# Improving Email Management

Tõnu Tamme, Ulrich Norbisrath, Georg Singer, Eero Vainikko
Institute of Computer Science, University of Tartu
Tartu, Estonia
tonu.tamme@ut.ee, ulrich.norbisrath@ut.ee, georg.singer@ut.ee, eero.vainikko@ut.ee

***Abstract*—For twenty years, email has been the prominent means of computerized communication. Each day we receive a growing number of email messages from different origins, related to different topics, people, and locations. Some belong to the professional sphere, others are private. Usually we keep our messages in the inbox or store them in several mostly manually created hierarchical folders. Showing information in hierarchies and lists can be nowadays amended by views which allow a more explorative approach to access this. The goal of this paper is to analyze the automatic information management capabilities of present standard email clients and webmail services, show their shortcomings, and show some improvements of them through the use of auto categorization and graph exploration. We show that categorization is not supported by traditional email tools but that it facilitates discovery of new relations between email messages and therefore improves email management.**

***Keywords*—email; exploratory search; categorization; n-grams; ontology.**

## I. Introduction

Email has undoubtedly conquered the position of being the most important means for written communication. Business and private life without email is unthinkable. However, this success has unavoidably entailed another challenge. The sheer amount of messages that we receive each day and the lack of appropriate tools even lowers productivity in some cases.

Today, we keep our messages in the main folder or store them in several specialized folders of our email program — one for each person or a group of persons, one for each institution, topic, or project. Most e-mail clients offer filters to do some of this sorting automatically, based on various keyword-like criteria. In spite of these sorting options, working through your emails and addressing relevant emails for your daily tasks becomes harder and harder with the growing load.

As a motivating example, we assume that we are working for a company, which is being audited. The auditor demands a list of ongoing projects between us and our clients and their interconnections. Therefore, we have to create a report describing each project, involved and related persons, and compile the material these persons worked on. We assume that there is no central file containing this information. The only sources are the email conversations between the employees and the respective clients (including various email attachments). The amount of emails to consider will be more than several thousands. In this paper, we will in particular look at the possibility to list all attachments between a group of persons and creating interconnections via conversation topics.

Addressing this with current email clients offers the following options: Standard email programs like Outlook, Thunderbird and webmail services like Gmail, Hotmail, or Yahoo mail allow manual sorting of email by sender, recipient, date received and date sent. Furthermore, they allow to use standard keyword search over the body and over some parts of the header. Gmail is the only solution offering a working and highly performing full text index. Gmail and Thunderbird both allow manual tagging, but with an increasing amount of tags, the management effort makes it less usable. All standard email systems support basic filtering functionality using patterns. In case of having emails sorted into person related folders, a task like described in the motivating example above will be time consuming and tedious as project related information is distributed over all those folders. If extensive manual tagging has been done in foresight of such a task, the effort will be significantly smaller. However, such extensive tagging is usually not done or not even supported by the email client.

To get an overview for such a report, exploring context related data in the emails would be very helpful. "Exploring" is used here in the sense of exploratory search. Exploratory search is defined in [1] like the following: *"Exploratory search can be used to describe an information-seeking problem context that is open-ended, persistent, and multi-faceted; and to describe information-seeking processes that are opportunistic, iterative, and multi-tactical. [...] In exploratory search people usually submit a tentative query to get them near relevant documents then explore the environment to better understand how to exploit it, selectively seeking and passively obtaining cues about where their next steps lie."* Exploratory search in emails is nowadays mainly based on browsing your own folders and tags as well as creating multiple queries for retrieving messages matching keywords.

There is more email search support from several recently emerged addons for Outlook and Gmail like Xobni [2], [3] or Xoopit [4]. Nevertheless, also these tools do not lighten the complexity in the aforementioned example. The shortcomings of the existing email solutions are mainly due to the following: None of the existing solutions have support for categorization via text analysis nor exploration apart from folder and tag browsing by any means. In this paper we suggest the combination of a categorization method and the automatic association of these detected categories with the mails and the involved persons.

This paper is organized as follows: Section II gives an overview of related work and tries to motivate our approach. Section III outlines our enhancement to classic email management. Section IV shows some results using these enhancements. Section V shows our conclusions and outlines future

 ISBN: 978-1-61208-162-5

work.

## II. State of the art

This section will give an overview of the features of today's email systems and possibilities to analyze and enhance them. The focus is on their support of search and information management and their possibility to enhance automatic email management. We will analyze the methods and algorithms for automatic categorization and tagging on the one hand and their realization in email clients and email client plug-ins on the other hand. For a more practical and case driven analysis, we use the Enron Email Dataset. The dataset is a collection of 500.000 emails, organized in folders, that contains information from 150 senior management staff members at Enron. It is one of the few substantial "real" email repositories that was made available to the public for the purpose of improving email tools.

### A. Email clients and email clients plug-ins

A study from June 2011 with over a billion of emails [5] shows a market share of email clients of Microsoft Outlook with 27% followed by 16% iOS devices (iPhone, iPad and iPod Touch), 12% Hotmail, 11% Apple Mail and 9% Yahoo Mail. Gmail has 7% market share being used as an email client, Android 1.7% and Thunderbird 1.2%. Microsoft Outlook, Apple Mail and iPhone Mail are the only clients, not coming with a full text search function over a full text index. Outlook can be enriched with a full text index through Google Desktop or Microsoft Search. Microsoft Outlook, GMail, and Thunderbird support manual tagging, but interoperability between different mail systems (with an exception of Gmail's IMAP tag emulation) is very limited. The share of the clients is recently shifting heavily to mobile clients (like iPhone mail and Android, which is basically Gmail).

Standard email clients involve different visualization and tagging techniques. For example Mozilla Thunderbird categorizes search results into suitable time intervals like years, months, or days of month.

One of our main complaints about most existing email clients is their lack of support for networking emails with each other. They only support manually sorting emails in hierarchic folders. Without duplication, it is not possible to assign an email to several topics. Tagging supports such a way of sorting, but usually breaks the order of hierarchies and therefore allows only flat ordering. Tagging allows to add special markers to emails with the aim of grouping similar messages together and making their finding easier without having to sort the mail in a strictly hierarchical directory structure. In some clients a different name might be used for this function, such as marking, labeling, categorizing, or adding keywords. Some clients (Mozilla Thunderbird, Microsoft Outlook, Opera Mail, Gmail) also enable the user to define new tags. Due to the flat nature of such tags, using tags does not really help to create structure. Therefore, there is usually only a limited set of tags suggested. For example Thunderbird and Gmail suggest personal and work tags. Gmail also has by default a separate tag for traveling purposes. Thunderbird and Opera also have a todo tag for emails indicating an assignment or task. Often it is possible to tag spam or junk messages. This function helps the client to avoid unwanted messages automatically in the future. Opera offers also "send reply", "call back" , "funny", and "valuable" tags by default. Apple Mail does not support tagging of emails, it has some support for flagging emails like staring mails in Gmail or marking mails as junk. It allows to create virtual folders, this means different views, but this does not allow to define tags as these are only views not folders where emails can be moved.

An interesting search task is tracking the attachments exchanged between two persons. The task is not trivial as it involves two persons and there must also be an opportunity to check the attachments. If such an attachment check is not part of the search, sorting can also be used. An attachment-based search is not possible in Yahoo Mail and Windows Live Mail because they lack the OR operator that enables matching the same name for both sender and receiver. The task is not difficult in Microsoft Outlook, but in Thunderbird and Opera Mail sorting has to be used to filter out messages with attachments.

There are several plug-ins available, enhancing the experience in email systems:

*Xobni* [2], [3] is the most common protagonist of the group of Outlook add-ons. As a plug-in it allows keyword based search and people based navigation in their Outlook mailboxes. Xobni extends the support for person related information and operates as an integrating platform between Outlook and online services like Facebook, Twitter, and LinkedIn. It automatically creates profiles of persons and their connections. These profiles contain statistics about relationships, contact information, threaded conversations, shared attachments, and information on that contact pulled from earlier mentioned online services.

*Nelson Email Organizer (NEO*) [6] is an Outlook add-on allowing different views of all messages in the inbox. It offers different views in different tabs and allows to organize messages in these tabs by date, by sender, or by attachment. All views support full text index keyword search. These are helpful features. However, they address no major conceptual simplification.

The Firefox extension *Xoopit* [4] turns Gmail into a robust, searchable media management tool for every piece of media that comes through the inbox. By indexing every attachment as well as every link to photos and videos from sites like Flickr, Picasa, and YouTube, Xoopit allows to easily search for and find any picture or video and view it from directly inside Gmail. XOOPIT was acquired by Yahoo in July 2009 and is now integrated into the Yahoo mail environment.

*TaQuilla* [7] is a Thunderbird extension which can be trained to "soft tag" incoming emails for different categories depending on an existing training set. It uses Bayesian analysis of the tags already given on stored emails to do this classification. In order to make the Bayesian filter work for a new tag, the existing messages have to be trained by showing the system examples of emails that carry the tag and messages that do not carry the specific tag. This needs to be done manually at the beginning for a certain amount of messages

 ISBN: 978-1-61208-162-5 

until the system can take over. For example as outlined in [8] a user applied Taquilla to automatically separate "Personal" and "Business" related (not personal) messages. On the "Personal" side Taquilla analyzed a weekly general mailing from the user's church's pastor and came up with tokens like life, pastor, worship, sunday, thinking, and children. All those tokens showed an over 80% probability that the messages should be tagged as personal. On the business side on the other hand, it let Tacquilla analyze a posting on the Thunderbird testing list that was not tagged personal. It showed for other tokens like advance, feedback, bug, vista, and Thunderbird with an under 10% probability that these tokens were personal.

### B. Integrative frameworks

*ClearContext* [9] is an email experience enhancing Outlook plugin. It has a very task driven integrational approach. It is very contact focused and supports scraping of appointment data from the emails. It helps filing, prioritizing emails based on sender, unsubscribing from conversation, defering emails to come back to your inbox, and various task and appointment management functions. It integrates these features into convenient workflows. Task management and communication is lightly integrated, but our idea of networking all the information of communication participants, topics, dates, and content into a free explorable graph is not realized.

*Radar [10]* is a research prototype also trying to advance task and email integration. It consists of several assistants employing machine learning. It supports automatic categorization of emails and scraping of relevant data out of emails and using this for guiding users through corresponding tasks. Regrettably, the product is not available as prototype or at the market.

*Windows Search*, *Spotlight* (Apple), *Google Desktop* are very similar. They all create a full text index of all files and emails stored on your computer and in case of Google also of mails stored on your Google account and make them accessible in a local Desktop query based keyword search. There is no special exploration support [11], [12], [13], [14], [15].

None of these tools or systems allow the exploration of the data pool contained in a standard email repository. Radar and ClearContext offer via their task integration some light support, but not in a graph-based manner. Thus, the interaction with today's email systems is still typically based on keyword search queries and browsing of manually maintained folder or tag hierarchies and the referenced messages.

As shown in the paper by Singer et. al. [16] keyword based search alone does not cover many of nowadays information needs. This is of course also true in the email context. The results support the hypothesis that also in the email context a graph based exploratory approach will significantly improve the search experience.

### C. Automatic categorization and tagging

As already mentioned in the context of the example described above, manual tagging can be a tedious process. As machine learning algorithms become better with progresses in research, it makes sense to use these to facilitate the tagging or categorization process. Dredze et al. present an approach for generating summary keywords for emails in [17]. Their approach selects a set of keywords describing a single message in context to the topics of all messages. Their method is unsupervised. They are using Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDS) to determine a minimalistic amount of possible latent concepts. They use for their research the very versatile toolkit MALLET [18]. MALLET stands for MAchine Learning for LanguagE Toolkit. It is based on Java and supports statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning mechanisms applicable for text-based problems. Due to its huge variety of tools for text-processing, it is very well suited for carrying out experiments for topic and categorization analysis. It supports also the just mentioned LDS method.

It is also possible to use n-gram-based Text Categorization [19]. This method uses fingerprints of training texts and allows to compare other texts to these fingerprints. One of the main usages for this method is language detection. N-Gram distribution is very specific to languages, making it a very good choice for language detection. LibTextCat [20] is a freely available software supporting this method.

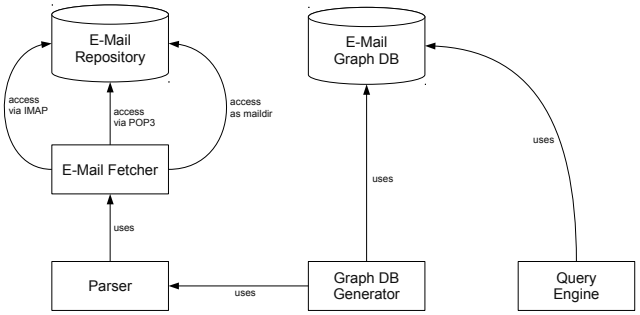


Figure 1. Email experimentation environment architecture.

## III. PROPOSAL

We have shown in Section II that current email clients have problems with handling attachment search and creating interconnections via topics.

To overcome these shortcomings we have built an experimental email handling environment. The tool consists of four components: email fetcher, parser, database generator, and query engine. The architecture is depicted in Figure 1. The parts are acting autonomously. Thus they can be easily replaced with other suitable software for experimentation purposes.

The email fetcher reads the contents of an email account using the IMAP protocol. It facilitates retrieving information from different accounts. The email parser transforms a mailbox into a list of messages. Each email is a list of attribute value pairs. The body of a message is parsed into a three level list: paragraphs consist of lines and lines itself consist of words.

The first two components are written in Python as it has special libraries for dealing with emails and mailboxes.

 ISBN: 978-1-61208-162-5

Database generation and query engine are implemented with Prolog that has built-in support for presenting relations and exploring the search space due to its predicate structure.

The database generator transforms a list of email messages into a graph database that consists of labeled nodes and edges. For example a message node may have related nodes for its sender, receiver, subject, or date. A sender of a message may be receiver of another message. Thus, the graph database does not need to duplicate their nodes.

The text-based query interface provides a set of tools for visualizing and analyzing the message database. We can observe all the messages in a mailbox. We can also filter out all the emails from (or to) a person or between two persons. This can be done by tracing the values of From, To and Cc-attributes of messages.

In addition to using existent person→message relations Prolog helps in constructing new virtual relations on top of existent ones. To allow inferential relations between messages it will be beneficial to assign to each message 5–10 keywords. This provides us with alternative views to our email collection that can produce novel deduced relations not covered by the existing thread structure. With the help of message→topic relations we can build two new relations: person→topic and topic→person.

Several text analysis methods can be used to determine the topic of a message. Our first choice was to calculate the word frequency table. This method is language dependent and must be amended with a suitable list of stopwords and a stem picking routine.

The relative frequency method eliminates the need for using a stopword list. We calculate the frequency tables for the whole message set and for a chosen message. Comparing those frequencies for a word in the message we can choose potential keywords as words whose local frequency in a message is bigger than the global frequency in our whole text corpus. The frequency tables can be given in advance or built in the course of the analysis.

Some experiments to classify text with the help of n-grams have been carried out. This method relies on the availability of sample texts about potential topics.

For our classification we also link to large freely available ontologies like WordNet [21] and OpenCyc [22]. The ontologies consist of concepts, their definitions, and corresponding terms. Words acting as terms in an ontology are good candidates for a topic of an email message. Concepts and their hyperonyms are the basis of the conceptual network.

We have tried those methods to find the topics of a message automatically. Then we add the topics to the existing email graph and execute on it Prolog queries to resolve indirect connections between messages.

## IV. Experiments

We made experiments with several mailboxes and chose as a testbed the Enron email dataset that has been made publicly available by the federal commission after the bankruptcy of the American company. The dataset consists of about 150 personal mailboxes and half million messages. A sample Enron message looks like the following:

```
Message no 844
From = John Arnold
To = Frank Hayden
Date = Fri, 14 Jul 2000 19:46:00 +0200
Subject = Re: Market Opinion about AGA's
Interesting observation...but I'm not sure I
    agree. I think consensus opinion is that
    anything under 2.7 TCF is very dangerous
    entering the winter. A month ago, analysts
     were predicting we would end the
    injection season with around 2.6 -2.7 in
    the ground. /.../
```

With the method described in the last section, we store it in our graph database in which the node numbers correspond to the internal construction of the graph. This will look like the following:

```
email(844).
node(844, 'XXX').
edge(844, 845, body).
node(845, [[['Interesting', 'observation...but
    ', "I'm", not, sure, 'I', 'agree.', 'I',
    think, consensus], [opinion, is, that,
    anything, under, '2.7', 'TCF', is, very,
    dangerous, entering, the], ['winter.', 'A
    ', month, 'ago,', analysts, were,
    predicting, we, would, end, the, injection
    ], [season, with, around, '2.6', -, '2.7',
     in, the, 'ground.', ...] ...]]).
edge(844, 847, subject).
node(847, 'Re: Market Opinion about AGA\'s').
edge(844, 848, from).
node(848, '"John Arnold"').
edge(844, 850, to).
node(850, '"Frank Hayden"').
edge(844, 851, date).
node(851, 'Fri, 14 Jul 2000 19:46:00 +0200').
```

As we have a graph database, a query for all attachments between groups of people is very easy to realize in contrast to classic email clients, which only rarely and insufficiently support such a query. In the following example, we list all the emails containing attachments exchanged between John Arnold and Mark Sagel. A person's name can be identified as a substring in his email-address. While collecting data the following basic query is also formatting the output.

```
?- F='"John␣Arnold"', T='"Mark␣Sagel"',
|      findall(X:From->To:Subject:Date->Attachment,(
|      email(X,Id,From,To,Date,Subject,Body),
|      (sub_atom(From,_,_,_,F), sub_atom(To,_,_,_,T)
|      ; sub_atom(From,_,_,_,T), sub_atom(To,_,_,_,F
    )),
|      edge(X, Y, 'attachment'), node(Y,Attachment))
    ,List),
|      print_list(List), length(List,N).
```

The output contains similar records as the query is not optimized to detect duplicate messages. The Enron dataset contains several of such duplicates due to its hierarchic nature.

```
3380:"John␣Arnold"->"Mark␣Sagel" <msagel@home.com> :
     Re: Natural gas update:Mon, 14 May 2001
    09:33:00 +0200->ng051301.doc
3389:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Natural gas update:Mon,
    14 May 2001 00:23:00 +0200->ng051301.doc
3397:"John␣Arnold"->"Mark␣Sagel" <msagel@home.com> :
     Re: Service Agreement:Mon, 27 Nov 2000 19:48:00
     +0200->Agree-Enron.doc
```

 ISBN: 978-1-61208-162-5

```
3406:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Service Agreement:Mon, 27
     Nov 2000 18:16:00 +0200->Agree-Enron.doc
3414:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Status:Thu, 16 Nov 2000
    19:30:00 +0200->energy2000-1112.doc
3422:"John␣Arnold"->"Mark␣Sagel" <msagel@home.com> :
     Re: Natural gas update:Mon, 14 May 2001
    09:33:00 +0200->ng051301.doc
3431:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Natural gas update:Mon,
    14 May 2001 00:23:00 +0200->ng051301.doc
3439:"John␣Arnold"->"Mark␣Sagel" <msagel@home.com> :
     Re: Service Agreement:Mon, 27 Nov 2000 19:48:00
     +0200->Agree-Enron.doc

...

3545:"John␣Arnold"->"Mark␣Sagel" <msagel@home.com> :
     Re: Service Agreement:Mon, 27 Nov 2000 19:48:00
     +0200->Agree-Enron.doc
3554:"John␣Arnold"->"Mark␣Sagel" <msagel@home.com> :
     Re: Natural gas update:Mon, 14 May 2001
    19:33:00 +0200->ng051301.doc
3572:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Natural gas update:Mon,
    14 May 2001 00:23:00 +0200->ng051301.doc
3580:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Service Agreement:Mon, 27
     Nov 2000 18:16:00 +0200->Agree-Enron.doc
3588:"Mark␣Sagel" <msagel@home.com> -> "John␣Arnold"
     <jarnold@enron.com> : Status:Thu, 16 Nov 2000
    19:30:00 +0200->energy2000-1112.doc
F = '"John␣Arnold"',
T = '"Mark␣Sagel"',
List = [ (3380:'"John␣Arnold"'->'"Mark␣Sagel"␣<
    msagel@home.com>':'Re:␣Natural␣gas␣update':'Mon,
    ␣14␣May␣2001␣09:33:00␣+0200'->'ng051301.doc'),
    (3389:'"Mark␣Sagel"␣<msagel@home.com>'->'"John␣
    Arnold"␣<jarnold@enron.com>':'Natural␣gas␣update
    ':'Mon,␣14␣May␣2001␣00:23:00␣+0200'->'ng051301.
    doc'), (3397:'"John␣Arnold"'->'"Mark␣Sagel"␣<
    msagel@home.com>':'Re:␣Service␣Agreement':'Mon,␣
    27␣Nov␣2000␣19:48:00␣+0200'->'Agree-Enron.doc'),
     (3406:'"Mark␣Sagel"␣<msagel@home.com>'->'"John␣
    Arnold"␣<jarnold@enron.com>':'Service␣Agreement'
    :'Mon,␣27␣Nov␣2000␣18:16:00␣+0200'->'Agree-Enron
    .doc'), (3414:'"Mark␣Sagel"␣<msagel@home.com>'->
    '"John␣Arnold"␣<jarnold@enron.com>':'Status':'
    Thu,␣16␣Nov␣2000␣19:30:00␣+0200'->'energy2000
    -1112.doc'), (3422:'"John␣Arnold"'->'"Mark␣Sagel
    "␣<msagel@home.com>':... : ... -> 'ng051301.doc'
    ), (3431:'"Mark␣Sagel"␣<msagel@home.com>'->... :
     ... -> 'ng051301.doc'), (... : ... -> ... ->
    ...), (... -> ...)|...],
N = 21.
```

The query can be enhanced and stored as a predefined predicate. The sorted list without duplicates looks like the following (Prolog code for actual sorting and removing duplicates omitted):

```
?- attachments_between('"John␣Arnold"','"Mark␣Sagel"
    ').
16 Nov 2000 19:30 ("Mark␣Sagel"  -> "John␣Arnold")
    Status          (energy2000-1112.doc)
27 Nov 2000 18:16 ("Mark␣Sagel"  -> "John␣Arnold")
    Service Agreeme (Agree-Enron.doc    )
27 Nov 2000 19:48 ("John␣Arnold" -> "Mark␣Sagel" )
    Re: Service Agr (Agree-Enron.doc    )
14 May 2001 00:23 ("Mark␣Sagel"  -> "John␣Arnold")
    Natural gas upd (ng051301.doc       )
14 May 2001 09:33 ("John␣Arnold" -> "Mark␣Sagel" )
    Re: Natural gas (ng051301.doc       )
14 May 2001 19:33 ("John␣Arnold" -> "Mark␣Sagel" )
    Re: Natural gas (ng051301.doc       )
true.
```

This result shows how beneficial a graph based structure is for creating a query about the exchange of attachments.

We have amended our emails via topic detection mechanisms discussed in the state of the art section with related keywords. Looking at the initial email example from John Arnold to Frank Hayden, we observe that John Arnold is concerned about events in winter. Thus we can try to find links to other messages dealing with this season. To construct the keyword list we first filter out stopwords and other ill-formed strings of messages like MIME code and HTML tags. We discover that the word "winter" is listed among the top ten keywords of the following message:

```
?- keywords_message(1128).
--Top10
5, prompt
5, $
3, position
3, futures
2, winter
2, stress
2, spread
2, scenarios
2, payout
2, normal
```

This message has a different subject than our initial message ("Re: Stress Testing"). Therefore, we have discovered here an interconnection between two different threads, solving our second described problem of finding such interconnections through exploration. We are able to discover two persons who are linked via a topic they discuss but who are not directly related via a thread. This is a property not derivable in a classic email system setup. The underlying automatically generated graph structure makes it possible to explore this relation.

## V. Conclusions and Future Work

In this paper, we have given an overview of the state of the art of automatic information management capabilities of today's email clients and webmail services. We have discussed their shortcomings and have presented ideas for their improvement by using auto categorization and graph exploration. Email has been the prominent means of computerized communication for a quarter of a century. However the core technology has remained unchanged. Especially an update of the information management capabilities seems to be needed for dealing with an ever increasing number of emails. We also have presented the results of our experiments with our email handling tool.

In our future work it will be possible to extend the capabilities of the email handling tool by providing a simpler user interface and extending it with graph navigation and visualization facilities. We are also planning to switch from n-gram and frequency based categorization to LDS based categorization and apply our methodology to a bigger dataset.

## Acknowledgment

This paper was supported by the European Social Fund through the Estonian Doctoral School in Information and Communication Technology.

 ISBN: 978-1-61208-162-5

## REFERENCES


[1] R. W. White, G. Marchionini, and G. Muresan. Evaluating exploratory search systems: Introduction to special topic issue of information processing and management. *Information Processing and Management*, 44:433–436, 2008.

[2] D. E. Descy. Microsoft add-ons and updates. *TechTrends*, 54(2):7–8, 2010.

[3] R. L. Scott. Wired to the world: Xobni. *North Carolina Libraries*, 66(3):64, 2009.

[4] Yahoo! acquires xoopit. Available from: http://www.myphotos.yahoo.com/ [cited August 24, 2010].

[5] Email client popularity: June 2011. Available from: http://www.campaignmonitor.com/stats/email-clients/ [cited July 17, 2011].

[6] NEO – the microsoft outlook email software Add-On. Available from: http://www.caelo.com/ [cited August 24, 2010].

[7] R. Kent James. MesQuilla » TaQuilla. Available from: http://mesquilla.com/extensions/taquilla/ [cited August 18, 2010].

[8] R. Kent James. MesQuilla » blog archive » TaQuilla provides automatic "soft" tags for messages. Available from: http://mesquilla.com/2009/02/26/taquilla-provides-automatic-soft-tags-for-messages/ [cited March 7, 2011].

[9] ClearContext – Outlook plugin to organize email and manage inbox. Available from: http://www.clearcontext.com/ [cited August 18, 2010].

[10] M. Freed, J. Carbonell, G. Gordon, J. Hayes, B. A. Myers, D. Siewiorek, S. Smith, A. Steinfeld, and A. Tomasic. Radar: A personal assistant that learns to reduce email overload. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1287–1293, 2008.

[11] P. A. Chirita, R. Gavriloaie, S. Ghita, W. Nejdl, and R. Paiu. Activity based metadata for semantic desktop search. *The Semantic Web: Research and Applications*, pages 439–454, 2005.

[12] D. Aumüller and S. Auer. Towards a semantic wiki experience–desktop integration and interactivity in WikSAR. In *Semantic Desktop Workshop*, 2005.

[13] Duen Horng Chau, Brad Myers, and Andrew Faulring. What to do when search fails: finding information by association. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 999–1008, Florence, Italy, 2008. ACM.

[14] D. Pogue. Google takes on your desktop. *New York Times*, 2004.

[15] B. Turnbull, B. Blundell, and J. Slay. Google desktop as a source of digital evidence. *International Journal of Digital Evidence*, 5(1):1–12, 2006.

[16] Georg Singer, Ulrich Norbisrath, Eero Vainikko, Hannu Kikkas, and Dirk Lewandowski. Search-Logger – analyzing exploratory search tasks. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 751–756. ACM, 2011.

[17] M. Dredze, H. M. Wallach, D. Puller, and F. Pereira. Generating summary keywords for emails using topics. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 199–206, 2008.

[18] MALLET homepage. Available from: http://mallet.cs.umass.edu/ [cited March 7, 2011].

[19] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[20] libTextCat – lightweight text categorization. Available from: http://software.wise-guys.nl/libtextcat/ [cited August 20, 2010].

[21] WordNet: A lexical database for English. Available from: http://wordnet.princeton.edu/ [cited July 17, 2011].

[22] Cycorp, Inc. Available from: http://www.cyc.com/ [cited July 17, 2011].


 ISBN: 978-1-61208-162-5 