweaver for code editing and Web interface development.

5.5 The Crawler Component

The crawler component of the system implemented by adopting Heritrix [56] open source crawler. Thus, we set the configuration properties that fit to our local Web content observatory system .We adopted Heritrix to crawled local Web documents in a host domain and to return the status of each Web documents while crawling. The crawler collects documents of 2000 seed URLs hosted under the .et domain. Depth tells how many times to crawl the Web page. To get all the available files, a depth of 10 is used. For this crawling, 15 threads that fetch in parallel are used. The goal of running multiple processes in parallel is to maximize the download rate while minimizing the overhead from parallelization.

[3] http://Java.sun.com
[4] https://netbeans.org

[5] http://www.adobe.com/products/dreamweaver/

The following commands start the crawler:

% export HERITRIX_HOME=/PATH/TO/BUILT/HERITRIX

where $HERITRIX_HOME is the location of extracted heritrix .Next run:

% cd $HERITRIX_HOME

% chmod u+x $HERITRIX_HOME/bin/heritrix

% $HERITRIX_HOME/bin/heritrix --admin=LOGIN:PASSWORD

The default heap memory assigned for java heap memory is 512MB, which is not enough to crawl a millions of pages, to enable the crawler not to crash during crawling the following command help to increase the heap memory .

% JAVA_OPTS="-Xmx4096m" $HERITRIX_HOME/bin/heritrix

Once Heritrix installed and logged into the Web user interface, the following three steps start crawling activity.

Step 1: Create a job

To create a new job we choose the Jobs tab, this will take you to the Jobs page. Once there you are presented with three options for creating a new job. Select with default configuration, this will create a new job based on the default profile or select new configuration, to specify new configuration setting for the crawler.

Step 2: Configure the job

Once a new job is created, there are a configuration parameter such as specifying the depth, the number of thread, user agent, the scope and others rules. We create new job configuration to crawl seed urls under the .et domain as presented in Appendix 1.

Step 3: Running the job

Submitted new jobs are placed in a queue of pending jobs. The crawler does not start processing jobs from this queue until the crawler is started. While the crawler is stopped, jobs are simply held. To start the crawler, click on the console tab. Once on the Console page, you will find the option Start at the top of the crawler Status box, just to the right of the indicator of current status. Clicking this option will put the crawling into crawling Jobs mode, where it will begin crawling any next pending job, such as the job you just created and configured.

5.6 Statistical Tracker

The Statistics Trackers implement the Statistics Tracking interface. Statistics Tracking interface initialization method provides the new statistics tracker with a reference to the crawl controller and thus the module has access to any part of the crawl. For new statistics tracking modules to be available in the Web user interface their class name must be added to the StatisticsTracking.options file under the conf/modules directory. The classes full name (with package info) should be written in its own line, followed by a'|' and a descriptive name (containing only [a-z,A-Z]). The Statistics Tracker is designed to write progress information to the progress-statistics.log file as well as providing the Web user interface with information about ongoing and completed crawls. It also dumps various reports at the end of each crawl. The Web Console presents an overview of the status of the crawler while crawling as shown in the Figure 5-1.

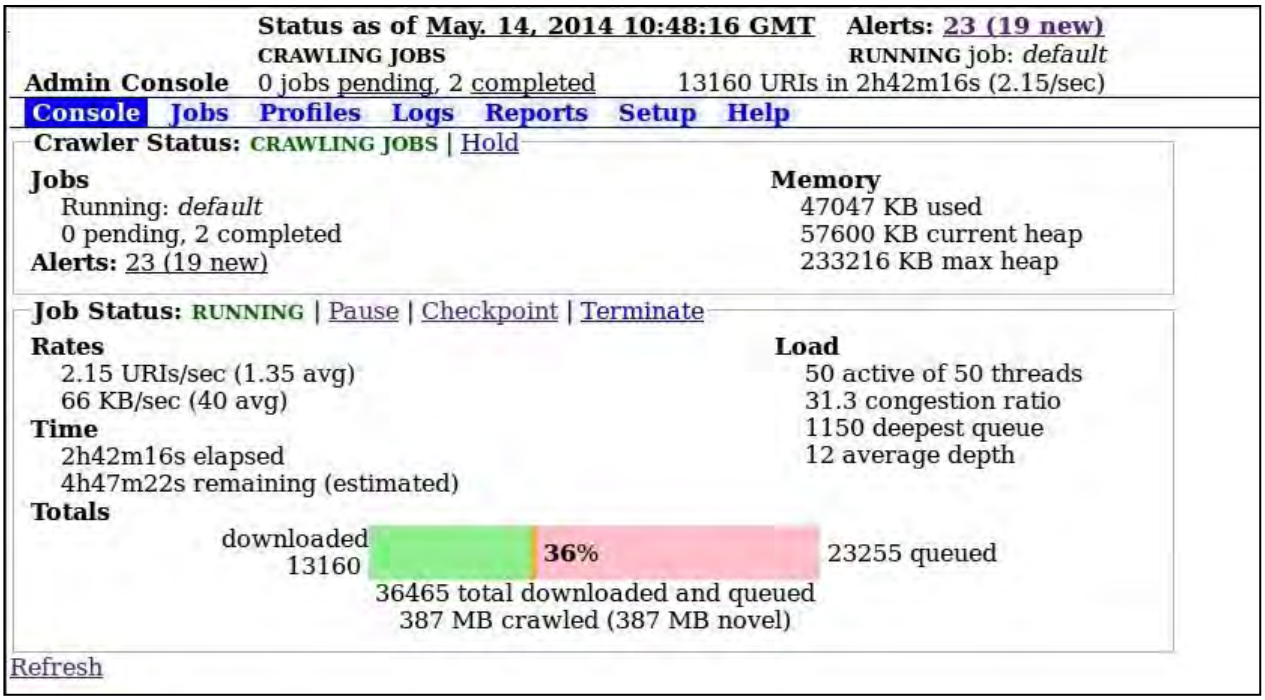
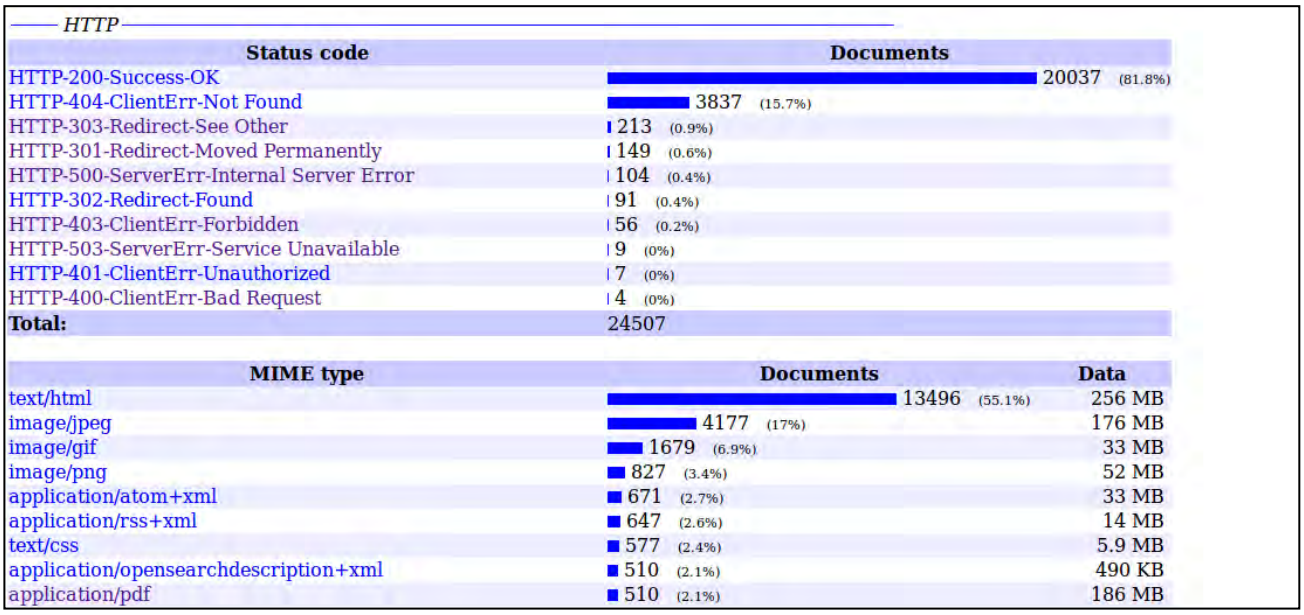


Figure 5-1: The Crawler Consol

The Report Component generates report about crawler, frontier and seed report. The report are statistics about the number of URLs pending, discovered, currently queued, downloaded etc.

Figure 5-2 shows the crawler report generated in the middle of crawling indicating the status of crawled Web documents.



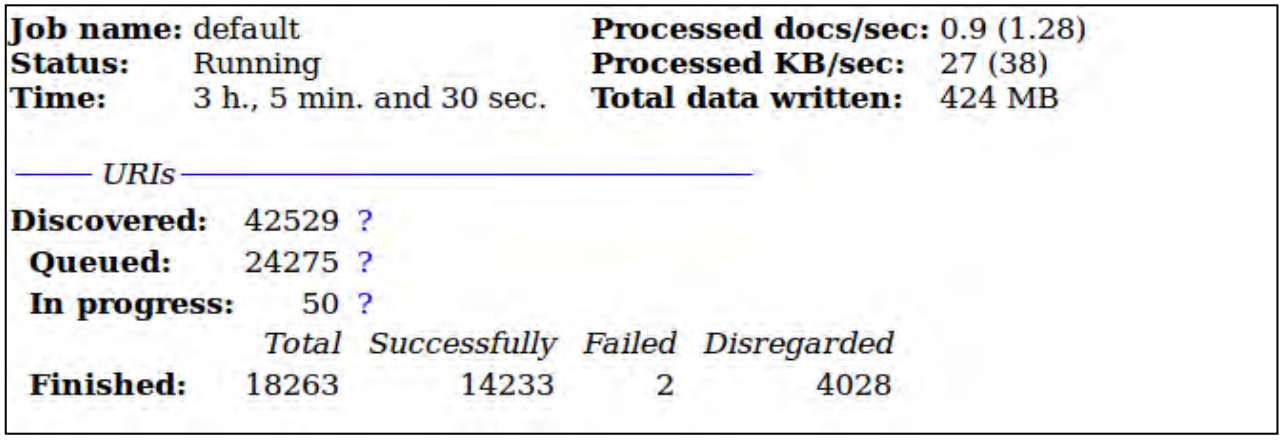
HTTP

Status code	Documents
HTTP-200-Success-OK	20037 (81.8%)
HTTP-404-ClientErr-Not Found	3837 (15.7%)
HTTP-303-Redirect-See Other	213 (0.9%)
HTTP-301-Redirect-Moved Permanently	149 (0.6%)
HTTP-500-ServerErr-Internal Server Error	104 (0.4%)
HTTP-302-Redirect-Found	91 (0.4%)
HTTP-403-ClientErr-Forbidden	56 (0.2%)
HTTP-503-ServerErr-Service Unavailable	9 (0%)
HTTP-401-ClientErr-Unauthorized	7 (0%)
HTTP-400-ClientErr-Bad Request	4 (0%)
Total:	24507

MIME type	Documents	Data
text/html	13496 (55.1%)	256 MB
image/jpeg	4177 (17%)	176 MB
image/gif	1679 (6.9%)	33 MB
image/png	827 (3.4%)	52 MB
application/atom+xml	671 (2.7%)	33 MB
application/rss+xml	647 (2.6%)	14 MB
text/css	577 (2.4%)	5.9 MB
application/opensearchdescription+xml	510 (2.1%)	490 KB
application/pdf	510 (2.1%)	186 MB

Figure 5-2: Sample Crawler Report

The frontier report provides information about discovered, queued and in progress URLs. Sample frontier report is shown in the figure 5-3.



Job name: default
Status: Running
Time: 3 h., 5 min. and 30 sec.
Processed docs/sec: 0.9 (1.28)
Processed KB/sec: 27 (38)
Total data written: 424 MB

URIs

Discovered: 42529 ?
Queued: 24275 ?
In progress: 50 ?

	Total	*Successfully*	*Failed*	*Disregarded*
Finished:	18263	14233	2	4028

Figure 5-3: Sample Frontier Report

5.7 The Content Extractor

The heritrix crawler writes the crawled Web content as Internet Archive compressed version 1 ARC files. The compressions are done with zip, but rather compress the ARC as a whole, and instead, each ARC Record is in turn zipped. All zipped records are concatenated together to make up a file of multiple zipped members. The ARC file metadata contains information about the host name and IP address of each host. The content extractor extract the content of each ARC file, organize it in its respective domain and match the host name with the seed URLs to cheek if it exist in the seed URLs. While extracting the content of the crawled Web documents index file is created as a reference to search crawled Web contents. The index operation uses the StateJobReader class to verify that the crawling process did not generate an error status and that it is not currently running, otherwise an IndexFault class is returned to the user. If this can be verified, crawling process generated some ARC files to be indexed. When a crawling process finishes it can often take a few minutes for the process to close all the open ARC files. If any open ARC files exists, an IndexFault is returned to the user. If no ARC files exist, an IndexFault is returned to the user. If only closed ARC files exist, then the prerequisite that the crawling process completed successfully can be said to have been met. If the training or crawling processes are currently running an IndexFault response is returned to the user. By default the Overwrite Files parameter in the index operation request is set to false, in this case, if an indexing process is currently running, an IndexFault response is returned to the user. If Overwrite Files is set to true, the indexing process will be killed using the kill PID key.pl script, any previous index data is deleted using the clean index files.pl and a new indexing process launched.

5.8 The Language Identifier

Language observatory module under Local Web Content Observatory under the .et Domain adopts Global Information Infrastructure Laboratory Language Identifier (G2LI) [58] to identify the language of each Web page under the .et domain. For this research work, we focused our study on 2000 domains hosted under the .et domain. We made a modification on G2LI to identify the language of bulk crawled Web document and to insert the result into the database for further processing as shown in Appendix 2.G2LI is trained by three different corpuses to improve the performance of the identification. Figure 5-4 shows the user interface of the language identifier.

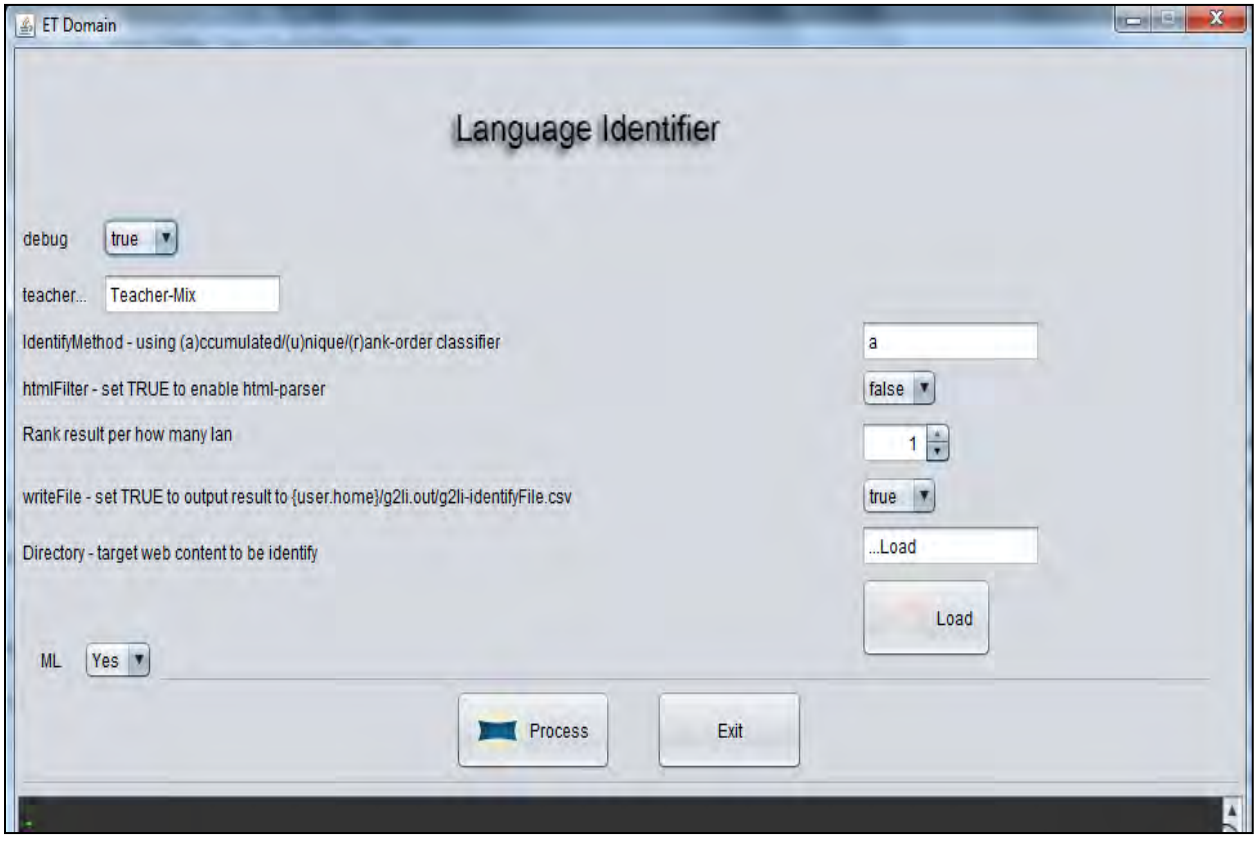


Figure 5-4: Language Identifier User Interface

A user can select a set of parameter and the crawled Web document directory as shown in the figure 5-6.The parameters are HTML parser, identification method, ranking of result per how man language and specifying the directory of the crawled Web documents. The HTML parser

parses the Web page in a linear fashion. It searches for HTML tags from the beginning to the end of page. It looks for valid HTML start and end tags and marks all blocks of HTML tags. The parser removes all detected HTML blocks and return remaining content for language identification. The language identifier generates n-gram from parsed HTML together with the frequency information of each trigram. The matching process for detecting a language can be summarizing as below [58].

1. Let N be the number of trigrams in target document.
2. All the trigrams from the target document u1, u2, ..., uN are listed. Let uj be the j^{th} n-gram in the target language model.
3. Let Ti be the i^{th} language model in the training corpus. Ri (or R-values) is calculated from every i^{th} language model using equation (1), where Ri is the rate at which the set of trigrams in i^{th} language model of the training corpus appears in the target document.

   $R_i = \sum_{i=1}^{n} \frac{f(U_j)}{n}$ where, $f(U_j)$ 1 if $uj \in T$ otherwise 0.

The highest matching returned as a language of a given Web document. Detected language and the file name are stored in the database to generate statistical report.

5.9 The Report Generator

The Report Generator module generates information about the number of documents identified per language. The report is presented in the form of different charts and tables. Figure 5-5 shows the script used to generate a report in the form of a chart.

```
<script type="application/Javascript">
                    var chart1 = new etChart('chartCanvas1');
                    chart1.data = [
                    <?php
                    $query = mysql_query("select * from sts") or die(mysql_error());
                    while ($row = mysql_fetch_array($query)) {        ?>
                      <?php echo $row['cn'] . ','; ?>
                    <?php }; ?>
                    ];     chart1.labels = [
                    <?php
                    $query = mysql_query("select * from sts") or die(mysql_error());
                    while ($row = mysql_fetch_array($query)) {
                      ?>        <?php echo "'" . $row['lan'] . "'" . ','; ?>
                    <?php }; ?>
                    ];
                    chart1.colors = ['#006CFF', '#FF6600', '#34A038', '#945D59', '#93BBF4',
'#F493B8'];                                    chart1.randomColors = true;
                    chart1.animate = true;
chart1.animationFrames = 90;
                    chart1.draw();                    </script>
```

Figure 5-5: A Script to Generate Statistical Report of Identified Language

A user selects the type of chart to view the statistical report and the result is displayed in the form as shown in the Figure 5-6

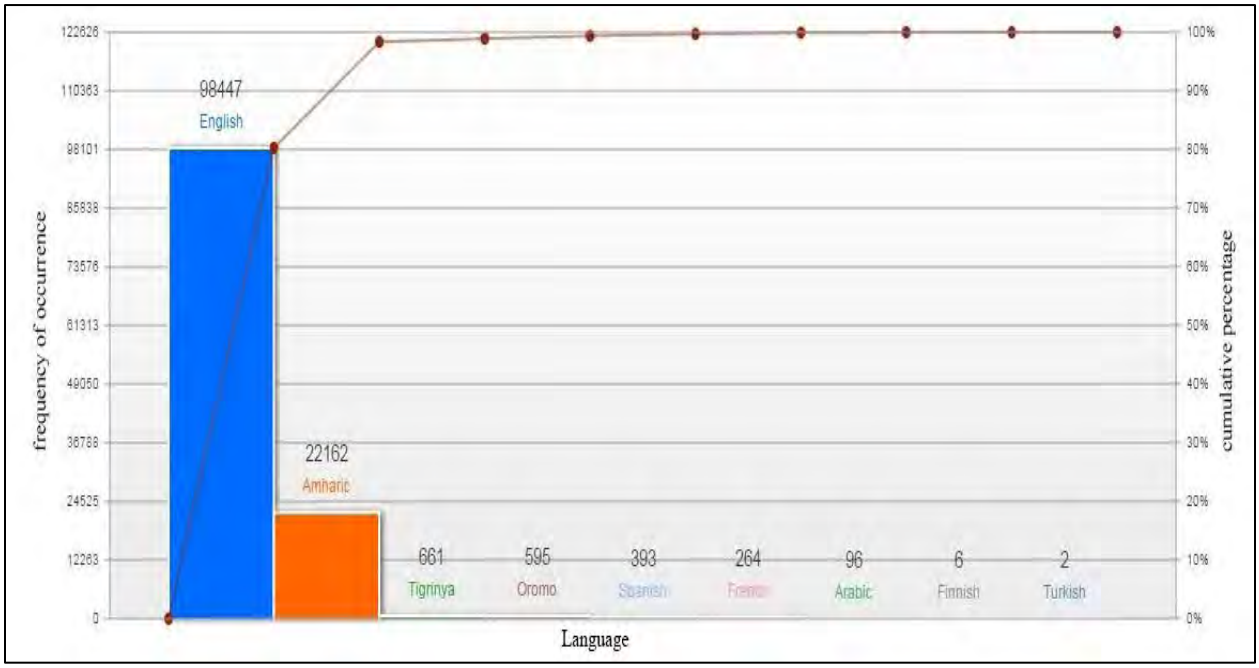


Figure 5-6: Report Generated using Pareto Chart

5.10 Web Document Categorizer

There are different algorithms for Web content categorization. For this research work we select a Web document structure to categorize a given Web document into one of the predefined classes. A document would never be visited, except in casual browsing or through a direct referral, unless there were perceivable clues of its possible interest to potential readers. To categorize a document we extract the meta data information such as title, and consider term frequent in the document. The result returned based on the relevance of each document for a given class label. The returned result includes the total number of pages that matches a category. The system is accessed through Web user interface as shown in the Figure 5-7.

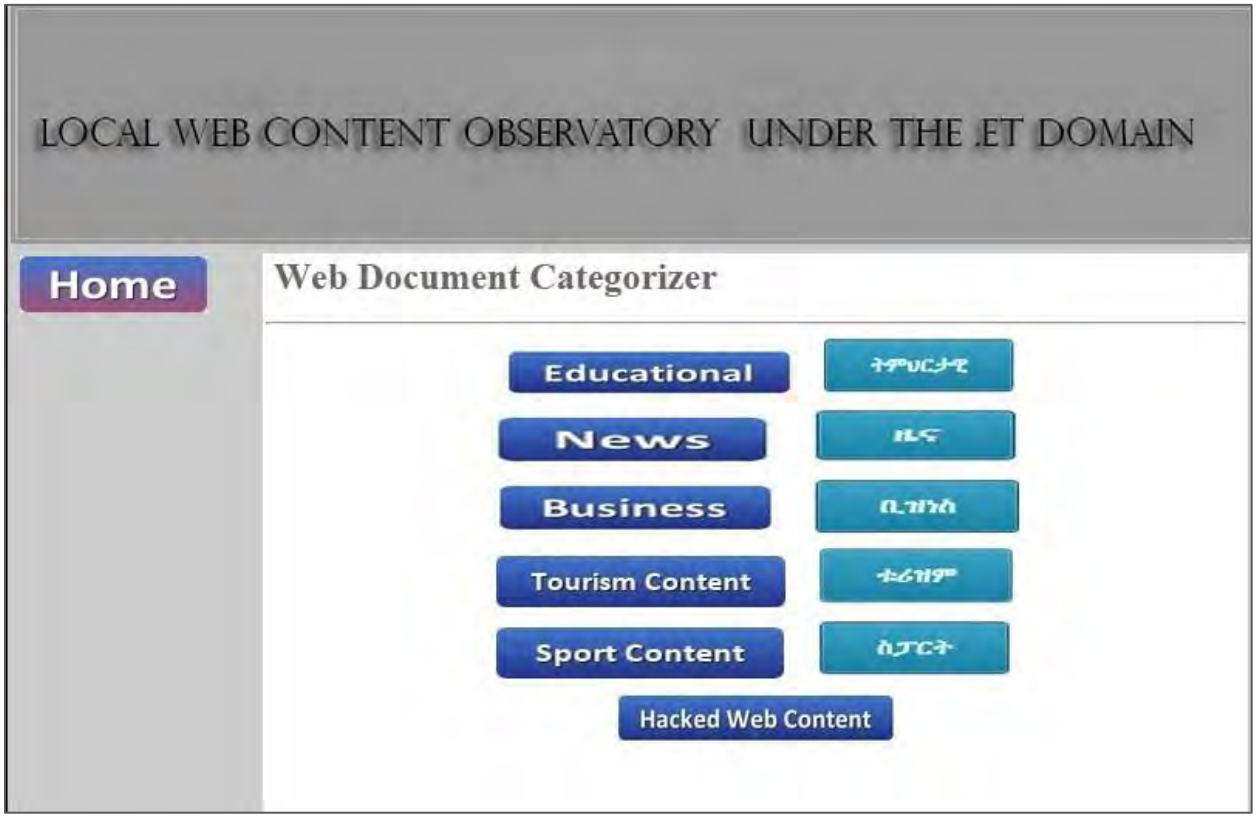


Figure 5-7: Web Content Categorization User Interface

Chapter Six: Experimental Result

In this chapter we evaluate the performance of the Local Web Observatory System under the .et domain. From the research objective point of view we will evaluate the following evaluation points.

- The performance of the crawler
- The performance of the language identifier and categorizer
- The performance of the Web content categorizer
- Investigate the performance and integration issue of a rise in building Local Internet observatory system under the .et domain

First we search the crawler then the language identifier module indentifying the crawled Web content and finally categorized Web documents.

6.1 The Crawler of the Local Web Content Observatory System

We evaluated the performance of the crawler of the local Web content observatory system under the .et domains. The crawler operation was invoked two times on different days for 2000 seed URLs. The crawling service was performed at the data center of Addis Ababa University. We used depth of 10 and 15 threads to accomplish the crawling activity. The connection speed was 20 Mbps. Crawling activities done two times, during the first run the crawler collect 192,390 documents in 16 hours and 50 minute. In the second run, the crawler collects 263,031 documents in 2 days and 17 hours. After extracting archived crawled Web document using content extractor we verify that, the crawler downloaded different document types, videos, and audio, zip files and each documents are organized into its respective domains.

Table 6-1: Status of Domains under the .et ccTLD

No	Respond Code (HTTP)	Description	Number of Documents	Selected Sample for Evaluation	Accuracy Rate
1	200	Success- Ok	200,000	1500	97%
2	404	Client Err – not Found	5025	300	98%
3	302	Redirect Found	302	100	98%
4	301	Redirected-Moved Permanently	63,031	800	97.5%
5	500	Server Err – Internal Server Err	160	100	98%
6	303	Redirect – see other	123	123	100%
7	403	Client Err Forbidden	81	81	97.5%
8	401	Client Err- unauthorized	17	17	100%
9	503	Server Err – service unavailable	17	17	100%
10	400	Client Err- Bad Request	10	10	100%
11	204	Success – No Content	2	2	100%
Average					98.72%

The challenges we faced during crawling are Heritrix memory leaks, power fluctuation and connection interruption. In better crawler performance result the best computing environment is required, such as high performance computer servers, high connection speed and implementing distributed crawling techniques.

To evaluate the accuracy of the crawler HTTP respond code report of the statistical tracker, we took samples from the result and cheek page by page. Table 6-1 shows the performance of the crawler status code report. The result shows that accuracy rate of 98.72% which is a promising result.

6.2 The Language Identifier

There are four sets of data used in this study. The first three set are the training corpus, which contains training data used to train the language models. The fourth set is the evaluation corpus, which is a collection of crawled Web pages used as target documents in the experiments.

In this research work, we prepared three sets of training data. The first set of training data, training corpus A, is constructed from 565 Universal Declaration of Human Rights (UDHR) texts collected from the Office of the High Commissioner for Human Rights (OHCHR) Web site and Language Observatory Project (LOP). UDHR was selected as it is the most translated document in the world, according to the Guinness Book of Records.

The OHCHR Web site contained 394 translations in various languages. However, 80 of them are in Portable Document Format (PDF). As a result, only 314 languages were collected from OHCHR. The LOP contributed 18 new languages. The total size of the first set of training data is 15,241,782 bytes. Individual file size ranged from 4,012 to 55,059 bytes.

The second set of training data, training Corpus B, increases the number of languages by 33. It contains 65 (some are same language but in different encoding schemes) Biblical texts collected from the United Bible Societies (UBS). All files have similar content, but written in different languages, scripts and encodings. The total size of the second set of training data is 1,232,322 bytes. Individual file size ranged from 613 to 54,896 bytes.

The third set of training data, Training Corpus C, contains Amharic, Oromiffa and Tigrigna corpus collected from Wikipedia, from previous research works and different Web contents. The total size of the training data is 3,996,125 bytes.

Most languages have more than one training file in the training corpora. This is because the same language can be written in different scripts and encodings. We evaluated the language identification module in two experiments. In the first experiment we trained the language model using Training corpus A and B and evaluate the language identification. Experiment one is used as a baseline for a comparison. After manually inspecting the webpages and the result manually we found 10 out of 90 incorrectly identified because of unavailability of the language model. This is an evidence that training corpus A and B alone was inadequate led to a decision to expand the training corpus and conduct the second experiment. The second experiment was designed to evaluate the improvement achieved through extension of the training corpus. The overall performance of the language identification for two experiments is shown in the Table 6-2.

Table 6-2: Training Corpus Performance Result

Experiment	**One**	**Two**
Training Corpus	A and B	A ,B and C
Accuracy rate	88.89%	98.67%

We compared adapted algorithm with other language identification algorithm LIM [54] for randomly selected 100 domain Webs contains. The performance result of the comparison is presented in the Table 6-3. The results show adopted G2LI algorithm accuracy is higher than that of LIM.

Table 6-3: Performance Evaluation between LIM and Adopted G2LI

Algorithm	**LIM**	**G2LI**
Correct/total	78/100	98/100
Accuracy rate	78%	98%

Our crawler collects around 263,031 different webpages, which is different file format, the language identifier module ignores some extension files that don't provide a hint for the language of the Web such as different image file format, js, css, zip, rar, different audio files, swf and asf files. Each collected Web document language is identified iteratively by the language identifier module and the summary result is shown in the Figure 6-1. The results of the language identification indicate that most of the Web documents under the .et domain are dominated by English contents.

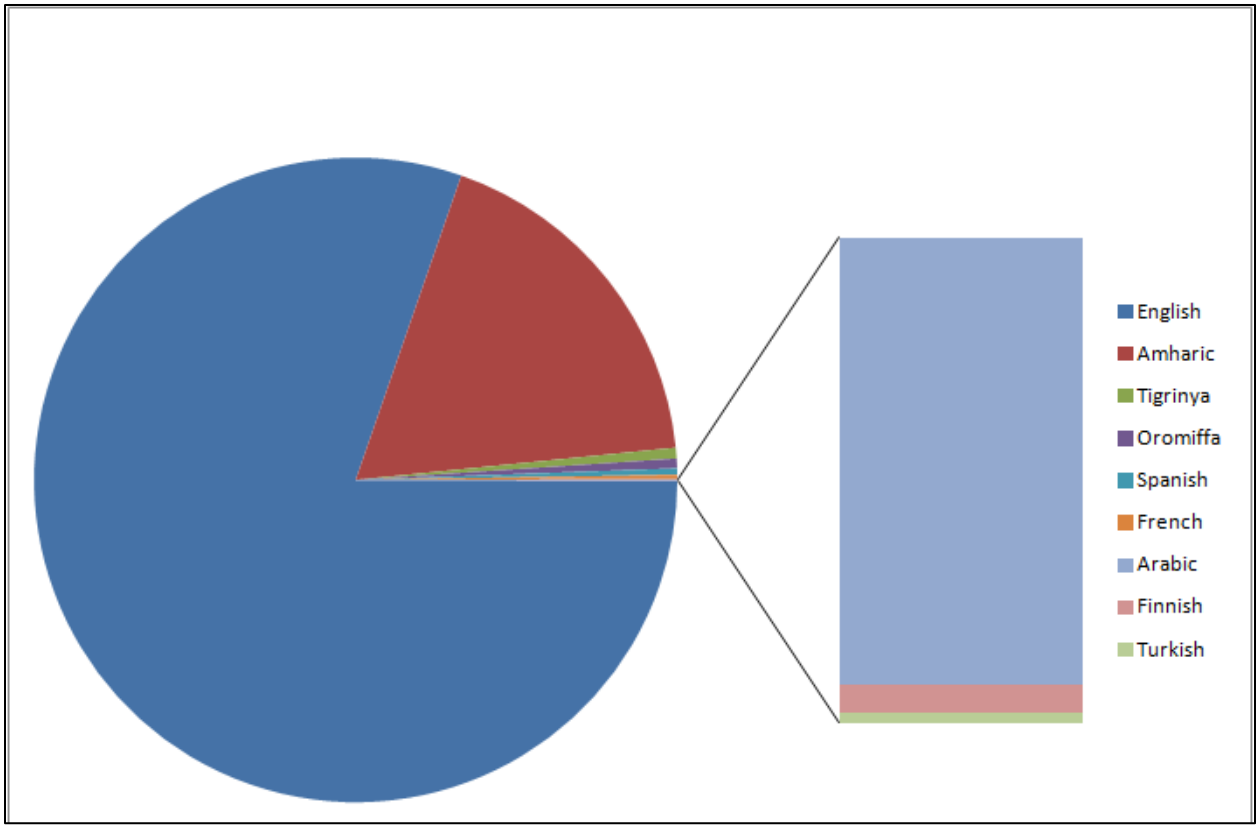


Figure 6-1: Identified Web Documents Per Language

Table 6-4 summarizes the identified Web documents per language under the .et domain.

Table 6-4: Identified Web Documents Per Language

No	Language	Number of Web documents
1	English	98447
2	Amharic	22162
3	Tigrinya	661
4	Oromiffa	595
5	Spanish	393
6	French	264
7	Arabic	96
8	Finnish	6
9	Turkish	2

6.3 Web Document Categorizer

Web documents categorizer categorize crawled Web documents based on meta data information and frequent item sets. Table 6-5 shows the result of categorization for the English language. Amharic Web document categorization is shown in Table 6-6. To evaluate the efficiency of the categorizer we used precision, recall and F-measure. Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is the ratio of the number of relevant documents retrieved to the total number of relevant documents. F-measure is a harmonic mean of Recall and Precision. That is:

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precsion} + \text{Recall}}$$

To measure Precision, Recall and F-measure we identified categorized Web documents as True positive (TP), False Positive (FP) and False Negative (FN). TP is correctly identified for a given class, FP is incorrectly identified into the given class, whereas FN is incorrectly classified into another class.

Table 6-5: Evaluation of Categorized English Web documents

Class Label	**Categorized Web documents**	**Evaluation**					
		TP	**FP**	**FN**	**Precision (%)**	**Recall (%)**	**F- Measure (%)**
Education	1186	1123	63	59	96.64	95	95.81
News	3,316	3121	195	67	94.1	97.9	95.96
Business	1,095	1062	33	8	97	99.2	98.09
Tourism	346	340	6	1	98.2	99.7	98.94
Sport	186	150	36	14	80.6	91.4	85.6
Hacked	43	36	7	0	83.7	100	91.1
Average					91.71	97.2	94.25

Table 6-6: Evaluation of Categorized Amharic Web Documents

Class labels	Categorized Web documents	Evaluation					
		TP	FP	FN	Precision (%)	Recall (%)	F- Measure (%)
ትምህርት	82	65	32	25	79.26	72.2	75.56
ዜና	151	148	3	11	98	93	95.43
ንግድ	53	50	3	8	94	86.2	89.93
ቱሪዝም	12	9	3	0	75	100	85.7
Average					86.56	87.85	86.65

The result shows that most English and Amharic Web documents are represented by news and educational contents.

The accuracy of English documents categorization is better than Amharic Web documents. This indicate that further work to be done on Web documents categorization that consider the nature of the language such as stop word removal, steaming and implementing vector space model or other machine learning approach for better accuracy.

Chapter Seven: Conclusion and Recommendations

7.1 Conclusion

The Web is the largest repository with diverse sources of information. Moreover, the contents of this information can obviously be in different languages. With the explosion of multi-lingual data on the Internet, the need and demand for an effective automated language identifier for the Webpage is further increased. Wikipedia, a rapid growing multilingual Web -based encyclopedia on the Internet, can serve as a measure of the multilingualism on the Internet. We can see that the number of Web pages and the kind of languages each document is presented has increased tremendously in recent years. In Ethiopia there is a rapid growth of Internet user each year. There are governmental and non-governmental organizations that provide hosting domain name service for customers.

The structure of the Web is increasingly being used to improve organization, search and analysis of information on the Web. Different organizations provide information through the Internet which helps for the society to access the information from anywhere. The amount of information on the local Web content is growing rapidly. Absent of tool to quantify qualitative and quantitative data found under certain domains are one of a challenge for policy makers to promote local Web content on the Internet. Although, there are governmental and non-governmental organization currently providing hosting domain name service for the society but there is no work so far that checks the status of each domain and the kind of language each domain presents.

This research work came up with the design and implementation of the local Web content observatory system under the .et domain. In order to address the objective stated in Chapter one it was necessary to examine the techniques and technologies for the development of local Web content observatory system under the .et domain. A comprehensive review of the state of the arts in the Web IR was undertaken. Open source tools providing different services were surveyed. An understanding of these techniques and technology allowed the formulation of requirements for each component of Local Web Content observatory system. Local Web content observatory

consists of crawler, statistical extractor, content extractor, language identifier, Web document categorizer and report generator components.

In developing the Local Web content observatory system under the .et domain it was found that the existing version of the Heritrix crawler used was insufficient to meet the requirement of the system; it was necessary to migrate to provide a JMX interface that allows the monitoring and management of the crawler. G2LI was adopted for Language identification of each crawled Web page. The language identifier is implemented by the well known n-gram algorithm which can handle HTML parsing and the ranking of the returned result. Because Web documents have special characteristics, the method was complemented with a set of heuristics, in order to better handle this information. To categorize Web content, Web document structure and frequent item sets were used as criteria.

Finally, in order to verify that the objective was met, evaluation of the Local Web Observatory System under the .et domain was undertaken. The evaluation took from the development of a demonstration application and system performance tests. The success of the demonstration and performance tests clearly shows the feasibility of providing a system for observing the Web content under the .et domain. To evaluate the performance of the crawler without a crash, the crawler completes the crawling operation which lasted for 2 days and 17 hours and collects around 263,031documents and downloaded all required document types. . To demonstrate the effectiveness of the local Web content categorizer precision, recall and F-measures test were conducted and average precision of 91.7%, recall of 97.2% and F-measures of 94.25% obtained for English document and precision of 91.7%, recall of 87.85% and F-measures of 86.65% obtained for Amharic document. The average accuracy rate of the statistical tracker is 98.72%.

The overall study has the following contributions:

- Identify appropriate crawling tool and modify it in order to make it fit to the requirement.
- Developed an algorithm that can identify the status of the Web content while crawling.
- Developed an algorithm to extract archived crawled Web contents and organize it in its respective domains.

- Identify appropriate language identifier tool and modified that tool to enhance it so that it can select the text content of a given file, ignore file type that does not provide hints for the language of a give file, process several files at a time in the process of language identification and identify the language of multilingual Web documents.
- Developed an algorithm that generates statistical data related to Web content language.
- Developed a system for Web content categorization by topic and by language.
- Developed a Web content observatory system that can identify that status and the language of Webs content under the .et domain.

7.2 Recommendations

In this research, we designed and developed a local Web content observatory system under the .et domain. In order to develop that a general local Web content observatory system under the .et domain and other domain that includes Ethiopic contents future research need to be done. Although the system had already demonstrated a good performance on a realistic setting, there is still room for further improvement. Future work may address the following in order to have a full- fledged Web content observatory system.

- A work on distributed Web crawler that minimizes the amount of time required to crawl billions of Websites.
- Categorize Web documents using machine learning approach.
- Integration of language identification in to Heritrix Web user interface to perform language identification while crawling.
- Include all domains hosted other than the .et domain.
- Integrating with local search engines.

References

[1] "The Relationship between Local Content, Internet Development and Access Prices, a Collaborative Research Report by ISOC", OECD and UNESCO, Geneva, 2012.

[2] Solomon Atnafu, "Promoting Local Internet Content: the Case of Ethiopia, A Report Submitted to ISOC Community Grant", June 30, 2013.

[3] Tessema Mindaye, "Design and Implementation of Amharic Search Engine", Unpublished Master Thesis, Department of Computer Science, Addis Ababa University, 2007.

[4] Hassen Redwan Hussen, "Enhanced Design of Amharic Search Engine", Unpublished Master Thesis, Department of Computer Science, Addis Ababa University, 2007.

[5] Tesfaye Guta Debela, "Afaan Oromo Search Engine", Unpublished Master Thesis, Department of Computer Science, Addis Ababa University, 2010.

[6] Hailay Beyene Berhe, "Design and Development of Tigrigna Search Engine", Unpublished Master Thesis, Department of Computer Science, Addis Ababa University, 2013.

[7] Miller, R. and Bharat, K. Rsphinx, "A Framework for Creating Personal Site-Specific Web Crawlers", Computer Network and ISDN System, V.30, pp.119-130, 1998.

[8] Website of ethio telecom, Available at https://www.ethionet.et, Accessed on September 8, 2014.

[9] Language Observatory Team, Nagaoka University of Technology, Language Observatory Web Crawling, 26-28 June, 2006.

[10] Y. Mikami, P. Zavarsky, and C.Campus, "The Language Observatory Project", in proceedings of WWW, May, 2005.

[11] P.Boldi, B Codenotti, M.Santini, and S.Vigna, "UbiCrawler: A Scalable Fully Distributed Web Crawler", Software: Practice & Experience, 34(8):711-726, 2004.

[12] Meikobyashi and Koichi Takeda, "Information Retrieval on the Web", IBM Research, 2004.

[13] W.B. Frakes and R. Baeza-Yates, eds., "Information Retrieval: Data Structures and Algorithms", Prentice Hall, Englewood Cliffs, N.J., 1992.

[14] E. Garfield, "Citation Analysis as a Tool in Journal Evaluation Science", 178(60):471–479, November, 1972.

[15] Gabriel Pinski and Francis Narin, ”Citation Influence for Journal Aggregates of Scientific Publications: Theory with Application to the Literature of Physics”, 12(5):297–312, 1976.

[16] Jon M. Kleinberg, “Authoritative Sources in a Hyperlinked Environment”, ACM, 46(5):604–632, 1999.

[17] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, “The Page Rank Citation Ranking: Bringing Order to the Web”, Technical Report 1999-66, Stanford Info Lab, November, 1999.

[18] Junghoo Cho and Hector Garcia-Molina, “Parallel Crawlers; In Proceedings of WWW”, Honolulu, Hawaii, USA, May 7-11, 2002.

[19] Sergey Brin and Lawrence Page, “The Anatomy of a Large-Scale Hyper Textual Web Search Engine”, In Proceedings of WWW Conference, 1998.

[20] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andrews Paepcke and Sriram Raghavan, ”Searching the Web”, In Proceedings ACM International Conference on Management of Data (SIGMOD) Conference, May, 2000.

[21] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang, ” Accessing the Deep Web Community”,ACM, 50(5):94–101, 2007.

[22] Marcus P.Zilman, “Deep Web Research and Discovery Resources”, 2014.

[23] Brian D. Davison, “Topical locality in the Web”, In SIGIR ’00: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 272–279, New York, NY, USA, 2000.

[24] P. M. E. De Bra and R. D. J, “Post Information Retrieval in the World-Wide Web: Making Client Based Searching Feasible”, Computer Networks and ISDN Systems, 27(2):183–192, 1994.

[25] Michael Hersovici, Michal Jacovi, Yoelle S. Maarek, Dan Pelleg, Menachem Shtalhaim, and Sigalit Ur, ”The Shark-Search Algorithm, An application: Tailored Web Site Mapping”, Computer Networks and ISDN Systems, 30(1-7):317–326, 1998.

[26] Sougata Mukherjea, ” WTMS: A System for Collecting and Analyzing Topic Specific Web Information”, Computer Networks, 33(1-6):457–471, 2000.

[27] Soumen Chakrabarti, Martin van den Berg, and Byron Dom, “Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery”, Computer Networks, 31(11-16):1623–1640, 1999.

[28] Open Directory Project, “The Largest Human-Edited Directory of the Web”, Available at http://www.dmoz.org/, accessed on June 10, 2014.

[29] Filippo Menczer and Richard K. Belew, “Adaptive Retrieval Agents: Internalizing Local Context and Scaling up to the Web, Machine Learning”, 39(2):203–242, May, 2000.

[30] Thomas Bayes, “An Essay Towards Solving a Problem in the Doctrine of Chances”, Philosophical Transactions of the Royal Society of London, 53:370–418, 1763.

[31] Jason D. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger, ” Tackling the Poor Assumptions of Naïve Bayes Text Classifiers”, In ICML, pages 616–623, AAAI Press, 2003.

[32] S. Kullback and R. A. Leibler, “On Information and Sufficiency”, Annals of Mathematical Statistics, 22:49–86, 1951.

[33] C. W. Gini, “Variability and Mutability, Contribution to the Study of Statistical Distributions and Relations”, Studi Economico-Giuridici della R. Universita de Cagliari, 1912. Reviewed in: Light, R. J., Margolin, B. H, “An Analysis of Variance for Categorical Data”, Journal of American Statistical Association, Vol. 66 pp. 534-544, 1971.

[34] S. Haykin, “Neural Networks: A Comprehensive Foundation. Macmillan”, New York, 1994.

[35] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers”, In COLT ’92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pp.144–152, New York, NY, USA, 1992.

[36] W.B. Frakes and R. Baeza-Yates, eds., “Information Retrieval: Data Structures and Algorithms”, Prentice Hall, Englewood Cliffs, N.J., 1992.

[37] G. Salton, “Automatic Text Processing”, Addison-Wesley, 1989.

[38] George K. Zipf, “Human Behavior and the Principle of Least-Effort”, Addison-Wesley, Cambridge MA, 1949.

[39] Julie B. Lovins, “Error Evaluation for Stemming Algorithms as Clustering Algorithms”, Journal of the American Society for Information Science, pp. 22(1):28–40, 1971.

[40] E. M. Keen and J. A. Digger, “Report of an Information Science Index Languages Test”, Aberystwyth College of Librarianship, Aberystwyth, Wales, 1972.

[41] Gerard Salton and C.S. Yang, “On the Specification of Term Values in Automatic Indexing”, Journal of Documentation, 29:351–372, 1973.

[42] C. J. van Rijsbergen, “Information Retrieval”, Butterworths, London, England, second edition, 1979.

[43] Jason D. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger, ” Tackling the Poor Assumptions of Naïve Bayes Text Classifiers”, In ICML, pp. 616–623. AAAI Press, 2003.

[44] S. E. Robertson and Karen Sparck Jones, “Relevance Weighting of Search Terms”, Journal of the American Society for Information Science, 27(3):129–146, 1976.

[45] Jay M. Ponte and W. Bruce Croft, “Language Modeling Approach to Information Retrieval”, In SIGIR ’98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval”, pp. 275–281, New York, NY, USA, 1998.

[46] Ulla-Alexandra Mattl, “Language Observatory Final Report”, EUNIC in Brussels, May 2012.

[47] The Poliglotti4.eu project, available at http://poliglotti4.eu/php/language-tools , Accessed on December 11, 2014.

[48] A.Mendez-Torreblanca and M. Monte, “A Trend Discovery for Dynamic Web Content Mining”, IEEE, Intelligence System, Vol. 14, pp. 20-22, 2002.

[49] Kshitija Pol, Nita Patil, Shreya Patankar, and Chhaya Das, “A Survey on Web Content Mining and Extraction of Structured and Semi structured Data”, First International Conference on Emerging Trends in Engineering and Technology, Vol. I, March 14-16, 2012.

[50] Katia Hayati, "Language Identification on the World Wide Web", Unpublished Master Thesis, University of California, June, 2004.

[51] Yew Choong Chew, Yoshiki Mikami, and Robin Lee Nagano, “Language Identification of Web Pages Based on Improved N-gram Algorithm”, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May, 2011.

[52] Boldi.P,Codenotti.B, and Santini.M, ”Ubicrawler: A Scalable Fully Distributed Web Crawler, Software: Practice and Experience”, Vol.34, NO.8, pp.711-726.

[53] Suzuki I., Mikami Y., and Ohsato A., “A Language and Character Set Determination Method Based on N-gram Statistics”, ACM Transactions on Asian Language Information Processing, Vol. 1. No. 3, pp. 270-279, 2002.

[54] Language Observatory, Available at: http://gii2.nagaokaut.ac.jp/gii/blog/lopdiary.php, Accessed on November 20, 2014.

[55] S. T. Nandasara1, Shigeaki Kodama, Chew Yew Choong, Rizza Caminero, Ahmed Tarcan, Hammam Riza, Robin Lee Nagano, and Yoshiki Mikami, "An Analysis of Asian Language Web Pages", The International Journal on Advances in ICT for Emerging Regions, Vol 1, pp. 33-41,2008.

[56] Heritrix, "The Internet Archive's Open-Source, Extensible, Web-scale, Archival-quality Web Crawler Project", Available at http://crawler.archive.org/, Accessed on September 10, 2014.

[57] Web Crawler, available at: http://en.wikipedia.org/wiki/Web_crawler, Last Accessed on September 8, 2014.

[58] Choong.C, and Mikami.Y, "Optimization of N-gram Based Language Identification for Web Documents", Unpublished Master Thesis, Nagaoka University of Technology, March, 2007.

[59] Methanol Toolkit, Fully Customizable Web crawling system, Available at http://metha-sys.org/, Accessed on September 10, 2014.

[60] Grub Toolkit, Distributed Web Crawling System, Available at: http://www.grub.org, Accessed on September 14, 2014.

[61] The Lemur Toolkit, Open-source Suite of tools Designed to Facilitate Research in Language Modeling and Information Retrieval, Available at http://www.lemurproject.org/, Accessed on September 10, 2014.

[62] Lucene, Java-based Indexing and Search technology, available at http://lucene.apache.org/, Accessed on September 15, 2014.

[63] Xapian, Toolkit which Allows Developers to easily add advanced indexing and search facilities to their own applications, available at http://xapian.org/, Accessed on September 10, 2010.

[64] Nutch, Open Source Web-search Software, available at http://lucene.apache.org/nutch/, Accessed on September 15, 2014.

[65] Gordon Mohr, Michael Stack, Igor Ranitovic, Dan Avery and Michele Kimpton, "An Introduction to Heritrix an Open Source Archival Quality Web Crawler", 4th international Web archiving workshop, 2004.

[66] Qingzhao Tan, "Designing New Crawling and Indexing Techniques for Web Search Engines", 2008.

[67] Rui Cai, Jiang- Ming Yang, Weilai Lai, Yida Wang, and Lei Zhang, "IRobot: An Intelligent Crawler for Web Forum", Microsoft Research, Asia, April 21-25, 2008.

[68] M. Diligenti, F.M. Coetzee, S.Lawrence, C.Giles and M.Gori, "Focused Crawler Using Context Graphs", NEC Research Institute, Princetone, USA, 2000.

[69] Focused Crawler, Available at: http://en.wikipedia.org/wiki/Focused_crawler, Accessed on September 20, 2014.

[70] Jose Manuel, Perez Ramirez, Luis Enrique, and Colmenares Guillen, "An Approach of Crawlers for Semantic Web Application", 2012.

[71] Pooja Mudgil, K. Sharma, and Pooja Gupta, "An Improved Indexing Mechanism to Index Web Documents", 5th International Conference on Computational Intelligence and Communication Networks, Indian, 2013.

[72] Vector Space Model, Available at: http://en.wikipedia.org/wiki/Vector_space_model, Accessed on September 20, 2014.

[73] Arual Parkash, Kranthi Kumar, "Web Page Categorization base on Document structure, Center for Visual Information Technology Institutional Institute of Information Technology", India, 2002.

[74] Olga Artemenko, Thomas Mandl, Margaryta Shramko, and Christa Womser, "Implementation and Evaluation of a Language Identification System for Mono- and Multi-lingual Texts", 2006.

[75] McNamee P., and Mayfield J., "Character N-Gram Tokenization for European Language Text Retrieval", 2004.

[76] Martins B., and Silva M., "Language Identification in Web Pages", Proc ACM SAC Symposium on Applied Computing, pp. 764-768, Santa Fe, New Mexico, USA, March 13.-17, 2005.

[77] Souter C., Churcher,G., Hayes J., Hughes J., and Johnson S., "Natural Language Identification Using Corpus-Based Models", 1994.

Appendix-1: New Crawling Job Configuration

```
 <?xml version="1.0" encoding="UTF-8"?><crawl-order
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="heritrix_settings.xsd">
  <meta>
    <name>ETDomain1</name>
    <description>ETDomain1</description>
    <operator>Admin</operator>
    <organization/>Addis Ababa University</organization>
    <audience/>
    <date>20140522103455</date>
  </meta>
  <controller>
    <string name="settings-directory">settings</string>
    <string name="disk-path"/>
    <string name="logs-path">logs</string>
    <string name="checkpoints-path">checkpoints</string>
    <string name="state-path">state</string>
    <string name="scratch-path">scratch</string>
    <long name="max-bytes-download">0</long>
    <long name="max-document-download">0</long>
    <long name="max-time-sec">0</long>
    <integer name="max-toe-threads">15</integer>
    <integer name="recorder-out-buffer-bytes">4096</integer>
    <integer name="recorder-in-buffer-bytes">65536</integer>
    <integer name="bdb-cache-percent">0</integer>
    <newObject name="scope"
class="org.archive.crawler.deciderules.DecidingScope">
      <boolean name="enabled">true</boolean>
      <string name="seedsfile">seeds.txt</string>
      <boolean name="reread-seeds-on-config">true</boolean>
      <newObject name="decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
        <map name="rules">
          <newObject name="rejectByDefault"
class="org.archive.crawler.deciderules.RejectDecideRule">
          </newObject>
          <newObject name="acceptIfSurtPrefixed"
class="org.archive.crawler.deciderules.SurtPrefixedDecideRule">
            <string name="decision">ACCEPT</string>
            <string name="surts-source-file"/>
            <boolean name="seeds-as-surt-prefixes">true</boolean>
            <string name="surts-dump-file"/>
            <boolean name="also-check-via">false</boolean>
            <boolean name="rebuild-on-reconfig">true</boolean>
          </newObject>
          <newObject name="rejectIfTooManyHops"
class="org.archive.crawler.deciderules.TooManyHopsDecideRule">
            <integer name="max-hops">20</integer>
```

```
          </newObject>
          <newObject name="acceptIfTranscluded"
class="org.archive.crawler.deciderules.TransclusionDecideRule">
            <integer name="max-trans-hops">3</integer>
            <integer name="max-speculative-hops">1</integer>
          </newObject>
          <newObject name="rejectIfPathological"
class="org.archive.crawler.deciderules.PathologicalPathDecideRule">
            <integer name="max-repetitions">2</integer>
          </newObject>
          <newObject name="rejectIfTooManyPathSegs"
class="org.archive.crawler.deciderules.TooManyPathSegmentsDecideRule">
            <integer name="max-path-depth">10</integer>
          </newObject>
          <newObject name="acceptIfPrerequisite"
class="org.archive.crawler.deciderules.PrerequisiteAcceptDecideRule">
          </newObject>
          </map>
          </newObject>
          </newObject>
          <map name="http-headers">
          <string name="user-agent">Mozilla/5.0 (compatible;
heritrix/1.14.1 +http://peterpuwang.googlepages.com )</string>
         <string name="from">arsgashaw1@gmail.com</string>
          </map>
        <newObject name="robots-honoring-policy"
class="org.archive.crawler.datamodel.RobotsHonoringPolicy">
        <string name="type">classic</string>
        <boolean name="masquerade">false</boolean>
        <text name="custom-robots"/>
        <stringList name="user-agents">
        <string>my-heritrix-crawler (+http://aau.edu.et)</string>
        </stringList>
        </newObject>
        <newObject name="frontier"
class="org.archive.crawler.frontier.BdbFrontier">
      <float name="delay-factor">4.0</float>
      <integer name="max-delay-ms">20000</integer>
      <integer name="min-delay-ms">2000</integer>
      <integer name="respect-crawl-delay-up-to-secs">300</integer>
      <integer name="max-retries">30</integer>
      <long name="retry-delay-seconds">900</long>
      <integer name="preference-embed-hops">1</integer>
      <integer name="total-bandwidth-usage-KB-sec">0</integer>
      <integer name="max-per-host-bandwidth-usage-KB-sec">0</integer>
      <string name="queue-assignment-
policy">org.archive.crawler.frontier.HostnameQueueAssignmentPolicy</st
ring>
      <string name="force-queue-assignment"/>
      <boolean name="pause-at-start">false</boolean>
      <boolean name="pause-at-finish">true</boolean>
      <boolean name="source-tag-seeds">false</boolean>
```

```
      <boolean name="recovery-log-enabled">true</boolean>
      <boolean name="hold-queues">true</boolean>
      <integer name="balance-replenish-amount">3000</integer>
      <integer name="error-penalty-amount">100</integer>
      <long name="queue-total-budget">-1</long>
      <string name="cost-
policy">org.archive.crawler.frontier.ZeroCostAssignmentPolicy</string>
      <long name="snooze-deactivate-ms">300000</long>
      <integer name="target-ready-backlog">50</integer>
      <string name="uri-included-
structure">org.archive.crawler.util.BdbUriUniqFilter</string>
      <boolean name="dump-pending-at-close">false</boolean>
    </newObject>
    <map name="uri-canonicalization-rules">
      <newObject name="Lowercase"
class="org.archive.crawler.URLs.canonicalize.LowercaseRule">
        <boolean name="enabled">true</boolean>
      </newObject>
      <newObject name="Userinfo"
class="org.archive.crawler.URLs.canonicalize.StripUserinfoRule">
        <boolean name="enabled">true</boolean>
      </newObject>
      <newObject name="WWW[0-9]*"
class="org.archive.crawler.URLs.canonicalize.StripWWWNRule">
        <boolean name="enabled">true</boolean>
      </newObject>
      <newObject name="SessionIDs"
class="org.archive.crawler.URLs.canonicalize.StripSessionIDs">
        <boolean name="enabled">true</boolean>
      </newObject>
      <newObject name="SessionCFIDs"
class="org.archive.crawler.URLs.canonicalize.StripSessionCFIDs">
        <boolean name="enabled">true</boolean>
      </newObject>
      <newObject name="QueryStrPrefix"
class="org.archive.crawler.URLs.canonicalize.FixupQueryStr">
        <boolean name="enabled">true</boolean>
      </newObject>
    </map>
    <map name="pre-fetch-processors">
      <newObject name="Preselector"
class="org.archive.crawler.prefetch.Preselector">
        <boolean name="enabled">true</boolean>
        <newObject name="Preselector#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <boolean name="override-logger">false</boolean>
        <boolean name="recheck-scope">true</boolean>
        <boolean name="block-all">false</boolean>
        <string name="block-by-regexp"/>
```

```
        <string name="allow-by-regexp"/>
      </newObject>
      <newObject name="Preprocessor"
class="org.archive.crawler.prefetch.PreconditionEnforcer">
        <boolean name="enabled">true</boolean>
        <newObject name="Preprocessor#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <integer name="ip-validity-duration-seconds">21600</integer>
        <integer name="robot-validity-duration-
seconds">86400</integer>
        <boolean name="calculate-robots-only">false</boolean>
      </newObject>
    </map>
    <map name="fetch-processors">
      <newObject name="DNS"
class="org.archive.crawler.fetcher.FetchDNS">
        <boolean name="enabled">true</boolean>
        <newObject name="DNS#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <boolean name="accept-non-dns-resolves">false</boolean>
        <boolean name="digest-content">true</boolean>
        <string name="digest-algorithm">sha1</string>
      </newObject>
      <newObject name="HTTP"
class="org.archive.crawler.fetcher.FetchHTTP">
        <boolean name="enabled">true</boolean>
        <newObject name="HTTP#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <newObject name="midfetch-decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <integer name="timeout-seconds">1200</integer>
        <integer name="sotimeout-ms">20000</integer>
        <integer name="fetch-bandwidth">0</integer>
        <long name="max-length-bytes">0</long>
        <boolean name="ignore-cookies">false</boolean>
        <boolean name="use-bdb-for-cookies">true</boolean>
        <string name="load-cookies-from-file"/>
        <string name="save-cookies-to-file"/>
        <string name="trust-level">open</string>
        <stringList name="accept-headers">
```

```
        </stringList>
        <string name="http-proxy-host"/>
        <string name="http-proxy-port"/>
        <string name="default-encoding">ISO-8859-1</string>
        <boolean name="digest-content">true</boolean>
        <string name="digest-algorithm">sha1</string>
        <boolean name="send-if-modified-since">true</boolean>
        <boolean name="send-if-none-match">true</boolean>
        <boolean name="send-connection-close">true</boolean>
        <boolean name="send-referer">true</boolean>
        <boolean name="send-range">false</boolean>
        <string name="http-bind-address"/>
      </newObject>
    </map>
    <map name="extract-processors">
      <newObject name="ExtractorHTTP"
class="org.archive.crawler.extractor.ExtractorHTTP">
        <boolean name="enabled">true</boolean>
        <newObject name="ExtractorHTTP#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
      <newObject name="ExtractorHTML"
class="org.archive.crawler.extractor.ExtractorHTML">
        <boolean name="enabled">true</boolean>
        <newObject name="ExtractorHTML#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <boolean name="extract-Javascript">true</boolean>
        <boolean name="treat-frames-as-embed-links">true</boolean>
        <boolean name="ignore-form-action-URLs">false</boolean>
        <boolean name="extract-only-form-gets">true</boolean>
        <boolean name="extract-value-attributes">true</boolean>
        <boolean name="ignore-unexpected-html">true</boolean>
      </newObject>
      <newObject name="ExtractorCSS"
class="org.archive.crawler.extractor.ExtractorCSS">
        <boolean name="enabled">true</boolean>
        <newObject name="ExtractorCSS#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
      <newObject name="ExtractorJS"
class="org.archive.crawler.extractor.ExtractorJS">
        <boolean name="enabled">true</boolean>
```

```
        <newObject name="ExtractorJS#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
      <newObject name="ExtractorSWF"
class="org.archive.crawler.extractor.ExtractorSWF">
        <boolean name="enabled">true</boolean>
        <newObject name="ExtractorSWF#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
    </map>
    <map name="write-processors">
      <newObject name="Archiver"
class="org.archive.crawler.writer.ARCWriterProcessor">
        <boolean name="enabled">true</boolean>
        <newObject name="Archiver#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <boolean name="compress">true</boolean>
        <string name="prefix">ETD</string>
        <string name="suffix">${HOSTNAME}</string>
        <long name="max-size-bytes">100000000</long>
        <stringList name="path">
          <string>arcs</string>
        </stringList>
        <integer name="pool-max-active">5</integer>
        <integer name="pool-max-wait">300000</integer>
        <long name="total-bytes-to-write">0</long>
        <boolean name="skip-identical-digests">false</boolean>
      </newObject>
    </map>
    <map name="post-processors">
      <newObject name="Updater"
class="org.archive.crawler.postprocessor.CrawlStateUpdater">
        <boolean name="enabled">true</boolean>
        <newObject name="Updater#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
      <newObject name="LinksScoper"
class="org.archive.crawler.postprocessor.LinksScoper">
        <boolean name="enabled">true</boolean>
```

```
        <newObject name="LinksScoper#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
        <boolean name="override-logger">false</boolean>
        <boolean name="seed-redirects-new-seed">true</boolean>
        <integer name="preference-depth-hops">-1</integer>
        <newObject name="scope-rejected-URLs-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
      <newObject name="Scheduler"
class="org.archive.crawler.postprocessor.FrontierScheduler">
        <boolean name="enabled">true</boolean>
        <newObject name="Scheduler#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
          <map name="rules">
          </map>
        </newObject>
      </newObject>
    </map>
    <map name="loggers">
      <newObject name="crawl-statistics"
class="org.archive.crawler.admin.StatisticsTracker">
        <integer name="interval-seconds">20</integer>
      </newObject>
    </map>
    <string name="recover-path"/>
    <boolean name="checkpoint-copy-bdbje-logs">true</boolean>
    <boolean name="recover-retain-failures">false</boolean>
    <boolean name="recover-scope-includes">true</boolean>
    <boolean name="recover-scope-enqueues">true</boolean>
    <newObject name="credential-store"
class="org.archive.crawler.datamodel.CredentialStore">
      <map name="credentials">
      </map>
    </newObject>
  </controller>
</crawl-order>
```

Appendix-2: Source Code for Web Document Language Detection

```
public String identifyFile(String teacherDir, String identifyMethod, Boolean htmlFilter, int noOfResult,
boolean writeFile, String inFilename)
   throws Exception
  { if (!Validate.isGoodFile(inFilename)) //this is for file not directory cheek for directory
   if (!Validate.isGoodDir(inFilename)) {    setLiTarget("FILE");
return getNullResultStr();
} if (getLiTarget() == null){
String shortFilename = inFilename;
int fsIdx = inFilename.lastIndexOf(
System.getProperty("file.separator"));
     if (fsIdx > 0) {
     shortFilename = inFilename.substring(fsIdx + 1);     }
  setLiTarget(URLSEncoder.encode(shortFilename, "utf-8"));
   }
BufferedInputStream fs = null;
 byte[] targetBuffer = (byte[])null;
   try     {
   System.out.println("List of file to identify from selected directory" + "\n");
    File dir = new File(inFilename);
       List<File> files = RecursiveDirListing.getDirListing(dir);
       String filename = null;
    for (File fileToIdentify : files) {
     if (!fileToIdentify.isDirectory()) {
     if (!fileToIdentify.getPath().contains("CVS")) {
      if (!fileToIdentify.getPath().contains(".zip"))
```

```
{   if (!fileToIdentify.getPath().contains(".rar")){   if (!fileToIdentify.getPath().contains(".z"))
{    if (!fileToIdentify.getPath().contains(".mp4")) {    if (!fileToIdentify.getPath().contains(".flv"))
{   if (!fileToIdentify.getPath().contains(".mp3")) {  if (!fileToIdentify.getPath().contains(".wmv"))
 { if (!fileToIdentify.getPath().contains(".js")) {   if (!fileToIdentify.getPath().contains(".swf"))
{if (!fileToIdentify.getPath().contains(".css")) if (!fileToIdentify.getPath().contains(".png"))
{ if (!fileToIdentify.getPath().contains(".gif")){ if (!fileToIdentify.getPath().contains(".bmp"))
{ if (!fileToIdentify.getPath().contains(".jpeg")) {   if (!fileToIdentify.getPath().contains(".tif"))
{   if (!fileToIdentify.getPath().contains(".jpg")) {     if (!fileToIdentify.getPath().contains(".ico"))
{    if (!fileToIdentify.getPath().contains(".mpeg")) {    if (!fileToIdentify.getPath().contains(".asf"))
{
       filename = fileToIdentify.getAbsolutePath();
setLiTarget(filename);
      List<File> files = RecursiveDirListing.getDirListing(dir);
File file = new File(filename);
fs = new BufferedInputStream(new FileInputStream(file));
long fileSize = file.length(); if (fileSize > 21474836457L) {
throw new Exception("File size > 2147483647");}
int length = (int)fileSize;
  targetBuffer = new byte[length];
    fs.read(targetBuffer, 0, length);    }}}}}}}}}}}}}}}}}}}}
System.out.println(identifyByte(targetBuffer, teacherDir, identifyMethod, htmlFilter.booleanValue(),
noOfResult, writeFile));
    }}}}    catch (IOException e){   e.printStackTrace();   }
  Finally    {
   IoUtil.close(fs);
   }   return ("Identification completed"); }
```

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:
Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____

Place and date of submission: Addis Ababa, March 2015.