A Tool For Threading, Organizing And Presenting Emails Using A Web Interface.

by

Apurva Aravindakshan Nair

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved May 2012 by the
Graduate Supervisory Committee:

Hasan Davulcu, Chair
Partha Dasgupta
Arunabha Sen

ARIZONA STATE UNIVERSITY

August 2012

ABSTRACT

The overall contribution of the Minerva Initiative at ASU is to map social organizations in a multidimensional space that provides a measure of their radical or counter radical influence over the demographics of a nation. This tool serves as a simple content management system to store and track project resources like documents, images, videos and web links. It provides centralized and secure access to email conversations among project team members. Conversations are categorized into one of the seven pre-defined categories. Each category is associated with a certain set of keywords and we follow a frequency based approach for matching email conversations with the categories. The interface is hosted as a web application which can be accessed by the project team.

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude and appreciation to everyone who helped make this master's thesis possible. I sincerely thank my advisor, Dr. Hasan Davulcu, for his continued guidance, support and encouragement during my masters and while writing this thesis. My sincere thanks to Dr. Arunabha Sen, Dr. Partha Dasgupta and Dr. Johoung Lee for being on my thesis supervisory committee and for providing the guidance and the feedback on my work. I would also like to specially thank Shreejay Nair for his ideas, encouragement and support and valuable feedback that helped me make this thesis more structured and comprehensive. Many thanks to all my lab-mates for making this journey so exciting and asking this a great learning experience. I am very grateful for the love and the unconditional support of my family and friends.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Minerva Initiative[1] focuses on understanding the social, political and cultural forces that influence different regions of the world. The research targets seven important areas: Strategic Impact of Religious and Cultural Changes, Terrorism and Terrorist Ideologies, Science, Technology and Military Transformations in China and Developing States, National Security Implications of Energy and Environmental Stress, New Theories of Cross Domain Deterrence, Regime and Social Dynamics in Failed, Failing and Fragile Authoritarian States and New Approaches to understanding dimensions of national security, conflict and cooperation.

Muslim extremism[2] is one of the most critical issues that the global community faces and its exclusivist and extremist interpretations of Islam pose a threat to Muslim as well as non-Muslim communities. The project focuses on the radical and counter radical movements in three distinct regions : Southeast Asia, West Africa and Western Europe with a view to analyze the contribution of religion in the rise of “new colonialism”(quote) instead of economical factors.

The overall contribution of the project at ASU aims at creating a multidimensional view of social organizations on the scale of how influential their radical or counter radical ideologies are in the Muslim world. The project also aims at tracking the shifts of these organizations over a period of time.

---

[1] The Minerva Initiative http://minerva.dtic.mil/funded.html

[2] http://csrc.asu.edu/research/projects/finding-allies-mapping-counter-radical-muslim-discourse

1.1 Motivation

The project team communicates on a daily basis via group emails to exchange their opinions about the on-going research. This includes sharing resources related to the organizations prevalent in the areas under consideration that will help us towards the overall goal of mapping organizations on the radical-counter radical scale.

Currently these conversations are present in the email formats without any proper form of organization. Minerva Wiki presents a secure environment where these emails can be presented in an organized and convenient manner to the project team. Team members can access email content, videos, images, articles, web links and other such resources easily using the web interface. This tool provides a centralized content management system that can help the team document the progress of the project, discuss and finalize milestones and also keep track of valuable resources that can prove vital to the overall research.

1.2 Thesis Outline

The project team communicates on a daily basis via group emails to exchange their opinions about the on-going research. This includes sharing resources related to the organizations prevalent in the areas under consideration that will help us towards the overall goal of mapping organizations on the radical-counter radical scale.

Currently these conversations are present in the email formats without any proper form of organization. Minerva Wiki presents a secure environment where these emails can be presented in an organized and convenient manner to the project team. Team members can access email content, videos, images, articles, web links and other such resources easily using the web interface. This tool

provides a centralized content management system that can help the team document the progress of the project, discuss and finalize milestones and also keep track of valuable resources that can prove vital to the overall research.

Chapter 2

SYSTEM OVERVIEW

The overall contribution of the tool is to provise a secure and centraized content management system. It is used to store and keep track of email conversations taking place amongst the project team members. It also provides a way to organize resources such as images, videos, documents, web links and share important information pertaining to the on going research.

2.1 Tool Functionality

The tool is implemented as a web application with the standard three layers of implementation i.e the server side, the database layer and the front end interface. It provides secure access to team conversations and resources pertaining to the research. An authorized user can log in to the system, view conversations and download documents. It provides a system using which the team can keep track of the research, upcoming and already achieved milestones and ensure that resources are properly organized and available to access at all times.

2.2 Implementation and Hosting Details

The project is hosted as a web application on the XAMPP-Tomcat server and has the back end, the data layer and the front end as its three major components.

The back end of the project is base on an algorithm for parsing and extracting information from the given dataset. We have used Java to implement the extraction algorithm. The data layer is implemented using the MySql service provided by XAMPP-phpmyadmin. The front end is designed using JSP with use of jquery components for data display and visual enhancements

Chapter 3

CLASSIFICATION OF DATA

The main objective of the system is to implement an algorithm that can classify the given dataset into appropriate categories. Every category is recognized by a list of keywords. The algorithm is based on a simple frequency based matching logic to associate the data with the categories.

3.1 Data Source

The content on the tool is mainly from email conversations. Emails conversations are of the type ".msg" (Microsoft Outlook). They can be either single emails or an email conversation thread with multiple senders and multiple content blocks. Some of the emails also have attachments. These attachments can be images, videos, web- links or documents. We will require the subject, sender name, the email body and the date the email was sent. Once these contents have been retrieved they are stored in the database within a table corresponding to the category of the email.

The source emails are mainly of two types:

1. Single emails
2. Email conversations

3.1.1 Individual Emails

These emails have a single body of text and we can directly use the msgparser Java library functions to get the required information.

3.1.2 Email Conversations

These emails are generally conversations among team members. They can be viewed as multiple single emails embedded into one. In this case, using the message parser library we can find out the details of only the most recent

conversation in the email. To find details of older conversations, we parse through the email content and extract field information. Also in case of an email conversation the original date would be the date the first email was sent. This requires us to display the extracted content in the reverse order in order to follow the chronological sequence.

The algorithm for extracting dates and content has been discussed in the following sections.

3.2 Introduction to categories and keywords

Minerva wiki focuses on seven distinct categories. Each category is associated with its own set of keywords. The categories and keywords are listed below:

1. Music – music, song, rap, rapper, hip hop.
2. Muslim Youth – youth
3. Women – women, gender, girl, wife, mother, daughter
4. Sufism/Salafism – sufi, salafi, wahhabi, Sufism, salafism, Wahhabism
5. Sin in Public Places – sin, haram, Bid'ah, shirk
6. Secularism and Muslim Identity – secularism, Laicite, theocracy, identity, Citizen, citizenship, Immigrants, immigration, nation, society, legal, European muslim, French Muslim, British muslim, German muslim, islamophobia.
7. Quran Exergesis – qur'an, quran, exergesis, verse,ayah.

3.3 Classification logic

The algorithm follows a frequency based categorization of the email content. It calculates the frequency of all the above mentioned keywords within the email. The email is classified into the category with maximum keyword frequency in the email content. If the email does not belong to any category, it is simply stored in

the uncategorized folder. If it belongs to any particular category, we parse through the email body to get the contents.

The algorithm for parsing the content is discussed in the next section.

Categories	Keywords
Music	music, song, rap, rapper, hip hop.
Muslim Youth	Youth
Women	women, gender, girl, wife, mother, daughter
Sufi/Salafi	sufi, salafi, wahhabi, Sufism, salafism, Wahhabism
Sin in Public Places	sin, haram, Bid'ah, shirk
Secularim and Muslim Identity	secularism, Laicite, theocracy, identity, Citizen, citizenship, Immigrants, immigration, nation, society, legal, European muslim, French Muslim, British muslim, German muslim, islamophobia.
Quran Exergesis	qur'an, quran, exergesis, verse,ayah

Table 3.1 Categories and there corresponding keywords

Chapter 4

SYSTEM DATA LAYER

The processed information is stored in a MySql database provided my Xampp-phpmyadmin. Information pertaining to each email is stored in a table corresponding to the category to which the email belongs.

4.1 Database Schema

The database contains seven tables corresponding to each one of the seven categories. The table names are as follows:

1. music
2. muslimyouth
3. sufi
4. sipp (Sin in public places)
5. quran
6. women
7. secular

The table structure is as shown below:

Column Name	**DataType**
Date	Timestamp
Subject	varchar
Body	varchar
From	varchar
Attachments	varchar
Replies	integer

Table 4.1 DatabaseTable Structure

Column Descriptions:

1. Date: A Timestamp field containing the date and time of the email. For single email it is the original date of the email. For email conversations it stores the date of the oldest conversation.
2. Subject: Contains the subject of the email
3. Body: Stores the formatted body content of the email.
4. From: Stores the original sender name for the email.
5. Attachments: Stores a single string containing formed by concatenation of multiple absolute paths. These paths correspond to the email attachments.
6. Replies: Stores the number of emails. It is zero for a single email and the number of individual emails for email conversations.

4.2 Inserting and Retrieving Data

The front end and the back end interface with the database using jdbc-mysql connectors. The subject field in the tables is indirectly used as a primary key while querying the tables to retrieve data and display it on the web interface. A sample insert and select query is given below:

We specifically make use of the PreparedStatment class provided by the java.sql library while providing the query parameters. This helps us to avoid any errors caused due to escape characters present in the any of the field data.

Chapter 5

SYSTEM BACK END

5.1 Data Parsing

The source emails are of the type ".msg". We are interested in the following components of the email:

1. The date the email was sent.
2. The name and email address of the sender.
3. The subject of the email.
4. The email body.
5. Number of emails (in case of email conversations)
6. List of attachments if any.

We use the msgparser Java library to parse the contents of the email. The library allows us to view an email as a Message object and provides different member functions to get details such as subject, body, sender and recipients names. It also provides functionality for separating attachments from the email body and storing it separately. Some of the member functions that we have used in the project are listed below:

1. getDate() : Returns the date of email as per header information
2. getSubject() : Returns the subject of the email
3. getFromName() :Returns the name of the sender
4. getBody() : Returns the body content of the email
5. getAttachments(); Returns an array containing attachments present in the email.

5.1.1 Accessing the dataset

We use the msgparser library to read the email files which are of type ".msg".

Firstly we create a message entity as shown below that will refer to the email file we need to parse.

Message msg= new Message ("mail.msg")

Once the entity is created we can use functions from the msgparser library to extract the following details:

5.1.2 Field Selection

In the given dataset, specific labels denote the email subject, the sender name and email specific address and the date of the email. We make use of these labels to find the position of the corresponding fields and extract the label values.

The following four sets of labels are present in the sample dataset:

1. Set 1

   From:

   Sent:

   To:

   Subject:

   [Email Body]

2. Set 2

   From:

   To:

   Subject:

   Date:

   [Email Body]

3. Set 3

   Sujet:

   Date:

   De:

   Pour:

   [Email Body]

4. Set 4

   Von:

   Gesendet:

   An:

   Betreff:

   [Email Body]

5.2 Data Extraction

The algorithm can be logically split into two stages

1. Date and Attachments extraction.
2. Content extraction and saving values to the database.

5.2.1 Extraction of Dates

In case of a single email, the msgparser library allows us to directly obtain the date of the email, using the function message.getDate().

For an email thread, we will need to parse through the contents of the email and get positions of fields that hold the email date. Dates for individual emails inside the conversation are associated with certain labels that can be used to locate the date fields. In the given sample set, the following labels or phrases are used to denote the date of the email:

1. Sent:
2. Date:
3. Gesendet:
4. On <date> , <senderName> wrote

The algorithm handles each label separately to extract the date. The steps to locate the date in the email content are enumerated below:

1. Create message entity msg by calling the Message constructor.
2. Create the StringBuffer bodyBuffer to store the body of the email.
3. Create the map dates with the subject of the email as the key and a list of dates as the value. The date list will hold all the date values present in the email. In case of a single email, it will hold just one value that is the original date of the email.
4. Get the original date of the email using the function msg.getDate().
5. Get index values for the following labels, using the function bodyBuffer.indexOf().
    a. Sent:
    b. Date:
    c. Gesendet:
6. Calculate the start index for the pattern On <date> <sender> wrote. The function for returning the start index of the phrase has been discussed below.
7. Calculate the minimum of the four values obtained in steps 3 and 4. This index gives us the location of the next date value in the email content. The value is stored as <variable name>

8. If the startIndex < -1 ,the email is a thread email and contains more than one conversation.
9. If the startIndex is index of labels Sent, Date or Gesendet:
    a. Store the content of bodyBuffer from the start index till the first newline character is encountered. This gives the date of the email.
10. If the startIndex is the index of the pattern “On <date> <sender> wrote”
    a. Find the index of the of the term “wrote”. The end index of the entire phrase will be indexOf(wrote)+6.
    b. Copy the content of bodyBuffer from startIndex to endIndex. This gives us the entire date line.
    c. Obtain the date of the email by splitting the date line and then obtaining the terms which define the date. (For eg: Consider date line is as follows: On 20th July 2010, Jane Doe wrote. The algorithm will split the dateline into 6 parts. We then merge the parts 20th, July and 2010 to form the date for the element.)
11. Add date value to the date list.
12. Calculate the new value for startIndex using the same step as mentioned in step 6.
13. While startIndex is not equal to -1 and the end of body is not reached, repeat steps 7-10.
14. Put the datelist as a value in the dates map, with the email subject as the key.
15. Repeat steps 1 to 14 for all the email files.

The above algorithm creates a map containing titles and dates for all the emails present in the dataset.

This map can be used to access the email body and split the content corresponding to each date value present in the date list.

Before the content of the email can be extracted, we check if the email belongs to any particular category. Only if it belongs to one of the seven categories, the content will be parsed.

5.2.2 Email Attachments

In addition to extracting date, this stage of the algorithm also uses the msgparser library to separate email attachments from the email and save them on the server. The absolute path for the attachment is stored in the attachmentlist. This list will be used to store the attachment paths in the database. These attachments can be then accessed through the wiki interface.

5.2.3 Content Extraction Algorithm

The content extraction portion of the algorithm aims at extracting the following data values from the email content:

1. The date the email was posted.
2. Sender details
3. Body of the email

The dates for each email is stored in the dateList as discussed above. The algorithm extracts the body content corresponding to each date in the list. This ensures that the email body is parsed accurately and every date in the list (in case of threads) corresponds to the right content.

Before the content extraction is performed, the category of the email is determined. Only if the email belongs to any one of the above mentioned categories, the content is extracted.

This stage of the algorithm also uses the attachments list to build an attachment string. This string will basically contain all elements of the attachment list. It will be stored in the database. The client side logic will focus on splitting this unified string into individual attachment names to be displayed on the interface.

Input to the algorithm is a single string containing the email body, subject, sender name, recipient name and the list of attachments.

The algorithm uses labels to distinguish one email from another. These labels are shown in the table below.

Start Label	Date Label	End Label
From	Sent	Subject
From	Date	Date
Sujet	Date	Pour
Von	Gesendet	Betreff

Table 5.1 Possible label combinations used for splitting of content

The content extraction algorithm is as follows:

1. Initialize string variable attachmentString to empty string.
2. If attachment list of email is not empty.
    a. For each element currElem in the attachment list
        i. attachmentString+= “EMAILATTACH”+currElem
3. If the dateList has single element (single email)
    a. Split the input string to get subject, from and body of the email.
    b. Save date, subject, body, from, size of datelist and attachmentString to the database.

4. If the dateList has multiple element (email thread)
    a. Split the input string to get values of the email subject and the sender and the body content of the email.
    b. Split the body content of the email into individual emails with the help of the sortThread function. It takes as input: the subject, the date, the sender details of the most recent email in the thread and returns a string value containing the subject, the actual start date of the conversation and the body content.
    c. Split the result string to obtain the respective values.
    d. Save the email date, subject. Body, original sender and attachments if any in the database.

5.2.4 Function sortThread

The sortThread function is majorly used to split up an email conversation into individual emails. It basically makes use of label data present in the email content to find boundaries for each email present in the conversation.

It takes as input the email subject, the email date corresponding the to latest email in the thread, a list of dates present in the email and the sender name of the latest email.

The function associates each date present in the date list with a single block of email data present in the email. This date-content pair is stored in a hashmap which allows us to re-construct the email in the reverse order i.e: from the oldest to the latest email and store the content in the database.

The algorithm used in the function is as follows:

1. Store the first element of the date list as the startDate of the email.
2. Find the index of the labels: “From”, On index, “Von”,”Sujet”. These labels indicate the beginning of emails in the thread. We use a pattern matcher to find index of the pattern “On <date> <sender> wrote” that finds the On index.
3. Find the minimum of the above mentioned index values and assign it to the variable cutIndex. The logic ensures that values of -1 will be handled appropriately.
4. Initialize the integer variable startIndex to 0 and string variable content to “Posted By: ”+senderName+” on”+” startDate”.
5. If the cutIndex is not -1.
    a. Add content of the body from startIndex to cutIndex to content.
    b. Delete the contents of the body from the startIndex till the cutIndex.
6. Else add the entire body string to content.
7. Skip to the next element in the date list.
8. Repeat steps 2 and 3 to find new value of startIndex.
9. Initialize StringBuffer fromName to “”. This buffer will hold the sender details of that particular email. Initialize string variable currBody to empty string.
10. If the startIndex is the index of ‘From’
    a. Find the index of the labels “Subject”, “Date”,”To” and “Sent” and assign the maximum value to variable endIndex. Negative values are appropriately handled.
    b. The endIndex basically gives the position of the next label. Using both the startIndex and the endIndex we can find the sender name and email address of that particular email.

c. Copy contents of body from startIndex to endIndex to fromName.

11. If startIndex is the index of ‘Von’
    a. Find index of the label “Gesendet” and assign it to endIndex.
    b. Copy contents of body from startIndex to endIndex to fromName
12. If startIndex is the index of ‘Sujet’
    a. Find index of label “De:” and assign it to endIndex.
    b. Copy contents of body from startIndex to endIndex to fromName.
13. If startIndex is the On index
    a. Initialize StringBuffer dateline. The dateline buffer will hold the entire line of content belonging to the “On <date> <sender name and email> wrote”.
    b. Find the index of the pattern “wrote”.
    c. Copy contents of body from the startIndex till the index of the pattern “wrote” to dateline.
    d. Remove any whitespace characters except for a space from the dateline and split the deadline using space as the delimiter.
    e. We can obtain the sender name by concatenating appropriate tokens generated by splitting the dateline.
14. If the startIndex is the index of “From”
    a. Find index values for the terms : “Subject”, “From” and “Date”.
    b. Find the maximum of these index values and assign it to variable cutIndex. Negative values are handled appropriately. Also since the label “Date” appears in different orders in different label sets, we also ensure that the index calculated is within the current label set.
    c. The cutIndex indicates the end of the label set and the beginning of the body content. Delete the contents of body till the cutIndex.

15. If the startIndex is the On Index
    a. Starting from the OnIndex delete contents of the body till a newline character is encountered.
    b. Assign the new OnIndex value to cutIndex.
16. If the startIndex is the index of ‘Von’
    a. Find the index of the term “Betreff” and assign it to cutIndex.
    b. Delete contents of the body till the cutIndex.
17. If the startIndex is the index of ‘Sujet’
    a. Find the index of the term “Pour” and assign it to cutIndex
    b. Delete the contents of the body till the cutIndex.
18. To find the end of the email body we find the index of the beginning of the next label set. Repeat steps 2 and 3 to get the value of endIndex.
19. Assign the value currBody+ the substring of the body from cutIndex till endIndex to the currBody variable.
20. Put the date and the contents of the currBody variable in the contentMap hash table.
21. Repeat steps 8 to 20 till there are elements in the datelist.
22. Reverse the datelist so that the order of dates is from the oldest to the latest email.
23. Get the value from contentMap corresponding to each element in the datelist.
24. Append each content value to variable finalStr and separate the content using the term “STARTEMAILFROM”
25. Return finalStr.

5.3 Backend Overview

The back end system logic basically makes use of text extraction techniques to thread email conversations. It creates a convenient and easy to read format of the email content which can then be stored in the database. The front end interface will just have to read the data as it is from the database and display appropriately. Thus the back end is designed to generate easily presentable information which considerably reduces the amount of processing to be done at the client side, thus making the interface less complicated, faster and easy to use.

Chapter 6

WEB INTERFACE

The Minerva Wiki tool is presented to the user in the form of a web application. It provides a secure access to the project team to view email conversations. It interfaces with the data layer to retrieve information from the tables and display them for the user.

6.1 Interface Components

The basic page is designed using JSP. To display the data to the user we have used the Datatables plugin provided by jQuery. Datatables allow us to design a simple, efficient and easy to use view to display information. It is equipped with in-built sorting and searching features that make the interface extremely user friendly and guarantees maximum utilization of the main functionality of the application.

In addition to data tables, we have used jQuery tabs to enhance the web interface. Each tab represents a category and the tab content is the data table showing information related to that particular category.

The front end consists of the following pages:

1. The index page
2. The tool homepage
3. The email content page
4. The attachments page

6.2 Information Display

The interface connects to the database through the mysql-jdbc connector. The tables are queried using the subject line as the search key. Information is retrieved and filtered into seven different tables depending upon the category of

each email. The subject line is linked to the email content page which gets the subject and the category as request parameters. The email content is then retrieved from the database using the category name and the subject line as search parameters

6.2.1 Index Page

The index page is the first page displayed to the user. It displays a brief introduction about the application. It also displays a login panel. Only a project team member with a valid username and password can access the system.

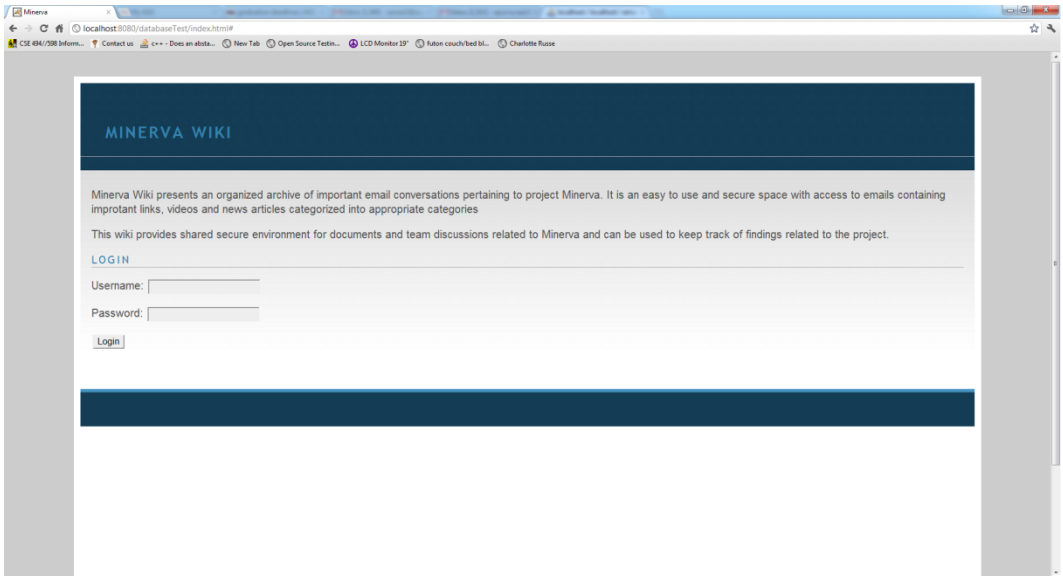

Figure 6.1 Screenshot of the Index.html page

6.2.2 Tool Homepage Display

It is the main interface of the application which displays the category tabs and the datatable information to the users. The jsp connects to the database to retrieve contents from the category tables. The data is organized in datatables and each datatable is displayed under the tab associated with its category. Every datatable has the following six columns:

1. Date: It displays the timestamp of the email. The date is of the format yyy-mm-dd hh:mm:ss.
2. Subject: It displays the subject of the email. The subject line is linked to the actual content of the email. When an user clicks on a subject link, the subject of the email and the name of the tab it is in is sent to the emailBody jsp. The subject is used to query the database to retrieve the email contents.
3. Replies: This column denotes the number of emails present within any email. If it is a single email it will have zero replies. In case of a thread it will display the number of individual emails present in the thread.
4. The datatable provides an option of sorting any column in increasing or decreasing order , using this column we can sort the emails such that we can see all single emails at once or all the email conversations together.
5. From: It displays the sender name. For a single email it is the original sender name. In case of an email conversation. The sender name is the name of the person who initiated the conversation.
6. Attachments: The attachments column provides access to the email attachments if any for an email. If an email has attachments, the attachment column for that email will have a link “Click to view attachments”. This link

opens up a dialog box with a list of attachments present in that email. If the email does not have attachment the text “No Attachments is displayed.”

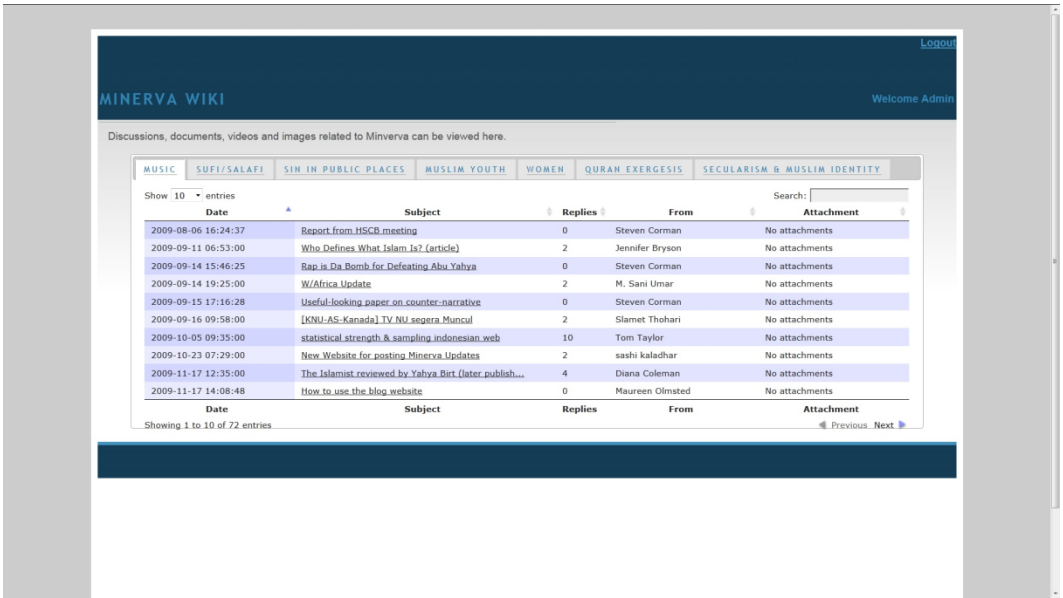

Figure 6.2 Screenshot of the Table.jsp page

6.2.3 Email Content Display

This page displays the email contents. When the user clicks on any email, a request is sent with the email subject and the category as parameters to this page. The jsp connects to the database and retrieves the email body content and displays it to the user.

A single email is displayed with the subject of the email as the header, followed by the content. An email conversations is displayed with the subject as the header followed by the body of the email. Each individual email starts with the sender details, the date and time information.

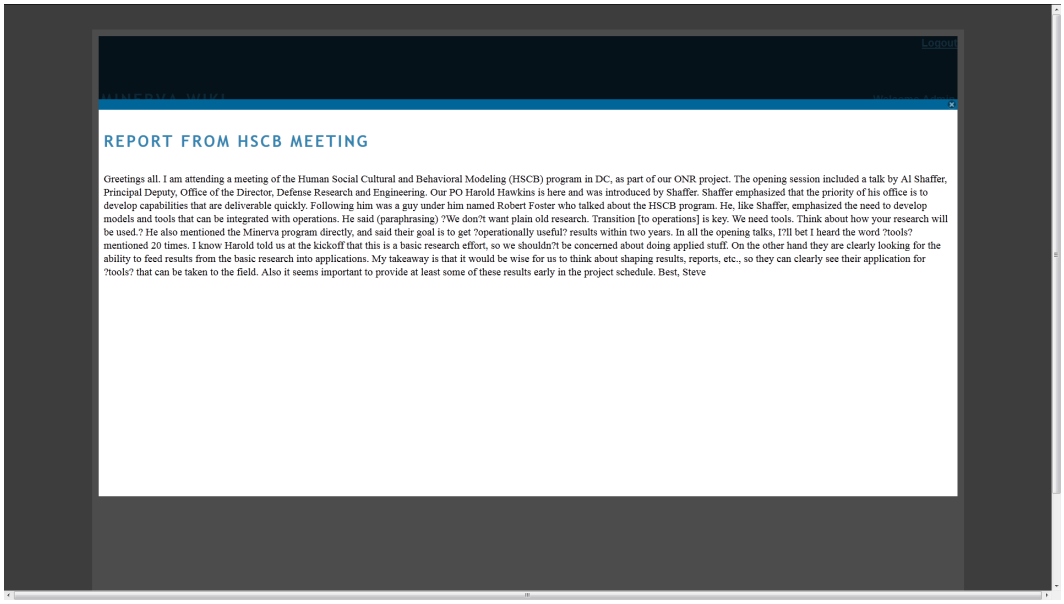

Figure 6.3 Screenshot of the Email content page for a Single email

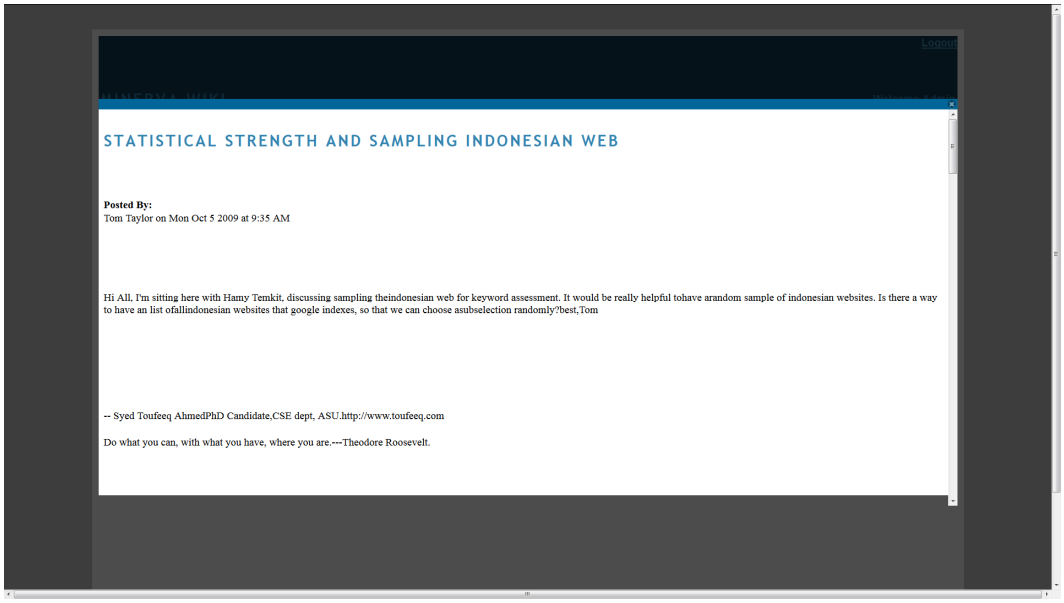

Figure 6.4 Screenshot of the Email content page for an email conversation

6.2.4 Email Attachmens Display

This page opens as a dialog box listing all the attachments an email has. It is linked with the main table through the "Click to view attachments". The attachment either opens up in a new window or gets automatically downloaded on the user's hard-drive.

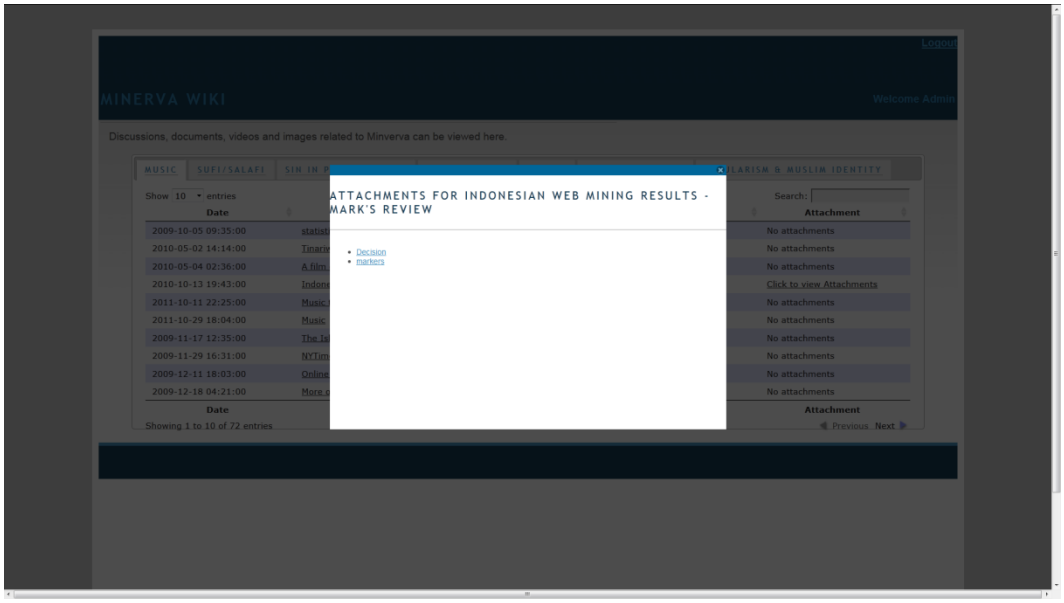

Figure 6.5 Screenshot of the Attachments Page

Chapter 7

CONTRIBUTION AND FUTURE WORK

The Minerva Wiki tool serves as a content management system for sharing project resources. It is secure, easy to use and well organized. It can provide useful information about the ongoing research, its current progress and future milestones. This can be a starting point for any individual new to the team and help him/her understand the status of the project in a convenient manner

As a part of the future work, we can devise more efficient classification methods in order to improve the accuracy of the tool. Also we can devise algorithms to find more keywords present in the dataset that can be associated with the categories. This will help us improve the efficiency and the utilization of the tool.

REFERENCES

Tikves, S.; Banerjee, S.; Temkit, H.; Gokalp, S.; Davulcu, H.; Sen, A.; Corman, S.; Woodward, M.; Rochmaniyah, I.; Amin, A., "A System for Ranking Organizations using Social Scale Analysis" (2011). Intelligence and Security Informatics Conference (EISIC), 2011 European

Y. M. Yang, T. Pierce, J. Carbonell. A Study on Retrospective and On-Line event detection. Proceedings of the 21st Annual International ACM SIGIR Conference,Melbourne, Australia. pp. 37-45, 1998.

Xiang-Ying Dai, Qing-Cai Chen, Xiao-Long Wang, Jun Xu. Online Topic Detection And Tracking Of Financial News Based On Hierarchical Clustering.

Bookstein and S.T. Klein, Detecting content-bearing words by serial clustering, Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 319–327, 1995

W.B. Croft and D.J. Harper, Using probabilistic models of document retrieval without relevance information, Journal of Documentation, 37:285–295, 1979.

R. Crelinsten, “Analysing terrorism and counter-terrorism: A communication model,” Terrorism and Political Violence, vol. 14, pp. 77–122, 2002.

H. Davulcu, S. T. Ahmed, S. Gokalp, M. H. Temkit, T. Taylor,M. Woodward, and A. Amin, “Analyzing sentiment markers describingradical and counter-radical elements in online news,” in Proceedings ofthe 2010 IEEE Second International Conference on Social Computing,ser. SOCIALCOM ’10. IEEE Computer Society, 2010, pp. 335–340.

W. Michael and J. Kogan, “Text Mining: Applications and Theory”. Wiley,2010.

G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” in Information Processing and Management, 1988, pp.513–523.

Feilmayr, C. ,”Text Mining-Supported Information Extraction
An Extended Methodology for Developing Information Extraction Systems”, Database and Expert Systems Applications (DEXA), 2011 22nd International Workshop on, 2011, pp. 217 - 221