Conversation based chat-routing
Finding an expert

Herman A. van Rink

June 5, 2008

Status: Final
Supervisor: Marco Aiello
Second supervisor: Sietse Achterop
Industry supervisor: Michel Veenhuis, eMAXX B.V.

Faculty of Mathematics and Natural Science
Department of Computer Science
Abstract

This Master of Science thesis concludes a project done for eMAXX B.V., Chatfone B.V. and the University of Groningen. This thesis project introduces searchengine technology to Chatfone, a professional chat platform that allows website visitors to come in direct contact with an organization.

The possibility of assisting a Chatfone operator (the organizations employee) in finding the right expert to answer a question has been investigated. When an operator does not have the answer, he looks in his contact list for someone who knows more about the specific topic. The system presented here assists in this routing problem based on the current conversation. After analysis of the conversation text the system shows a list of candidates from which the operator can make a choice. The search results are improved by user feedback given after a conversation.

To start with, a comparison of the for routing relevant aspects is made between telephone, email and chat conversations.
Contents

1 Introduction 1

2 Routing conversations 3
  2.1 Conversation 3
  2.2 Telephone conversations 4
  2.3 Email conversations 5
  2.4 Chat conversations 7
  2.5 Media compared 8
  2.6 Classification and destination selection 9
  2.7 Performance metrics 11
  2.8 Conclusion 11

3 Background techniques 12
  3.1 Information Retrieval 12
  3.2 Question answering 12
  3.3 Stemming 12
  3.4 Stop-word removal 13
  3.5 Query expansion 13
  3.6 Bayesian analysis 13

4 Problem 14
  4.1 Motivation 14

5 Existing architecture 16
  5.1 Chatfone 17
  5.2 Mid Office 20
  5.3 Lucene 22

6 Requirements 24
  6.1 Functional requirements 24
  6.2 Quality requirements 24

7 Detailed design 25
  7.1 Which entities to choose from? 25
  7.2 Building an index 26
  7.3 Training 27

8 Text processing 29
  8.1 Text analysis 29
  8.2 Cutting down on conversation size 30
  8.3 Discussion 31
    8.3.1 Shallow parsing grammar 32
    8.3.2 Query expansion 33

9 Prototype Implementation 34
  9.1 Preprocessing 34
  9.2 Query execution 35
  9.3 Result presentation 36
  9.4 Post conversation training 39

10 Experimental evaluation 40
  10.1 Setup 40
  10.2 Measures 40
  10.3 Results 41
  10.4 Discussion 41
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flow diagram vector matching</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Flow diagram comparing the routing process in three media</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Global system relations</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Chatbutton as seen on <a href="https://eloket.groningen.nl">https://eloket.groningen.nl</a></td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>Conversation flow diagram</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Schematic view of transfer options</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>Mid Office Layers</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>Mid Office Layers [Pun07]</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>Lucene's context [HG04]</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>Estimate experts</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>Learning step after a conversation has ended</td>
<td>28</td>
</tr>
<tr>
<td>12</td>
<td>Page to evaluate the last conversation</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>Graphical output of Alpino for: “Ik zoek een hotel in Groningen.”</td>
<td>32</td>
</tr>
<tr>
<td>14</td>
<td>Sequence diagram of a typical request</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>Screenshot of tagging-beta chat side tab</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>Screenshot of tagging-beta chat side tab in advanced mode</td>
<td>37</td>
</tr>
<tr>
<td>17</td>
<td>Screenshot of the visitor window</td>
<td>38</td>
</tr>
<tr>
<td>18</td>
<td>Page to evaluate the last conversation</td>
<td>39</td>
</tr>
<tr>
<td>19</td>
<td>Screenshot of tagging-beta chat side tab - large</td>
<td>48</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Conversation media compared</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Text statistics with the default DutchAnalyzer</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>Text statistics from J128 chat transcripts</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>List of most frequent terms, top 15</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>Text statistics with DutchChatAnalyzer</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>Text statistics with DutchChatAnalyzer - verification batch</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>Apache bench responsetime statistics</td>
<td>41</td>
</tr>
<tr>
<td>8</td>
<td>Requirements - evaluated</td>
<td>42</td>
</tr>
</tbody>
</table>
1 Introduction

Chatting online has become ever more popular in recent years, most of the chat activity is for personal communication and fun. But as chat becomes more mainstream, it has also shown to be an effective business tool. To facilitate the professional market, Chatfone B.V. has developed a web-based chat platform. The main application on this platform is known to customers as “de Chatfone”.

Any professional organization can take a subscription to this hosted solution. After receiving the login information they can get started. Current users include governmental bodies, health care organizations and real estate agents. The organization just has to login to the webapplication and can start answering questions immediately. Depending on their organizational structure they can create login accounts for additional employees themselves. Visitors of the organizations website can now get in contact by clicking on the special chat-button placed on the website. This button connects them to a preconfigured department in the Chatfone application, of which the logged in employee is a member.

In recent years the number of subscriptions has grown steadily. This is one of the signs that the professional market is also embracing chat as a business communication tool. Based on customer feedback the platform has been continually improved in both functionality, manageability and ease of use. One part that has been identified as a potential for improvement is the manner in which an employee is assigned to an incoming request. This process can be assisted by software to make sure that the most suitable employee is answering. This can be a new call that is coming in, in which case the most suitable member of the specific department is chosen. Another interesting case is where the employee does not have the answer and wishes to transfer the conversation to someone that does. To accomplish this transfer, there needs to be an index of people’s knowledge profiles that is searchable. Although making a profile of possible knowledge attributes is nothing new, the combination with chat has not been done before. Chapter 4 will discuss the problem further.

For this purpose a thesis project has been started under supervision of eMAXX B.V., which provides a Mid Office platform to a number of Chatfone customers. This Mid Office platform is build on Java technology and facilitates communication between front- and backoffice systems. Their main objective is to see their technology applied in a new business area. Currently they operate mostly in the government and agricultural market. Section 5.2 discusses more about eMAXX’s technology.

The stakeholders, and their main objective, in this project are:

- Chatfone B.V. - to improve efficiency and customer satisfaction
- eMAXX B.V. - to see an extra application of their platform
- University of Groningen - academic exercise and research

The scientific goal of this project is to gain a better perspective of the process of routing a conversation and the information that can be of use for this. With this improved perspective further improvements and automation in the conversation routing process come a step closer to being realized.

The next chapter will discuss the problem of finding the most suitable destination for a conversation in other research areas. The initial research was done mostly into the individual techniques, in a later stage a comparison has been made between chat, email and telephone. These media can all deal with conversations, in the case of email a thread of emails can form
a conversation. Although the comparison is not exhaustive, it serves the purpose of giving an insight in the possibilities.

After the comparison is made, and a description is given of several techniques, the problem statement for this thesis will be made in Chapter 4. Thereafter Chapter 5 introduces the existing structures that are used. Chapter 6 states the requirements for this project. After which Chapter 7 elaborates on the design decisions that have been made. Then Chapter 8 dives into the analysis of text. A prototype of the suggested solution is illustrated in Chapter 9 and evaluated in Chapter 10. And finally Chapter 11 holds a conclusion and thoughts for future work.
2 Routing conversations

Routing can be defined as the task of directing the forwarding of “something” towards a destination. The “something” in this definition can be anything from a data packet on a TCP-IP network [Dav88] to an email on the internet or a support request inside a large organisation. Routing therefore appears in many forms in our modern communication media and throughout a large spectrum of fields. Before focussing on the routing of chat conversations within the chatfone network, first an investigation is done in the aspects of routing in other media. The available information and methods are compared to gain a better perspective on the issues and challenges of routing.

In classic IP network routing, as well as on the internet, most traffic has a predefined destination. Although there are obvious exceptions to this, it holds for most traffic. At the source, and along the way, a destination address is available which enables the transport to take its course. On an email system the destination is in most cases an email address. For telephone audio traffic the physical phone corresponding to the phonenumber is usually its destination. Information about an object being routed, such as the destination address, is known as metadata. This metadata describes a piece of (extra) information about an object, for a telephone call this metadata includes for example the callers phonenumber.

In the case of telephone callcenters a new routing task arises after the connection is established to a frontoffice employee. This new routing task, which will further be referred to as conversation routing, directs a call within the organisation to the correct destination. In contrast the lower level routing of individual messages will be referred to as channel routing. In order to do conversation routing, the new destination has to be elicited from the initiating party. The elicitation process can be done in a number of ways. Ranging from fully automated solutions to a conversation with a human who knows the possible destinations. The focus here will be on professional communication where conversation is used to elicit the destination.

This conversation routing task can be applied to multiple communication media. The for Chatfone relevant chat medium will be compared to telephone and email. Although more media could be compared, the ones discussed here are the most prominent examples. It is also not the intention to provide an exhaustive comparison, it is rather to create an understanding of the common challenges and differences.

The next section will discuss conversations, starting with a definition and elaborating on the available information in this context. After that the different media, telephone, email and chat will be discussed. Then a comparison of the available data will be done for the different media. The selection of a destination is discussed in Section 2.6. After that performance metrics in Section 2.7 followed by a short conclusion.

2.1 Conversation

To understand what is involved in routing a conversation, first the concept of a conversation itself needs to be clear. This section will therefore discuss the concept of a conversation, which Liddicoat defines in [Lid07] as:

Conversation is the way in which people socialize and develop and sustain their relationships with each other. When people converse they engage in a form of linguistic communication, but there is much more going on in a conversation then just the use of a linguistic code.
This definition mostly reflects verbal conversations, where for example facial expressions and intonations can represent extra information. For face-to-face communication this works. But to communicate over a larger distance, a medium is required to transport the communication. On communication media, such as telephone, email and chat, the role of non-linguistic communication can be restricted due to technical limitations of the medium.

Facial expressions for example can not be transported over an, audio only, telephone line. And although an intonation can communicate information about someone's mood, the result is limited when compared to face-to-face communication.

The available data and metadata therefore depend on the communication medium. In the following sections the available data that can be used for routing will be identified for the media telephone, email and chat.

In general a conversation can be segmented into utterances, which make up contiguous fragments. The change of who is speaking in turn is an often used separation point for utterances. When parties start talking simultaneously this segmentation can become difficult, however this is not advisable for any conversation. In some cases is it possible to identify individual sentences within an utterance. Although conversation is less strict than formal written text, this differs per medium.

The focus is on professional conversations, where in most cases two persons take part.

### 2.2 Telephone conversations

Verbal conversations over a telephone connection has the focus of this section. The specific aspect that a regular telephone line introduces is the fact that only sound is transmitted. Modern multi-media solutions where additional channels support video communication are outside of the scope of this comparison.

When a telephone connection is open, there is no significant delay before the spoken words reach the other side. Therefore telephone connections can be viewed as a synchronous medium. Connections established to a larger organisation often start at a frontoffice department. In this department a choice has to be made about where to direct the call.

The most prominent methods to gather information on the desired destination are:

- Personal communication with a human
- Touch tone driven menus (IVR)
- Automatic Speech Recognizer (ASR) systems with speech based routing

Traditionally the first contact would be with a human telephone operator who could ask the preferred destination and either manually transfer the call, or ask for further information. This traditional method is the most flexible. And often remains available as a fallback option for other solutions.

The work of traditional telephone operators, in routing the conversation, has for a large part been replaced by interactive touch tone menus. These menus are also known a Interactive Voice Response (IVR) systems. The rigid hierarchical structure of these menu systems forces the end user to match his own need to one of the listed options. This hierarchical structure can be tuned to the available destinations to make the choice as clear as possible. An advantage of IVR is that the touch tone signals are relatively straightforward and not vulnerable to misinterpretation. Matching the destination is however ultimately delegated to the caller.
Another option can be found in Automatic Speech Recognizer (ASR) systems which are becoming available which can transcribe a conversation in real-time. The recognition of text in an audio file or stream however is still a difficult task. Periods of silence and a change in speaker indicate the start of a new utterance. An ASR is depending on features like audio quality and the pace of speaking to correctly recognize individual spoken words. The spoken dialect can also have a large impact on recognition ratios. The consequence of a low recognition ratio is that incorrectly recognized words can end up in the textual transcript. These incorrect words in the text transcript can lead to incorrect conclusions in later steps. Which in some cases would result in a transfer to the wrong destination.

The solutions developed in [DP01], [CCC98] and [GRW97] all describe a situation where an ASR system is used to convert the spoken words into text.

Durston uses in [DP01] the term call steering for the activity of directing a call to the right person. By only attempting to identify the topic of a conversation, less information is needed from the ASR system. Although good results were reached, a high proportion of user confirmations was needed before a final decision could be made. In these confirmations the caller would be asked to confirm or deny the found destination.

Solutions as presented in [DP01] and [CCC98, CCC99, RCCCC98] analyze incoming utterances from a conversation. And they attempt to match these utterances to a destination. In cases where their routing module does not come up with a single clear destination, Carpenter et al. [CCC99] work with a disambiguation module that can formulate a question to the caller.

The routing module used by Carpenter et al. [CCC99] used vector representations of the destinations and created a vector for the current call. The request vector must then be matched to all destination vectors on file, of which a distance can be calculated. Figure 1 shows a set of destinations (D1-6) as vectors plotted in a space. The left side of the figure contains the process in which the destination vectors are created. Under the “match” box a second vector space is show, but here with an additional vector representing the input from the conversation. The concentric circles are added to show the relative distances between the input vector and the destinations. The closest destinations to the input vector can be selected as candidates. This calculation can happen in various ways as described in [CCC99].

The disambiguation question can be in the form of a list of options (pronounced by a text-to-speech system, or pre recorded) or a request to repeat or reformulate the query. A text-to-speech system would be capable of pronouncing textual input, whilst pre-recorded solutions have a limited vocabulary which might be tailored to the applicable business domain.

Riccardi et al. in [GRW97] describe a system which analyzes the response to the open-ended question “How may I help you?”. Salient phrase fragments are matched to a corpus of earlier labeled fragments. The salience is a mathematical measure of indicating the information content of a fragment. [GAAJ02] Similar to Carpenter et al. they have the option of asking for confirmation.

2.3 Email conversations

Email messages sent back and forth between two or more parties can take the form of a conversation. For this type of conversation the email medium has become popular for business communication in recent years. An email message consists of a header segment with meta information, a message body and optionally any number of attachments. Technically the attachments are part of the body and MIME encoding separates these individual segments.
An email message has a single source and one or more recipients. An important aspect is the fact that email communication is asynchronous. This means that the other party in the conversation is not necessarily present, or going to reply immediately. A single email message does not constitute a conversation. This can be deduced from Liddicoat’s definition of a conversation above. A conversation can be modeled on top of email messages, only when a thread of messages is combined. A conversation modeled from a number of email messages can therefore span a larger amount of time than the previously discussed telephone conversation.

For the purpose of conversation routing however, it is not always necessary to have a conversation. Although the extra information can be of great use, a single message can be seen as a subset of that and can also be used for routing. When a subsequent message arrives a new routing decision has to be made. This can be as straightforward as sending it to the same employee that handled the previous message. Or a new selection can be done based on the content of the message. For example when the employee who handled previous messages is unavailable.

Compared to telephone conversations, individual messages can be parsed as utterances. An advantage of email is the fact that it is already in a digital text format. And therefore no solution like an ASR system is required. When available, punctuation can be used to separate utterances further. Filtering can be used to account for quoted sections from previous
messages.

The semi-structured form of email messages [MMT02] can provide extra information, which can be regarded a metadata. This includes the subject, sender and date fields included in the header segment of the message. The subject field can be expected to contain vital information such as keywords.

When the email originates from a web form that contains extra selection fields, the extra information can be compared to the feedback of an IVR system. The sender could for example be asked to select a topic from a pre-defined list of options, or enter a reference number. This pre-selection delegates part of the matching task to the end user.

Manco et al. describe in [MMT02, MMRT02] a mail classification system, which uses clustering algorithms to automatically filter and sort messages based on their content.

Lapalme et al. have explored methods to automatically follow-up on email messages in [LK03]. Three complementary approaches have been explored: Classification, case-based reasoning and Question Answering.

In [KW99] Khosravi and Wilks talk about routing email based on its function. With the function they mean a classification in messages containing primarily a statement of fact, a request for the recipient to do something, or ask a question.

2.4 Chat conversations

A chat conversation consists of (frequent) textual messages sent back and forth between two or more parties. These messages are usually transmitted one sentence (or less) at a time. The term Instant Messaging (IM), which has become a synonym for chat, signifies that a message is delivered without noticeable delay.

Although there is no need for an continuously open connection like with a telephone line or constant attention for incoming messages, the instantaneous aspect allows for a conversation to happen in a synchronous way. It is also possible for chat conversations to span larger amounts of time. In these cases where a conversation lasts longer, the notification system available in the messaging software becomes more important.

In the area of routing chat conversations little research was available. The research that is available on chat focuses mainly on four aspects:

- Channel routing - How the different platforms operate. [JNO+06]
- Separating parallel conversations
- Linguistic styles - Communicating emotions, and why it is less formal
- Topic identification

Historically the prominent chat platform has been Internet Relay Chat (IRC). An IRC network has channels where many parties can converse at the same time. After connecting to a relay server which is part of the IRC network, a user joins a channel after which he receives all messages sent to that channel from the server.

Most research into chat is therefore related to IRC and similar systems. Due to the nature of IRC these research projects, when related to routing, mainly focus on channel routing (as defined above for telephone conversations). The nature of IRC channels, where it is not uncommon for more then two people to have joined a channel, makes separating parallel conversations an appropriate research area. The identification of what a conversation is about, topic identification, can be useful in combination with separating parallel conversations. Following Liddicoat’s definition of a conversation, the non-linguistic information can be communicated.
in the form of emoticons. Emoticons are the little smiley faces used to express emotions in a
textual form, e.g. “:)” and “:(”. Although emoticons are less abundantly used in professional chat communication, its use is increasing.

In recent years multimedia chat solutions have become available, where e.g. audio and video channels are added to the text messaging. The scope of this project however limits us to professional text based chat conversations.

Most systems (including that of Chatfone) have information available on the source of each message. This identification of which party in the conversation says what is a vital part of metadata. The identity of the speaking parties is not always known, in the Chatfone application for example a website visitor can start a conversation without identifying himself first. In other systems both parties can be anonymous, although this is rarely the case for professional applications.

An optional pre-chat window can, depending on the organisations settings in the Chatfone, ask the visitor for a name and question before the chat conversation actually starts.

Individual messages can be identified as utterances. When people first start chatting online, one can observe behavior where multiple sentences to the extreme of complete letters, can be sent as a single message. For these cases further separation in utterances could be helpful. However when these new chat users become more experienced they tend to start sending shorter messages.

Other available metadata in the Chatfone application includes the URL from where a conversation was started. As mentioned in the introduction, a visitor of one of the by Chatfone supported websites can start a conversation by clicking a button or link on the webpage. The location of this website, including a specific page, is available to the operator during the conversation. There is also the visitors history, weather the person has had previous conversations with the organisation, and what these were about. This history and other, more technical, meta information includes the IP address from which the visitor made a connection. All of these can help an operator in making an informed decision about how to help the visitor.

2.5 Media compared

The previous three sections have discussed the media telephone, email and chat separately. In this section a comparison will be made between these different media.

The focus is on professional text chat conversations using the Chatfone application, therefore the other media are primarily compared to the chat medium. The architecture of Chatfone is used as a point of reference. First the more striking differences will be discussed and after that the similarities.

Compared to telephone conversations there is no need, by chat and email, for a solution to prepare the incoming data to gain a textual format, such as an ASR system. Therefore only telephone conversations have the added risk of incorrectly recognized input due to channel noise.

Meta information for the chat medium is richer. Partly because the input is already in a digital text format, is it easier to read data from the channel routing layer. In the case of Chatfone, since it is a web application, both parties are using a software interface that is under direct control. Therefore no problems arise with different implementations, and it is easier to gather more meta information.

Similar to email that is sent from a web form, it is possible to include one ore more pre-
selection fields. And compared to telephone the conversation can be synchronous. Utterances are separated when the speaking party changes.

Each medium has its own specific meta information available. For each medium there is an address available where the channel originates, however this does not lead to the identification of a person. This since all of these only represent a technical origin and can be manipulated (spoofing). Only via an extra method on top of the respective medium can the persons identity be verified. Although it is not always collected or available to the user, these parts of meta information are similar.

Table 1 shows the different aspects, as they were discussed in previous sections for each medium, in a tabular form.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Telephone</th>
<th>Email</th>
<th>Chat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Audio</td>
<td>Text</td>
<td>Text</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>ASR</td>
<td>Parsing headers</td>
<td>Architecture dependent</td>
</tr>
<tr>
<td>Channel noise</td>
<td>Possible</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Dialect problems</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Response</td>
<td>Synchronous</td>
<td>A-synchronous</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Known origin</td>
<td>Phone number</td>
<td>Email address</td>
<td>IP address</td>
</tr>
<tr>
<td>Identity known</td>
<td>Self introduction</td>
<td>Optional From field</td>
<td>Self introduction</td>
</tr>
<tr>
<td>Meta information</td>
<td>Limited</td>
<td>Standard headers</td>
<td>Architecture dependent</td>
</tr>
<tr>
<td>Utterance separation</td>
<td>Silence</td>
<td>Punctuation</td>
<td>Line</td>
</tr>
<tr>
<td>Emotion / non linguistic</td>
<td>Intonation</td>
<td>Emoticons</td>
<td>Emoticons</td>
</tr>
</tbody>
</table>

Table 1: Conversation media compared

From this comparison it can be concluded that although each medium has its unique properties, most differences are located on the channel routing layer. From the conversation routing perspective the steps are very similar. Figure 2 shows general steps from the routing process for the three media. After the preprocessing phase it can be seen that the possible steps are very similar. The phases following preprocessing are discussed in the following section.

2.6 Classification and destination selection

Previous sections have focussed on the input data gathered from a conversation. After input for the above mentioned conversation types has been prepared, the process of finding a destination can be very similar for these media.

In most cases there is a fairly static and limited collection of possible destinations. From these possible destinations a selection has to be made. For example a frontoffice department might have a list of all employees and departments with their respective number or extension.

Treating the selection of a destination as a classification problem is not uncommon. All destinations are then treated as a class. Different strategies have been developed to find the most suitable destination. In [KP], Kao et al. discusses four papers concerning text classification. The used techniques are Support Vector Machines, Naive Bayes, Hidden Markov Models and statistical part-of-speech tagging. The details of which are beyond the scope of this comparison. Further methods to optimize the available input for classification include stemming and stopword removal, both of which are explained in the next chapter.

Other examples of relevant research areas are Expert Systems [May06, MH02] and Decision Support Systems (DSS) [RHSC82]. Both use a form of classification to find a solution.
In [May06], Maybury makes a comparison of different commercial Expert Finding Systems. In Expert systems the central theme is the identification of experts in relation to their knowledge and experience. Datamining is used to gather data on potential experts. Possible sources that can be mined for information include internet pages, document management systems, email archives and other data repositories. The units containing knowledge fragments that can be retrieved are called “Experience Atoms” in [MH02] by Mockus et al.. These Experience Atoms could be processed and stored in an index for quick lookup. In the area of Decision Support Systems, the focus is more on the decision process. One such system is described in [Eno06], where Enomoto works on a system for supporting medical decisions. He uses soft computing to assist the use of protocols and guidelines for the Nursing Coordinators (NC). These NCs speak to patients, who have been discharged and are undergoing home care, over the phone and have to diagnose the situation. With these tools the NCs are able to focus their attention on the task at hand, not wasting precious time on needles data evaluation. The probability with which a classification is made can also be used as a ranking to order multiple candidate matches. These matches can then be presented to another system for further evaluation, or to a human operator. For the telephone medium the disambiguation module described by Chu-Carroll and Carpenter [CCC98, CCC99] can be used to present the available options in a sensible way. For the email and chat media this disambiguation is in most cases left to a human operator.

Figure 2: Flow diagram comparing the routing process in three media


2.7 Performance metrics

Depending on the solution, used for determining a destination, different performance metrics can be used. These metrics make it possible to evaluate the effectiveness of a solution. And to objectively compare solutions to each other. For typical information retrieval systems, precision and recall will be discussed below.

More technical metrics include the requirements in memory and processor capacity. And closely related to these physical requirements, the time it takes to come up with an answer.

The performance of an information retrieval system can be evaluated using the terms precision (Equation 1) and recall (Equation 2) \cite{BYRN99}. Where $|R|$ is the number of all available relevant items, $|A|$ the number of items within the answer set. And $|Ra|$ the number of items in the intersection of $|R|$ and $|A|$.

With these formulas a score can be calculated which can be used to compare different solutions.

\[
Precision = \frac{|Ra|}{|A|} \quad (1)
\]

\[
Recall = \frac{|Ra|}{|R|} \quad (2)
\]

Precision indicates the amount of relevant items in the answer set. This value is high when little or no irrelevant answers appear in the answer set. Recall indicates the amount of relevant items that are retrieved compared to the relevant items that were not retrieved. When relevant items are missed this value drops. When evaluating an information retrieval system the combination of a high recall and precision score would be desirable.

2.8 Conclusion

From the previous sections it has become clear that a conversation can take place in different media. Where each medium imprints its specific characteristics.

Characteristic for telephone conversations is the need for an ASR to preprocess the audio data and convert it to text. For email the most notable characteristic is the fact that it is asynchronous. And for chat the availability of more meta information during the conversation.

The terms “channel routing” and “conversation routing” have been introduced to separate different layers of routing functionality. After the next chapter explains several techniques, the following chapters will mainly discuss conversation routing.
3 Background techniques

This section will introduce several techniques and areas that are relevant to the following chapters.

It starts in the next section by explaining the general scope of the research field Information Retrieval. Section 3.2 discusses question answering. After that Section 3.3 will discuss stemming. Then stop-word removal is discussed in Section 3.4 Query expansion is shown in Section 3.5 And to conclude Section 3.6 discusses Bayesian analysis.

3.1 Information Retrieval

Information Retrieval (IR) generally stands for the science of searching for information in a collection of documents. In the past decades the area of Information Retrieval has grown rapidly [BYRN99].

Before the world wide web, most retrieval was done on large specialized systems. In recent years the used systems have become more and more sophisticated. This has given new focus to the area.

A broad variety of solutions has been developed to cater all sorts of specific cases. Ranging from solutions that retrieve information from text documents, images or other multimedia content. However most solutions use plain text as their main data source, or depend on converting a source to text.

3.2 Question answering

Question answering (QA) is one area of information retrieval that holds promise. QA systems can be divided in Open and Closed domain solutions. The advantage and main characteristic of a closed domain being the possibility to take advantage of specific domain knowledge. When working with an open domain, the diversity in input is large. Therefore the options of tailoring a solution to specific data are limited. For all these solutions a significant corpus with example data needs to be available.

The TextREtrieval Conference (TREC) is a yearly conference to stimulate the research in text retrieval. Each year a set of data and questions are published. Participants are encouraged to devise the best method of answering these questions in an automated fashion.

The Ephyra QA framework [SGS06] is an example of such a solution, which focuses on an open domain. Natural Language Processing (NLP) is used to analyze the questions After which several filters are applied to create a ranked list of answers.

3.3 Stemming

In stemming the words in a piece of text are replaced with their stem, for example “trees” is replaced by “tree” and “walking” by “walk”. This is done by removing morphological information from the word. In [Hul98] Hull makes a comparison between different stemming implementations. Baeza-Yates et al. [BYRN99] also describes stemming and gives a basic explanation of the working of the Porter algorithm. Which it considers the most popular algorithm for its simplicity and elegance. When stemming is done in the learning phase, it minimizes the index size. The same algorithm has to be applied to the search query to avoid mismatches. A side effect of stemming is that it reduces precision, in that the distinction
between different morphological forms ("walking" or "walk") is no longer available. Therefore stemming too aggressively (a.k.a. overstemming) should be avoided.

3.4 Stop-word removal

Stop-word removal entails removing non-discriminating terms from a piece of text. For English this includes terms such as “a”, “this” “to” and “are”. Removing these terms can significantly reduce a query to an information retrieval system and the size of an index in such a system. These non-discriminating terms are selected based on predefined lists, which are available for many languages.

In [FBY92] a list containing 425 such terms is documented for the English language. These terms can be left out without significant negative effect on performance attributes like recall and precision. Depending on the expected input these lists can be adapted to specific types of text. For example to better suit a particular dialect or business domain. According to [BYRN99] the reduction in text volume can be 40% or more.

3.5 Query expansion

Query expansion can help to broaden a search request, by adding related terms to the query. In [BYRN99] query expansion is defined as:

> A process of adding new terms to a given user query in an attempt to provide better contextualization (and hopefully retrieve documents which are more useful to the user).

With this technique the chance of finding not explicitly named data is increased. Most approaches are based on predefined synonyms for individual terms. These synonyms can be from domain specific or generic lists. In [QF93] a concept based method is presented which, instead of just the individual keywords, uses the entire sentence.

3.6 Bayesian analysis

Bayesian analysis is often used to classify (parts of) text. In [Gil02] a thorough description is given about the possible applications. In [Ner07] Nerbonne roughly sketches the basic idea and goes on to use it to determine the probable author of a text.

In the context of the current project, the name of the operator is usually known. The visitor’s identity could in a limited scope be determined using the mentioned method. This authorship attribution is however outside of the scope of this document. Another well known use of Bayesian analysis is the classification of email messages as SPAM or HAM. Based on characteristics that were trained with a specific classification, new data can be classified with a certain probability.
4 Problem

The problem in the existing Chatfone is that the actual question a visitor has does not weigh in the chat routing. For a new conversation an operator is selected purely on availability and current workload of the employee. When a chat needs to be transferred the destination department is chosen manually.

How can the routing and transferring of chat conversations, within the Chatfone network, be optimized to get the most suitable people answering a question? This should result in the best answerers to questions in the shortest amount of time. Which should in turn increase customer satisfaction for the organization using Chatfone.

To accomplish this it is necessary to analyze the conversation and metadata in realtime and let the software make suggestions on the most appropriate knowledge partners. This involves processing the natural language of an initial question and incrementally adding the ongoing conversation to a knowledge profile for this conversation. When the operator would like to transfer the call, the preprocessed conversation data should be updated. After which a set of suggestions can be generated.

After a chat is finished the operator should have the option to evaluate suggestions which let the system learn the operator's capabilities. This solution will have to be implemented and fitted to the situation.

The next section will elaborate on the problem and its scope.

4.1 Motivation

The problem of routing conversations between departments obviously does not occur in all organizations. In smaller setups the person online for a department is often the expert on the departments business domain. In these cases, only a small number of chat conversations would be transferred anyway.

In larger setups it is however feasible for chats to come into a central frontoffice department, which transfers the visitor to the most suitable department. Like it is done with telephone. This is the setup which most-frequently will be transferring a chat. Currently the choice to who a chat should be transferred is largely unassisted. An operator has the option to save favorite transfer partners for quick access, but the selection is still manual.

In the field of telephone callcenters, a similar problem exists. The main difference as identified in Chapter 2.5 however is that voice data is more difficult to analyze, although this has become more feasible in recent years. When a reliable speech-to-text solution is available, the solutions presented here should be adaptable to use a voice conversation as input.

Although the data describing the available operators is already accessible to the user, he has to browse his way to the specific data of interest. Information retrieval can improve on this by presenting the operator with the most relevant information. Thereby speeding up the process of finding the desired information.

Building the knowledge profiles can to a large extend happen automatically. However filling in an initial description about operators, departments and organizations helps to kick-start the index of knowledge profiles. With minor modification the description fields that already exist in the Chatfone can be used for this purpose. In Shirky talks about the motivation for individual users to train a system and how most of the value only comes when a larger group has done this. This also holds for this situation, because the system is most effective when many users have a substantial profile in the index. The advantage is only
noticed when other users have trained the system. Therefore it should be made very easy to
do some form of training. Possibly an almost automatic training process.
5 Existing architecture

This project uses several major existing structures, which are the Chatfone application, the eMAXX Mid Office and the Lucene searchengine library. In this chapter these structures will be described, an impression of the possibilities will be given using examples.

Section [5.1][1] covers the Chatfone webapplication which is written in PHP and runs on a Linux platform. This has proven to be a reliable platform and is well suited for rapid development of online applications.

For the solution developed in this project, a Java application was more suitable. The choice for Java is largely influenced by the availability of the Lucene project (described in Section [5.3][2]) and the eMAXX Mid Office (described in Section [5.2][3]).

Lucene is an open-source search engine library with a good support base. The library contains building blocks that can be used to add search functionality to applications. The support base consists of an active community of users communicating via mailinglist and other digital media. Making use of Lucene avoids having to do all the ground work, like maintaining a data storage format and parsing search queries.

![Figure 3: Global system relations](image)

The eMAXX Mid Office uses Enterprise JavaBean[1] and extends these to provide a generic infrastructure. This infrastructure is used for connecting Chatfone and Lucene. All Java code is running on a JBOSS[2] application server. Figure 3 shows the relation between Chatfone, the Mid Office and Lucene. Users connect to Chatfone over the internet, after which the Chatfone application initiates communication to its database and the TaggingService embedded in the Mid Office. The depicted “Tagging Module” houses this TaggingService and is specifically build to use Lucene in this context. It embeds the service into the same Java runtime environment and is described in more detail in Section[9].

The following sections will describe these in more detail. Section [5.1][1] describes the general working of the Chatfone application. Secondly in Section [5.2][3] a short overview of the eMAXX Mid Office is given. And Section [5.3][2] touches shortly on the history of Lucene and the theoretical models it uses.

5.1 Chatfone

The focus of Chatfone is to bring organizations together in a network and facilitate collaborative question answering. Currently only the Dutch language is supported, but the infrastructure for supporting additional languages is available.

All organizations have the option to transfer conversations to other departments. What is not seen in similar products is the option to transfer conversations outside of the organization. Which leads to a higher customer satisfaction because even if a question initially reaches the wrong department or organisation, it can still find its way to someone with an answer.

For example when a question reaches the city council of Groningen[3] that relates to starting a business. The visitor would have clicked on a button as shown in Figure 4. This kind of button is generally placed in a prominent position on the webpage, such as a menubar or a page with contact information. The design of which can be customized for every department. When a user clicks on this button, a small chat window pops-up which initiates a conversation with the predefined department. As this website and the people behind it, is dedicated to civil services, it is not the place to ask advise on running a business.

![Chatbutton as seen on https://eloket.groningen.nl](https://eloket.groningen.nl)

The central frontoffice department (Klant Contact Center or KCC) initially accepts the conversation. When evaluating the question, they have to choose if they are capable of giving an answer. Figure 5 shows a flow diagram for the choices that are made during a conversation. In this case the Chamber of Commerce (Kamer van Koophandel or KvK) is the option of choice in the “determin transfer destination” box. In the current implementation this has to be looked up in the global addressbook or the personal favorites list. Instead of having to explain who the user should contact with this question, or in the worst case just having to say NO, the conversation can be immediately transferred to the Chamber of Commerce.

Figure 6 shows a schematic view of the situation in the above example. In the middle is the KCC, which has chat buttons on eloket.groningen.nl and www.groningen.nl. Internally they can transfer conversations to one of their backoffice departments, where the more complicated questions are handled. Other organisations such as the public library, “het bedrijvenloket” and the healthcare organization MEE have placed their own chatbutton on various webpages. Questions concerning historical events could for example be transferred to the Al@din questionservice.

The Chatfone platform is built on opensource components and is using a LAMP architecture. In this case LAMP means Linux, Apache MySQL and PHP. When users log in, their

---

[3]https://eloket.groningen.nl
browser opens the Chat Contact Center (screenshot in Appendix [2]) which is the primary interface to Chatfone. In the background is a cluster of Apache web servers running on the Linux operating system. These cluster nodes serve the pages rendered by PHP, which gets data from a MySQL database server.
Figure 6: Schematic view of transfer options
5.2 Mid Office

The eMAXX Mid Office is a Java based integration platform originally targeted at the electronic government market. By connecting existing services it helps to enhance the service level and improve the electronic workflow within these organizations. This integration is realized by acting as an Enterprise Service Bus (ESB, see [Cha04]). The Enterprise Service Bus is a concept where different software services are connected through a common bus infrastructure. This lightweight infrastructure allows for a clean abstraction and reduces the need for individual point-to-point solutions.

Services are connected to the bus via XML (Extensible Markup Language) webservice or custom adapters. ETL (Extract, Transform and Load) options are also available for legacy systems. From these legacy systems data is periodically extracted, transformed into a suitable format and then loaded into an ESB connected system. The connected services are accessible for use by all processes on the bus, depending on the specified security parameters. A role based security model is in place to regulate access to sensitive data in great detail. Figure 7 shows the high level separation in layers which the Mid Office uses.

![Mid Office Layers](image)

The presentation layer holds servlets to generate a web interface. Which can consist of multiple websites for specific user groups. Figure 8 [Pan07] shows the Mid Office in its context. The “Digitaal loket” and “Medewerkers portaal” objects in Figure 8 are two examples of such sites. The vertical bars represent the different departments which have systems interacting with the Mid Office.

The processing layer in Figure 7 houses a collection of functional modules. These offer functionality for handling external authentication (e.g., DigiId[4] and OpenID[5]), mailings and working with additional personal information. A BPEL (Business Process Execution Language) engine is included for process orchestration, which can either be the Oracle BPEL engine or the opensource ActiveBPEL engine. The choice of engine is left to the customer.

The lowest bar in Figure 7 shows the data layer which houses storage systems, one of which is specifically based on the GFO-zaken specification (see [Pan04]). On the lowest level data is either stored in a PostgreSQL or Oracle database.

The included XML message broker is generic enough to interface many functional modules internally via JMS (Java Message Service) and XML. Externally this can communicate for example via HTTP and SOAP (Simple Object Access Protocol). This makes for a suitable and very flexible environment to develop services.

The SOAP protocol has been chosen for the connection between PHP and Java because this is by far the most suitable way to communicate in this context. Listing 1 shows an example

of the XML that can be sent to the Mid Office via the HTTP SOAP connector. The SOAP message starts with an envelope indicating the used XML namespaces and optional headers. The SOAP message body contains a requestProperties block which is generic and used for most requests to Mid Office. After that the extractTagsMessage block contains the request specific data, in this case new input text and a field for pre-parsed text that needs to be combined with the new text.
Listing 1: Example SOAP request

The Mid Office automatically generates service descriptions for all exported services in WSDL format (see Appendix C for a listing).

5.3 Lucene

Lucene is a Java library which adds text indexing and searching capabilities to applications. The first version of Lucene was written by Doug Cutting and released in March 2000. In September 2001 the project joined the Apache Foundation’s Jakarta family of high-quality open source Java products. After which the project has continued to grow [HG04].

The API (Application programming interface) offers two distinct feature sets. One for creating an index, another for searching through the index. Both can run independently from each other as can be seen in Figure 9 [HG04]. In the bottom is a common datastore which is used by both feature sets. Typically all data concerning an index is located in a single filesystem directory. Other storage options, such as in-memory or a database, are available in separate modules. Above the dotted line are the possible applications. One to gather data from various sources, the other to allow a users to searching through the index.

Lucene uses a combination of the Vector Space Model (VSM, described in [MRS] and originally published in [SWY75]) and the Boolean Model [BYRN99]. The Boolean Model is first used to narrow down the search based on the query specification, this is done by understanding ‘and’ and ‘or’ relations. Additionally the scope can be limited to that of a single organization or department in which to search. After the Boolean options have been applied, vectors are created from the resulting data. These vectors can then be used to look through the index and to calculate a score value. In most situations the highest scoring results will be returned to the user.

Lucene implementations come in all sizes, some of the larger known setups are Wikipedia.

6 http://lucene.apache.org
7 http://www.wikipedia.org
and IBM OmniFind. An Enterprise Content Management system like Alfresco integrates Lucene without the need for a user to be aware of it.

Figure 9: Lucene’s context [HG04]

Lucene has been loaded in the same runtime environment as the Mid Office and is used directly from the custom Java module (The “Tagging module” in Figure 3) that was created to integrate Lucene with Chatfone.

Section 7.2 describes how the Lucene index has been constructed.

---

8http://www.ibm.com/software/data/enterprise-search
9http://www.alfresco.com
6 Requirements

Before the project started a set of requirements was put together to define the general boundaries of the project. These were defined in discussion with the parties involved. The user interface aspects came mostly from Chatfone, while the more technical details were given in by eMAXX. The overall goal is to make the end results measurable.

Section 6.1 starts with the functional requirements, after which Section 6.2 continues with the quality requirements. These requirements will be evaluated in Chapter 10.

6.1 Functional requirements

1. The system has to online evaluate the incoming chat text and be ready to give a suggestion at any point during the conversation. This means that during the conversation, the incoming text already needs to be processed.

2. The operator has access to the search results from within the chat application. This should be accessible in a single click when the conversation panel is visible.

3. It should be visible in the presented search results what the online status of the presented entity is. This can be done using the existing status icons.

4. It should be possible to transfer the conversation to another operator from the presented search results. This can be presented using the existing transfer icon. In case the entity is not online and transfer is therefore not possible, no transfer icon should be shown.

5. After a conversation ends, a Chatfone user should be able to enhance his own knowledge profile. This can be done by weighing terms from the conversation.

6. With sufficient administration privileges, the Chatfone operator should be able to update the knowledge profile of his department and organization. This can be integrated with the existing preferences.

6.2 Quality requirements

1. Response to a search query from the Chatfone operator should be available within 500ms. This includes all steps taken from submission to the presentation on the operator’s screen.

2. Results for the search query from Quality requirement 1 should be relevant to the current chat. Although relevance is a highly subjective matter, a statistical method can be used to calculate a relevance score for each search result.

3. Results for the search query from Quality requirement 1 should be sorted by relevancy. The highest scoring results should be presented first.

4. The search interface within the Chatfone should be comprehensible to the average user. During development and testing this can be verified using a selected number of Chatfone users. When nearing completion this group should be expanded.
7 Detailed design

The following sections describe some of the choices that have been made during the design phase of this project.

Figure 10 is a detailed view of the process to prepare a conversation for transfer. It shows the flow inside the “Determine transfer destination” box from Figure 9. It starts by extracting the terms from the conversation, after which the initial search query is executed. From the presented results the user can make a choice or enter additional tags to alter the search query.

Figure 10: Estimate experts

Based on the problem statement in Chapter 4, elements of the desired solutions will be explained in the following sections. Section 7.1 describes the different entities relevant to this project. A description is given about the index that is build using existing searchengine components in Section 7.2. And Section 7.3 describes the two ways in which the system is trained.

7.1 Which entities to choose from?

Chatfone uses a natural separation in organizations, departments and department members. It is specifically allowed for a user to be member of multiple departments. This is done to allow using the same organizational structure as is common in many organizations. Each organization has the freedom to use this structure at their own discretion. As described
before in Section 4.1 an operator might want to transfer a conversation to someone else.

The most specific target for such a transfer would be a department member, identifying a unique person within a department. This is however not desirable for most organizations. When giving external operators this choice, the department abstraction is lost. For example if the specific person is out for lunch the routing path becomes a dead end. Besides, it is not always relevant to know the person's name one is transferring to. In most of the current transfers, only a department is targeted. Outside a given organization's addressbook, targeting a person is currently not even possible. The reason this is not allowed is that transferring to a specific person would bypass the departments routing priority.

Transferring to an organization could be useful for smaller organizations with one knowledge domain, or larger ones with a central frontoffice department. Transfers to these organizations could automatically be routed to the selected default department. The organization's administrative users can arbitrarily choose the department in the Chatfone management panel.

The most common option is to transfer to a department. This gives the best abstraction of an organization's knowledge domains. And also conforms to behavior in other communication media, making the experience more intuitive to the end user. Therefore departments should be the primary entities from which the operator can select a transfer destination. Only within the operators own organization will it be possible to view the individual users who are available for a department.

Behind the scenes there is one addition. Based on the sought after tags, the departments routing scheme could be influenced. When multiple operator's are online in the same line for a department the matching operator's score higher. And as such they would have a higher probability of getting the conversation routed to them.

7.2 Building an index

The main index used for this project is built using Lucene. As described before in Section 5.3, the Lucene library is integrated into the Mid Office. Using an exported webservice, new data can be fed into the index. This data should be analyzed and prepared for storage.

To make a comprehensive profile of a user, data from several sources is included in the index. Each entity that is used has an associated database field with profile information. Therefore the index could be rebuild from scratch if ever needed. This however takes considerable resources, which makes it unpractical for regular updates.

Some of the data fields already existed in the Chatfone application and were primarily used in StadNet. StadNet is a portal site, functioning as an addressbook for chatting organisations and their (potential) customers. A categorized list of organisations is displayed depending on the selected city. And gives an overview of the chatting organizations in that selected city. Depending on the organization the information stored in the datafields consists of a set of keywords or a description in natural language and can be input separately for the organization and each department. This can be done from the Chatfone management panel.

All text is however filtered to extract only essential data, this is the same method which will be used on conversation data. It does not make a difference if the user enters keywords or natural language. All data is collected in a document object where also specific field weights are specified. Lucene offers generic document objects which can be extended for this specific data structure. Along with the text data there are also numerical identifiers for the user,
department and organization. These identifiers reference the entities used within the Chatfone database. A combination of the user and department id is used as unique key to the index record. This unique key makes it possible to update or remove the index record without rebuilding the entire index from scratch.

7.3 Training

The index described above is trained in two ways, both designed to enhance the quality of the search results.

The first of these training methods is done offline and is the extension of the Lucene module DutchAnalyzer, which provides Dutch linguistic knowledge. After inheriting from the generic Analyzer module, specific Dutch properties for stemming and stopword removal, as discussed in Chapter 3, are added. The default analyzer contains a list of 33 English stop-words. While the Dutch analyzer has 101 stop-words by default, these are still very generic. For this project this list could be extended to contain words common to general conversation. This will be done by applying statistical analysis to a set of conversations and extracting non-discriminating terms. In Section 8.2 the Dutch stopword list is extended to form a DutchChatAnalyzer module. This Analyzer is used in two phases. When filling the index and to preprocess search queries.

The second method is more active and allows the user to enhance his own profile in real-time. A schematic view of this process can be seen in Figure 11. After a conversation ends the preprocessed list of tags is still available. If there is unprocessed conversation data left, then this will be processed and added in the “Extract tags” box. Once updated the user is shown a list of keyword proposals which is sorted by the automatically assigned weights. From this list the user can select keywords which should be associated with him. Also a weighing can be specified to indicate how strongly related the word is. The list of terms would be sorted to show only the most likely candidates first. Figure 12 shows a design for the supposed panel. Parts of this panel could be reused later to edit the users profile.

When saving the updated profile, the data should also be sent to Lucene. Therefore a call is made to the saveTags method from the TaggingService to update the index. Under normal circumstances this should be processed immediately but depending on the work load this could be scheduled for later processing.
Figure 11: Learning step after a conversation has ended

The conversation has ended.

Please consider the tags below to describe this conversation:

- [ ] Foo
- [x] Bar
- [x] Baz

Extra: Enter Text

Select all
Save
Skip

Mail transcript to: User@example.com

Figure 12: Page to evaluate the last conversation
8 Text processing

A search index needs input. In most circumstances, this comes in the form of textual keywords. In this project, the only available input is a conversation. Although the user could enter search terms by hand, this is not desirable. That is why feeding a large part of the conversation into the index holds promise, which is something that can happen automatically. This leads to large queries, with a large amount of insignificant text. Something that most search engines are not designed to handle. It is therefore useful to preprocess the conversation into a suitably sized query.

The following section starts with an analysis of the conversation data. Section 8.2 continues with this analysis to reduce the amount of text in a query. Section 8.3 discusses two additional methods which were investigated but turned out to be not suitable.

8.1 Text analysis

To understand the volume of text input in common conversations it was necessary to analyze a large number of real conversations.

For privacy reasons only the conversation meta-data is kept in the Chatfone database, the conversation itself is not stored beyond the session’s end. Although the Chatfone application does not store transcripts of the conversations its users have, it has the option of mailing a transcript after the conversation has ended. Access was granted to several thousands of conversations held with Chatfone employees in the period between January 2006 and August 2007.

Code was written to analyze every transcript for certain statistical features, which are used in the rest of this section. Intermediate results are saved into a database to allow incrementally adding data from more transcripts. This storage of intermediate results also makes it easier to reproduce the results and to include additional calculations.

By this test program, the text from the individual speakers was passed separately to the TaggingService for stemming and stop-word removal. The separation on speaker was based on the meta-data stored in the transcript. By comparing the in- and output of this process the effectiveness of the processing step could be measured. Table 2 shows the results of this test program using the default DutchAnalyzer. For text from the operator 56.17% remained. While from the visitor 57.07% remained. Overall that makes a reduction of 43.38%. Although this is a significant reduction in text volume, the expectation is that this can be improved.

<table>
<thead>
<tr>
<th>Property</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator words</td>
<td>67.36</td>
</tr>
<tr>
<td>Words remaining</td>
<td>38.49</td>
</tr>
<tr>
<td>Words remaining %</td>
<td>56.17</td>
</tr>
<tr>
<td>Visitor words</td>
<td>46.04</td>
</tr>
<tr>
<td>Words remaining</td>
<td>24.37</td>
</tr>
<tr>
<td>Words remaining %</td>
<td>57.07</td>
</tr>
</tbody>
</table>

Table 2: Text statistics with the default DutchAnalyzer

Table 3 shows a list of averages over the analyzed transcripts. It can be seen that most “spoken” text is coming from the operator’s (48.05%). This is mainly due to the fact that they often send standard prepared answers and that answering an open question in general takes more words then the question itself. The visitors account for 32.74% of the text.
The default Lucene analyzer for the Dutch language, used to generate Table 2, only comes with a list of 101 stopwords. Section 8.2 shows how this list is extended to form the basis of the DutchChatAnalyzer.

### 8.2 Cutting down on conversation size

There are several kinds of data that can be optimized away. These are system generated messages, layout tags, punctuation and non-discriminating terms.

The system generated messages can be easily extracted from the other data since the origin of a line is stored in the transcript data. These system generated messages can notify the user about certain changes or events that happen during the conversation. For example the notification that the conversation is being transferred. Generally these do not contribute to an answer the visitor is receiving. The on average 20.25 system words per transcript make up 19.2% of the overall text volume. This can be seen in Table 3.

Generally the visitor is the one mentioning most of the sought after terms. This knowledge can be used by boosting the weight of those terms. The operator’s text should however not be reduced too far since it can hold suggested synonyms and other valuable information. This boost can be given by appending a caret (\(^\wedge\)) and numeric boost value to certain terms. For example the words foo and bar can get a 50% boost relative to the rest of the query with the syntax “(foo bar)\(^\wedge\)1.5”. A value below 1 gives a negative boost, thus reducing the possible score when that term matches.

After the specific boosts are assigned the speaking parties name can be left out. Normally these are repeated for each line.

Characters that are not normally expected to be in a query do occur commonly in a conversation. This includes new-line characters, punctuation and HTML markup tags. These need to be removed before the text is ready for Lucene.

Based on the Analyzer code available in the Lucene project, a custom Lucene Analyzer was created. This module is doing stop-word removal on common greetings, conversation elements and other non-discriminating terms. By counting word frequencies, a large number of extra non-discriminating terms could be selected which are added to the default list of 101 Dutch words. Table 4 lists the top 15 of most frequent words with their frequency of occurrence in the test set.

This list of terms is distilled from 3128 chat transcripts. Terms shorter then 3 characters or terms included in the DutchAnalyzer’s stopword list are already excluded. Of these transcripts the words with the highest frequency were manually inspected and classified. The cut-off point was empirically chosen at a term frequency of 100. At that point the number of discriminating terms which had to be manually excluded became larger then desirable. This selected 300 terms, which accounts for around 35% of the frequency sum.

<table>
<thead>
<tr>
<th>Property</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversation duration in sec</td>
<td>463.08</td>
</tr>
<tr>
<td>Total number of words</td>
<td>133.65</td>
</tr>
<tr>
<td>Percentage system words</td>
<td>19.20</td>
</tr>
<tr>
<td>Percentage operator words</td>
<td>48.05</td>
</tr>
<tr>
<td>Percentage visitor words</td>
<td>32.74</td>
</tr>
</tbody>
</table>

Table 3: Text statistics from 3128 chat transcripts
Since all transcripts are in Dutch, this list was added to the list in DutchAnalyzer. Following the inheritance structure of Java, this became the subclass DutchChatAnalyzer.

After applying this new Analyzer, the text volume dropped significantly. For operator text the percentage remaining dropped another 40.82% to 15.35% and for visitor text it dropped 24.31% to 32.76%. Table 5 shows the results after the training step.

The difference between operator and visitor text can be attributed to the fact that the group of visitors is far more diverse. This diversity is visible in the sort of words that are used.

<table>
<thead>
<tr>
<th>Property</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator words</td>
<td>67.36</td>
</tr>
<tr>
<td>Words remaining</td>
<td>10.59</td>
</tr>
<tr>
<td>Words remaining %</td>
<td>15.35</td>
</tr>
<tr>
<td>Visitor words</td>
<td>46.04</td>
</tr>
<tr>
<td>Words remaining</td>
<td>13.55</td>
</tr>
<tr>
<td>Words remaining %</td>
<td>32.76</td>
</tr>
</tbody>
</table>

Table 5: Text statistics with DutchChatAnalyzer

8.3 Discussion

Not all methods that were attempted gave the desired result. This section therefore explains two methods that were explored but eventually discarded. The first method is shallow parsing the grammar, which was too slow for this application. Section 8.3.2 discusses query expansion, which broadens a search but yielded only limited improvements.
8.3.1 Shallow parsing grammar

The methods discussed thus far all work on individual words, while the context of a word also contributes to its meaning. Shallow parsing takes advantage of the grammatical information embedded in sentences and attempts to separate chunks of words with a specific meaning.

Individual words can be classified with part-of-speech tags, for example noun, verb or adjective.

Alpino is a collection of tools for parsing Dutch sentences. It can generate a dependency structure and a set of part-of-speech tags. Figure 13 shows a graphical representation of a tree structure which Alpino extracts from a sentence. This can be used to identify for example nouns and verbs which could receive a higher boost value.

![Graphical output of Alpino for: “Ik zoek een hotel in Groningen.”](image)

Figure 13: Graphical output of Alpino for: “Ik zoek een hotel in Groningen.”

This method has two problems. The first is the required processing time. A simple sentence like “Ik zoek een hotel in Groningen.” takes 0.5 seconds to parse (on a Intel Pentium 4 at 3.20GHz), initialization of the parser even adds another second. Experimentation shows that analysis takes up to 6 seconds for more complex sentences. This is obviously unacceptable for realtime analysis. Using the preprocessed query from the DutchChatAnalyzer in Alpino is not an option, as too much useful information is then lost. Although the speed is much better (0.6 sec) as opposed to using the whole query, the missing morphological information causes

11http://www.let.rug.nl/vannoord/alp/Alpino
misclassification. Although this could possibly be optimized in other ways, it is unlikely that it could be quick enough.

The second problem is its effectiveness. The results of parsing could be used to boost the weight of important terms (e.g. nouns and verbs). When combining this with stemming or stop-word removal, most of the remaining words receive a boost. This because nouns and verbs are the most common to remain. A short experiment with 10 sentences showed that about 90% of the remaining words were noun or verb.

This results in the conclusion that although Alpino could improve precision, the required processing time does not weight up to the improvement.

8.3.2 Query expansion

Query expansion, as described in Chapter 3, is a method of broadening a search. This would not improve precision, but could help improve recall. It works by adding related terms to the query, which would be gathered by using a source of synonyms.

One such source could be the redirect pages defined in Wikipedia. These redirects are used to guide visitors to the correct page when an alternative or incorrect spelling is used. Another source could be the EuroWordNet project.

In this context however a query is already relatively large. And since these large queries already pose a challenge, it would not be beneficial to use this technique. An option for future work could be expanding the stored knowledge profile at the time of index creation.

\[http://www.illc.uva.nl/EuroWordNet\]
9 Prototype Implementation

A prototype was developed to show the effectiveness of the proposed methods. Although the core search functionality is written in Java, most test and clientside code was written in PHP. Partly because of the ease of doing rapid development, but also because the main client (the Chatfone application) is written in PHP. The use of PHP made it a very close approximation of an actual implementation.

The process implemented in this chapter has already been shown in a flow diagram in Figure 10 (Chapter 7). Figure 14 shows a sequence diagram of the implementation made here. Starting from the left, the first role is that of Operator, which initiates a HTTP request to the Chatfone application. After processing the Chatfone sends a SOAP request to the Mid Office, which in turn makes a Java call to Lucene. In the right most column Lucene is shown, which receives the direct Java call from the Mid Office. The resulting data is passed back through the chain of roles in reverse order. The SOAP encoded result is formatted as HTML by the Chatfone application for presentation in the operator's browser.

![Sequence diagram of a typical request](image)

Figure 14: Sequence diagram of a typical request

The rest of this chapter is organized as follows: Sections 9.1, 9.2 and 9.3 follow the sequence of a normal search request. Section 9.4 explains how the online training is implemented.

9.1 Preprocessing

The internal format in which conversations are stored resembles HTML. For instance paragraph tags are used to separate individual messages and based on the class attribute, it can determined who said what. This format adds useful information, but also poses a problem to the query parser. For example punctuation or non-matching braces interfere with the Lucene query syntax. Therefore formatting is removed, links reduced to there title and times-

---

13 Lucene query syntax specification: http://lucene.apache.org/java/docs/queryparsersyntax.html

---
tamps are filtered out. Specific weights are applied to messages from different origins. The question that was entered in the optional pre-chat window is given an even higher boost then the normal conversation text from the visitor.

Then the `extractTags` method of the `TaggingService` is called to optimize the query. Here the Dutch stemming or stop-word removal is applied. Doing this in a preprocessing step saves resources for longer conversations. Even more importantly, it saves time when the search results are actually requested.

After the new input has been processed a new pointer is stored to the position in the conversation up to where the text has been processed. This pointer is stored in the users session, which also houses the preprocessed results.

When the preprocessing step is done depends on configuration. Currently it is done after every five messages sent by the visitor as to keep the request small and at the same time to keep up with the conversation. When the conversation finishes the remaining preprocessing is combined with the call to `extractTags`.

### 9.2 Query execution

When the user opens the search tab the preprocessed query is retrieved from the users session store. And this query is then sent to the eMAXX Mid Office for execution by Lucene.

From the WSDL specification exported by the Mid Office, a PHP stub class is generated (with wsdl2php\[14\]). This WSDL offers all the methods and the used datastructures from the `TaggingService`. This includes the following SOAP methods:

- `tagging:extractTags` - To preprocess text
- `tagging:findTaggable` - To search through the index
- `tagging:saveTags` - To update the index

The `TaggingService` stub object packages the request parameters in a `findTaggableRequestMessage` (see Listing \[2\]) and facilitates the SOAP transport. The effect of stemming can be observed (“verhuiz” instead of “verhuizen”) in the querystring element in Listing \[2\].

After arriving in the Mid Office tagging module and after the XML request has been properly parsed and validated, a Lucene query object is constructed. If a filter parameter for a certain organisation or department is included in the request, then this is added as an obligatory condition in the query object. Thereafter a set of department members is selected from the index. Members from departments that are flagged as “internal only” are removed for operators from other organizations.

After returning to PHP, the results are further processed. A grouping is made by department and a department score is calculated. This grouping is done because the department was chosen as the primary entity to choose from. The department score is calculated by the average score from all matched users in that department (Equation \[3\]).

\[
\text{score}_{\text{total}} = \frac{\sum_{\text{department members}} \text{score}}{\text{user count}}
\]  

(3)

Also some additional information is retrieved from the Chatfone database. This includes the online status of the individual departments.

\[14\]http://www.sourceforge.net/projects/wsdl2php

HERMAN A. VAN RINK

University of Groningen
9.3 Result presentation

Alongside each chat panel, the user has several connected panels with extra information and options related to that conversation. In one of these related panels the search results are presented to the user.

On the server side there is a template within the Chatfone application that is responsible for generating the user interface. This template iterates over the prepared results and shows them in a HTML page. Figure 15 shows a screenshot of the rendered page with some debug messages included (see Appendix B for a larger version).

![Screenshot of tagging-beta chat side tab](image)

Figure 15: Screenshot of tagging-beta chat side tab

Each row of the table shows the organization and department name. Just before the names there is an icon indicating the online status of that particular department. It has been considered to hide departments which are offline, since transferring to them is not possible. One opinion is that they are wasted screenspace, however experiences with users have shown...
it to be sometimes confusing. Even though a missing name is easily explained by the fact that no users are online for the department, this is enough to cause confusion for some users. In the advanced options a checkbox is included to remove offline departments from the results.

Behind the names is a clickable icon that transfers the conversation to the department. When this is done, the results are replaced by a message confirming or in case of an error denying, that the conversation was handed over.

Specifying the query is done using the form fill in interaction style as described by [Shn97] and [BYRN99]. Above the results is an input field which can be used to enter extra keywords. The extracted query is normally not displayed, but it is available in the advanced options. Also some extra options that give the user more control over the query are available in the advanced search dialog. Figure 16 shows the advanced interface.

Figure 16: Screenshot of tagging-beta chat side tab in advanced mode

The visitor would, after transferring, see something like the window shown in Figure 17. The figure shows a fictional conversation which has just been transferred and accepted by the new operator. In order to avoid a scrollbar, the text has been kept short.
Figure 17: Screenshot of the visitor window
Post conversation training

Section 7.3 described what should happen after a conversation has ended. The current section will explain the implementation of this design. Figure 18 shows part of the window which can be presented to the operator after a conversation ends. It offers to extend the personal profile with tags from the conversation. The optimized Lucene query that remains after the conversation contains a relatively simple syntax to specify boost values. A regular expression as seen below in Equation 4 is used to select all terms and their specific weight.

This regular expression captures (groups of) words that are surrounded by parenthesis and followed by a boosting value. After removing these weighted terms from the query string, all that remains are the neutral weighted terms. These can then be added to the associative array in which the term weights are stored. After sorting by weight the top most of these are presented to the user.

$$/\((\{\}\{\}\)*\)\^\((d + (? : \:\:d))\)/$$  

(4)

Figure 18: Page to evaluate the last conversation

The user sees a set of three radio buttons behind each word. Selecting one of these indicates it should be added to the user’s profile. Words that are already in the user’s profile are preselected, such that their weight can be changed. The right most radiobutton (n.v.t.) can be selected if a word is considered non-relevant.

After submission to the Chatfone server, the three separate fields are updated and committed to the Chatfone database. After which the TaggingService is called to update the index for that department user. In the TaggingService the same Java code is called that was used to build the Lucene index (see Section 7.2).
10 Experimental evaluation

This chapter describes the steps taken to evaluate the developed prototype. Section 10.1 starts by describing the environment in which all tests have been performed. Section 10.2 identifies the different metrics used for this evaluation. The test results are presented in Section 10.3 after that some discussion is included in Section 10.4.

10.1 Setup

The prototype described in Chapter 9 has been developed and tested on the testing infrastructure that was setup for Chatfone. The Chatfone application is running in its own VMware machine, which houses the database and a single webserver node. A second VMware machine houses the Mid Office in its JBoss runtime environment and the PostgreSQL database. Although this setup has obvious performance drawbacks it should be able to give a good simulation of production properties. Both virtual machines are running on a VMware server with a Core 2 duo 2.13 GHz and 4GB of internal memory, of which each VM uses 512MB. The client in this evaluation is a Pentium 4 workstation connected over a local network to rule out traffic congestion on the internet.

In production, Chatfone is balanced over multiple machines. These spend most of their resources on keeping all connected clients up to date. Together with the current number of requests, the impact of a single search request is negligible. If for every conversation two search requests would be performed, the total number of requests would increase only 0.09% (Based on the Chatfone webserver statistics of November 2007).

10.2 Measures

The measures presented here are the reduction in text volume and the average response times.

The reduction in text volume is a measure of the effectiveness of the processing steps. This is measured in the percentage of words that remain in the query relative to the amount of “spoken” words. A new batch of conversation transcripts was used to verify the effectiveness of the DutchChatAnalyzer training. This new set contains 585 more recent texts, taken from the same departments as the training batch.

Another measure is the average response time, for which five scenarios have been tested. To users, only the response time of the entire chain of components is noticeable. Therefore the first two tests are run against the Chatfone application. To simulate a real user-session, test code was written that provides the required session data and text input in the regular transcript format.

The Apache utility ab (Apache bench) is useful for executing this kind of test, it is one of the most commonly used tools for this purpose. In every run the target URL is called 100 times. This is first done in a single thread, then in two threads and finally in four concurrent threads.

To get a more clear view of the performance, the internal SOAP services have also been tested separately. This makes up the other three scenarios. The difference between the full chain and only the back-end components is the time the Chatfone application requires to do it’s processing.
10.3 Results

This section lists the results for the above specified measures. Table 6 shows the effectiveness of the DutchChatAnalyzer on the verification batch. It can be seen that the percentage of remaining operator and visitor words is slightly higher than on the training batch. For visitors and operators combined the reduction is 75.5%.

<table>
<thead>
<tr>
<th>Property</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator words</td>
<td>59.26</td>
</tr>
<tr>
<td>Words remaining</td>
<td>9.08</td>
</tr>
<tr>
<td>Words remaining %</td>
<td>15.40</td>
</tr>
<tr>
<td>Visitor words</td>
<td>43.74</td>
</tr>
<tr>
<td>Words remaining</td>
<td>13.32</td>
</tr>
<tr>
<td>Words remaining %</td>
<td>33.69</td>
</tr>
</tbody>
</table>

Table 6: Text statistics with DutchChatAnalyzer - verification batch

Table 7 shows the response times measured by ab. These are shown in milliseconds and indicate the total time required to do a request. For each service the different concurrency levels are shown in separate rows, indicated in the Threads column. Min, Mean and Max hold the average responsetimes, while Total time indicates the time in seconds required to execute all 100 requests.

<table>
<thead>
<tr>
<th>Property</th>
<th>Threads</th>
<th>Min</th>
<th>Mean</th>
<th>Max</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>extractTags via PHP</td>
<td>1</td>
<td>42</td>
<td>87</td>
<td>229</td>
<td>8.77</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>56</td>
<td>111</td>
<td>367</td>
<td>5.61</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>84</td>
<td>194</td>
<td>269</td>
<td>4.96</td>
</tr>
<tr>
<td>findTaggable via PHP</td>
<td>1</td>
<td>78</td>
<td>92</td>
<td>204</td>
<td>9.32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>109</td>
<td>161</td>
<td>309</td>
<td>8.12</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>146</td>
<td>230</td>
<td>397</td>
<td>5.86</td>
</tr>
<tr>
<td>findTaggable</td>
<td>1</td>
<td>30</td>
<td>78</td>
<td>368</td>
<td>7.94</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>29</td>
<td>148</td>
<td>327</td>
<td>7.49</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>29</td>
<td>212</td>
<td>477</td>
<td>5.41</td>
</tr>
<tr>
<td>extractTags</td>
<td>1</td>
<td>10</td>
<td>55</td>
<td>182</td>
<td>5.61</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
<td>104</td>
<td>482</td>
<td>5.25</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>24</td>
<td>214</td>
<td>433</td>
<td>5.51</td>
</tr>
<tr>
<td>saveTags</td>
<td>1</td>
<td>12</td>
<td>81</td>
<td>261</td>
<td>8.21</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12</td>
<td>130</td>
<td>1061</td>
<td>6.53</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>36</td>
<td>269</td>
<td>2115</td>
<td>7.23</td>
</tr>
</tbody>
</table>

Table 7: Apache bench responsetime statistics

10.4 Discussion

The statistics shown in Table 6 are similar to results gained on the training data with the DutchChatAnalyzer in Section 8.2. This confirms the significant reduction in the amount of input for the Lucene parser seen above.

The effectiveness of training in this situation is however still limited due to the large business domain, the wide variety of subjects and the large variation in vocabulary habits. If this were not the case, or if the solution could be trained per business domain, the reduction could be even greater.
The ab statistics show that the response time is within limits. The only abnormality comes up when the saveTags method is called in multiple concurrent threads. This was expected and is due to the exclusive lock Lucene requires before writing to the index. When taking too long these update requests should therefore be done in the background as described in Section 7.3.

In these tests, the saveTags method is called with a relatively small text to store. In production these will be somewhat larger, but this only adds marginal amounts of time. This is confirmed when looking at the extractTags performance, which is computationally the most intensive element of saveTags.

The above results together with Table 8 show that all of the requirements specified in Chapter 6 have been met. In Table 8 the requirement codes match the numbers of the functional and quality requirements in Chapter 6.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>✓</td>
<td>Text is preprocessed incrementally during the conversation, therefore the reaction can be seemingly instant.</td>
</tr>
<tr>
<td>F2</td>
<td>✓</td>
<td>The transfer tab is permanently visible when the chat is active.</td>
</tr>
<tr>
<td>F3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F4</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F5</td>
<td>✓</td>
<td>A suggestion is done to add the most promising terms to the users profile. A weight must be specified for these terms</td>
</tr>
<tr>
<td>F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Q1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Q2</td>
<td>✓</td>
<td>The score value provided by Lucene is used to calculate a relevancy score as demonstrated in Equation 8</td>
</tr>
<tr>
<td>Q3</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Requirements - evaluated
11 Conclusions & Future work

This thesis focuses on finding the most suitable person within the chat network of Chatfone to answer a specific question. In particular the knowledge that a users has is formalized in a digital profile which can be used to match that user to an ongoing conversation or to an incoming question. The next section draws conclusions and thereafter Section 11.2 discusses options for future work.

11.1 Conclusions

It has been shown that conversation routing behaves in a similar fashion on the compared media. Voice conversations over a telephone line, Email and Chat. From these media only telephone shows a major difference in the way it needs to be processed. The somewhat different approach to searchengine technology that is presented in this thesis has proved to be effective. The delivered suggestions help the user in transferring a conversation to the most suitable person. To produce these suggestions based on the current conversation, the text had to be brought down in volume.

The average reduction for a conversation has been measured at 75.5%. The remaining text (24.5%) consists of a parsed and weighted query. This query is ready to be executed by the Lucene search engine. Translation of conversation text to a set of keywords is for the most part hidden from the user. No user involvement is required and only if specifically desired, can the user manipulate the extracted keywords from the advanced search form.

Executing this query leads to relevant results and will help in lowering the barrier to successfully transfer a conversation. The user gets suggestions without having to think about the way these are gathered.

It can therefore be expected that, when this solution is in production, the number of transferred conversations will increase. Thereby strengthening the networks extra value and increasing the customer satisfaction.

11.2 Future work

In future development several steps could be taken to improve the here presented solution. These build on the developed solution and increase its accuracy and speed.

First, the index built with Lucene could be enhanced with data from the existing Chatfone knowledgebase. The articles in this knowledgebase range from short descriptions to thorough explanations of various topics. The author of an article could be used as extra hint as to the persons knowledge. Besides using the authorship relation, articles could also be offered along with the search results as alternative to transferring.

A second enhancement could come from the storage method used by Lucene. It currently supports boost values per document and per field. Marking specific terms with a boost value could improve precision. This would make it possible to adjust term weights after each conversation.

The third option is using data from the EuroWordNet project, or some other source of linguistic information, which could help improve the precision and recall. This data could be used to expand knowledge profiles with synonyms at the time of index creation.

Finally, a more sophisticated graphical user interface could assist in reweighing the extracted query terms and thereby improving the users knowledge profile.
References


A XML schema of SOAP requests

Listing 3: FindTaggable Requests XSD

Listing 4: saveTags Request XSD
Listing 5: extractTags Request XSD
B  Chat Contact Center

Figure 19: Screenshot of tagging-beta chat side tab - large
C WSDL for TaggingService

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://www.emaxx.org/msg/tagging"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    xmlns:mosoap="http://www.emaxx.org/core/soap"
    xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:tagging="http://www.emaxx.org/msg/tagging"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <wsdl:types>
        <schema attributeFormDefault="unqualified"
            elementFormDefault="qualified"
            targetNamespace="http://www.emaxx.org/msg/tagging"
            xmlns="http://www.w3.org/2001/XMLSchema">
            <xs:import namespace="http://www.emaxx.org/core/soap"/>
            <xs:complexType name="findTaggableResponseItem">
                <xs:sequence>
                    <xs:element name="type" type="xs:string"/>
                    <xs:element name="id" type="xs:string"/>
                    <xs:element name="name" type="xs:string"/>
                    <xs:element name="deptname" type="xs:string"/>
                    <xs:element name="company" type="xs:string"/>
                    <xs:element name="score" type="xs:float"/>
                    <xs:element name="userId" type="xs:int"/>
                    <xs:element name="deptID" type="xs:int"/>
                    <xs:element name="aspID" type="xs:int"/>
                    <xs:element name="visible" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </schema>
    </wsdl:types>
</wsdl:definitions>
```

HERMAN A. VAN RINK

University of Groningen
C  WSDL FOR TAGGINGSERVICE

</xs:complexType>
</xs:element>
<xs:element name="extractTagsResponseMessage">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="taglist" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="findTaggable">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="mosoap:requestProperties" />
      <xs:element ref="tagging:findTaggableMessage" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="findTaggableMessage">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="deptID" type="xs:int" />
      <xs:element name="aspID" type="xs:int" />
      <xs:element name="own_aspID" type="xs:int" />
      <xs:element name="querystring" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="findTaggableResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="mosoap:responseProperties" />
      <xs:element ref="tagging:findTaggableResponseMessage" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="findTaggableResponseMessage">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="querystring" type="xs:string" />
      <xs:element name="results">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="result" type="tagging:findTaggableResponseItem" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="saveTags">
  <xs:complexType>
    </xs:complexType>
</xs:element>
<xs:sequence>
  <xs:element ref="mosoap:requestProperties" />
  <xs:element ref="tagging:saveTagsMessage" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="saveTagsMessage">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="userID" type="xs:int" />
      <xs:element name="deptID" type="xs:int" />
      <xs:element name="aspID" type="xs:int" />
      <xs:element name="visible" type="xs:int" />
      <xs:element name="fullname" type="xs:string" />
      <xs:element name="deptname" type="xs:string" />
      <xs:element name="company" type="xs:string" />
      <xs:element name="tags" type="xs:string" />
      <xs:element name="tags_light" type="xs:string" />
      <xs:element name="tags_normal" type="xs:string" />
      <xs:element name="tags_heavy" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="saveTagsResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="mosoap:responseProperties" />
      <xs:element ref="tagging:extractTagsResponseMessage" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</schema>

<xs:element name="faultDetails">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="source" nillable="true" type="xs:string" />
      <xs:element name="code" nillable="true" type="xs:string" />
      <xs:element name="message" nillable="true" type="xs:string" />
      <xs:element name="stacktrace" nillable="true" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="requestProperties">
  <xs:complexType>
    <xs:sequence>
    </xs:sequence>
</xs:element>
<xs:element name="user" nillable="true" type="xs:string"/>
<xs:element name="password" nillable="true" type="xs:string"/>
<xs:element name="domain" nillable="true" type="xs:string"/>
<xs:element name="correlationId" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="responseProperties">
<xs:complexType>
<xs:sequence>
<xs:element name="correlationId" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</schema>
<wsdl:messages>
<wsdl:message name="extractTagsFault">
<wsdl:part element="mosoap:faultDetails" name="error"/>
</wsdl:message>
<wsdl:message name="extractTagsRequest">
<wsdl:part element="tagging:extractTags" name="parameters"/>
</wsdl:message>
<wsdl:message name="extractTagsResponse">
<wsdl:part element="tagging:extractTagsResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="findTaggableFault">
<wsdl:part element="mosoap:faultDetails" name="error"/>
</wsdl:message>
<wsdl:message name="findTaggableRequest">
<wsdl:part element="tagging:findTaggable" name="parameters"/>
</wsdl:message>
<wsdl:message name="findTaggableResponse">
<wsdl:part element="tagging:findTaggableResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveTagsFault">
<wsdl:part element="mosoap:faultDetails" name="error"/>
</wsdl:message>
<wsdl:message name="saveTagsRequest">
<wsdl:part element="tagging:saveTags" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveTagsResponse">
<wsdl:part element="tagging:saveTagsResponse" name="parameters"/>
</wsdl:message>
</wsdl:messages>
<wsdl:portTypes>
<wsdl:portType name="TaggingPortType">
<wsdl:operation name="extractTags">
<wsdl:input message="tagging:extractTagsRequest" name="extractTagsRequest"/>
</wsdl:operation>
</wsdl:portType>
</wsdl:portTypes>
<wsdl:operation name="findTaggable">
  <wsdl:input message="tagging:findTaggableRequest" name="findTaggableRequest" />
  <wsdl:output message="tagging:findTaggableResponse" name="findTaggableResponse" />
  <wsdl:fault message="tagging:findTaggableFault" name="findTaggableFault" />
</wsdl:operation>

<wsdl:operation name="saveTags">
  <wsdl:input message="tagging:saveTagsRequest" name="saveTagsRequest" />
  <wsdl:output message="tagging:saveTagsResponse" name="saveTagsResponse" />
  <wsdl:fault message="tagging:saveTagsFault" name="saveTagsFault" />
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="TaggingBinding" type="tagging:TaggingPortType">
<soapbind:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="extractTags">
  <soapbind:operation soapAction="tagging:extractTags" />
  <wsdl:input name="extractTagsRequest">
    <soapbind:body use="literal" />
  </wsdl:input>
  <wsdl:output name="extractTagsResponse">
    <soapbind:body use="literal" />
  </wsdl:output>
  <wsdl:fault name="extractTagsFault">
    <soapbind:fault name="extractTagsFault" use="literal" />
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="findTaggable">
  <soapbind:operation soapAction="tagging:findTaggable" />
  <wsdl:input name="findTaggableRequest">
    <soapbind:body use="literal" />
  </wsdl:input>
  <wsdl:output name="findTaggableResponse">
    <soapbind:body use="literal" />
  </wsdl:output>
  <wsdl:fault name="findTaggableFault">
    <soapbind:fault name="findTaggableFault" use="literal" />
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="saveTags">
  <soapbind:operation soapAction="tagging:saveTags" />
  <wsdl:input name="saveTagsRequest">
    <soapbind:body use="literal" />
  </wsdl:input>
</wsdl:operation>
Listing 6: TaggingService WSDL