The Implementation and Evaluation of a Speech Recognition Component for a Meeting Browser

Gerard Lynch
B.A. (Mod.) CSLL
Final Year Project, May 2006
Supervisor: Dr. Saturnino Luz
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

May 8, 2006
Gerard Lynch
Acknowledgments

I would like to thank Dr Saturnino Luz for his diligent supervision and helpful input during the project.

Many thanks are also due to Mr Matt Mouley-Bouamrane who was of great assistance with many technical aspects of the project throughout the year.

Thanks to Niamh for her support and eternal optimism.

Many thanks to Dr Carl Vogel, for without his dedication and steadfastness in the face of adversity, there would be no CSLL degree course as we know it.

A final note of thanks is also due to my parents, who despite their general fear and trepidation in regard to the field of technology, were always supportive of my endeavours.
'It is impossible to speak in such a way that you cannot be misunderstood'.

Sir Karl Popper, Austro-English philosopher of science (1902 - 1994)
## Contents

1 Introduction 1
   1.1 Background .............................................. 2
   1.2 Data Acquisition ....................................... 2
   1.3 Processing ............................................. 3
   1.4 Tools .................................................. 4

2 Meeting browsing and ASR 6
   2.1 Background .............................................. 7
   2.2 Systems ................................................ 7

3 System Overview 9
   3.1 The Recognizer ......................................... 10
      3.1.1 Background ........................................ 10
      3.1.2 Package .......................................... 10
      3.1.3 Foundations ..................................... 11
   3.2 Expansion ............................................. 12
      3.2.1 Sampled Audio Processing ......................... 12
      3.2.2 Basic Outline: Transcriber Algorithm .......... 13
      3.2.3 Explanation .................................... 14
      3.2.4 Improving the Output ............................ 15
      3.2.5 Word Lattice ..................................... 16
      3.2.6 Reorganizing the Result ......................... 16
      3.2.7 Front End ........................................ 17
   3.3 Integration ............................................ 18
      3.3.1 The Meeting Miner ............................... 19
      3.3.2 Post Processing ................................ 19
      3.3.3 Displaying the Output ......................... 20
3.4 Conclusion ......................................................... 20

4 Performance and Implementation Issues ......................... 22
  4.1 Introduction ..................................................... 23
  4.2 Installation Issues ............................................ 23
  4.3 Performance Issues ........................................... 23
    4.3.1 Word Lattice ............................................. 24
  4.4 Offset option .................................................. 25
  4.5 Issues with Sphinx system .................................. 25
  4.6 Conclusion ..................................................... 25

5 Evaluation ................................................................... 27
  5.1 Introduction ..................................................... 28
  5.2 Part One .......................................................... 28
    5.2.1 Word Error Rate ........................................... 28
    5.2.2 The Participants ........................................... 29
    5.2.3 Analysis ..................................................... 29
    5.2.4 Further Considerations .................................. 30
    5.2.5 Keyword Spotting ......................................... 31
  5.3 Part Two .......................................................... 33
    5.3.1 The Usability Study ....................................... 33
    5.3.2 The Questions ............................................... 33
    5.3.3 Analysis of Results ....................................... 34
    5.3.4 Questionnaire .............................................. 34
    5.3.5 Summary ..................................................... 35

6 Conclusion .................................................................. 37
  6.1 Summary .......................................................... 38
  6.2 Future Work ...................................................... 39
    6.2.1 Further Evaluation ......................................... 39
    6.2.2 Interface Improvement .................................... 39
    6.2.3 Improvement of Speech Recognition .................. 40

Bibliography .................................................................. 41

A Transcriber Code ..................................................... 42
## List of Figures

1.1  Sphinx 4 running as a project within Eclipse  

3.1  Hidden Markov model for the words one and two from: Sphinx Website  

3.2  Basic Transcriber Output  

3.3  Graphical Representation of a Word Lattice  

3.4  Current Transcriber Output  

3.5  Transcriber frontend  

3.6  Meeting Miner with integrated speech recognition  

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Sphinx 4 running as a project within Eclipse</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Hidden Markov model for the words one and two from: Sphinx Website</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Basic Transcriber Output</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>Graphical Representation of a Word Lattice</td>
<td>16</td>
</tr>
<tr>
<td>3.4</td>
<td>Current Transcriber Output</td>
<td>18</td>
</tr>
<tr>
<td>3.5</td>
<td>Transcriber frontend</td>
<td>19</td>
</tr>
<tr>
<td>3.6</td>
<td>Meeting Miner with integrated speech recognition</td>
<td>21</td>
</tr>
</tbody>
</table>
List of Tables

5.1  Keyword Spotting Accuracy Table ....................... 32
5.2  Usability Quiz Results ................................. 34
Abstract

Automatic speech recognition is a technology which has a myriad of real-world applications. From automatic answering machine services, to directory enquiries and automated delivery services, anywhere where humans interface with an autonomous system, speech recognition provides the link between man and machine. This project applies speech recognition to the field of meeting browsing, automating many of the tasks undertaken by the minute taker. Using an open source speech recognizer system, individual speaker recordings are processed and timestamped. The transcriptions are then processed and integrated into a meeting browser system, in order to facilitate the quick and easy access of meeting information.
Chapter 1

Introduction
1.1 Background

The field of LVR (Large Vocabulary Recognition) is an important part of speech recognition as an area of research in computer science. In recent times, with the development of ever faster processors and larger hard-drives, the computing overheads needed for these systems are no longer an issue in their implementation. Even with these developments in computing, speaker independent large vocabulary recognition systems are still far from a being a perfect solution for such tasks such as dictation, which still must be trained to an individual speaker's voice in order to produce usable output. There will always be OOV\(^1\) words in large vocabulary speech recognition because of the fact that the recognizer dictionary contains a finite number of entries.

There are many factors which make LVR a difficult task. Obviously the great difference in speaker's voices and accents is one aspect which must be considered. Another challenge is the idea of spontaneous speech, ie. everyday conversation which does not comply to fixed grammatical rules, speech containing 'false starts' where a speaker begins a word and does not finish it, and the many interjections and sounds which do not correspond to recognizable parts of speech. There is also the issue of task specific vocabularies, for example a speech recognizer trained on a legal corpus may not perform particularly well given data which has been culled from broadcast news. Coupled with cross talking and interruptions, the task of large vocabulary speech recognition is certainly a non trivial one. A more detailed account of the issues faced with in LVR is given by (Young, 1998) and (Bouamrane & Luz, 2006a).

1.2 Data Acquisition

This project is concerned with the recognition of spontaneous non-rehearsed speech, specifically focused on a meeting situation. The test meetings were recorded using the Shared Editor, known as RECOLED\(^2\) developed by Saturnino Luz and Matt Mouley Bouamrane at the Department of Computer Science, Trinity College Dublin in conjunction with David King and Masood Masoodian at the Computer Science Department of the University of

\(^{1}\)Out of Vocabulary, words which don’t occur in the recognizer dictionary  
\(^{2}\)REcording COllaborative EDitor (Masoodian, Luz, Bouamrane, & King, 2005)
Waikato, Hamilton, New Zealand.

This program facilitates remote meetings via a network connection and provides an audio conferencing feature, a shared text area which may be edited by both participants, and a back end which stores a record of when text was edited, by whom it was edited and a record of when text was pointed to and/or and circled using the virtual pointing and circling tools implemented in the interface.

1.3 Processing

The shared audio stream was recorded using the RTP (Real Time Transport Protocol) for streaming audio. In addition to this stream, which was deemed to be of too low quality for recognition purposes, each speaker wore a small clip lapel microphone which was connected to a portable hard disk recorder. This provided a high quality recording of each speaker, which was used as the recognizer input. The lower quality stream is used in the meeting browser interface as a recording of the meeting.

The recordings were carried out in the postgraduate networking laboratory in Oriel House, Westmoreland St, Dublin. Each speaker wore a headset microphone. Due to the fact that both participants were in the same room, it is possible to hear the other participant talking on the individual microphone recordings. There was some background traffic noise which is also audible on the recording.

After the meeting, the audio files were prepared to be processed by the recognizer. One drawback of the recognizer used in the project is that the user must specify the sample rate and endian-ness of the audio data used. The sample rate refers to the number of samples per second, and the endian-ness corresponds to the file format of the audio, with Microsoft’s wav format being little-endian. The issue of audio file processing will be dealt with more thoroughly in a later chapter.

Once the audio is of a suitable format, it is then processed by the recognizer program. The recognizer takes the audio as input and returns timestamped transcriptions of the audio data. This timestamped data is then input into the Meeting Miner, which is a meeting browser system created by M. Mouley Bouamrane at TCD. This system allows the user to browse the meetings which were recorded using the Shared Editor. The speech recogni-
tion output is used to augment the data which has been extracted from the
Shared Editor.

1.4 Tools

The programming language used in the project was Java. The IBM Eclipse
IDE was used to build and manage the speech recognition package, the CMU
Sphinx 4 Recognizer, developed at Carnegie Mellon University in conjunction
with Sun Microsystems and Mitsubushi Research Labs.

One of the requirements of the program was the installation of Apache
Ant to compile the Sphinx package. Apache Ant is a program for creating
XML build files for Java, and also integrates fully into the Eclipse IDE.
With Sphinx being a complicated package containing many separate files
and folders, Ant enabled one click compiling using the build files that were
provided with the Sphinx source code distribution.

Elements of the platform utilise the Java Speech API, and this was pro-
vided with the source code. The JSAPI is a Java package for use with speech
recognition systems and speech synthesis systems, and is not provided with
the Standard Edition of the Java platform.

Using Eclipse provided a stable platform to build and run the Sphinx
project. It was possible to edit all input parameters and a debugger was
available when things went wrong. The Sphinx website provided instructions
on how to get everything up and running using Eclipse.

The Sphinx system was chosen because of the fact that it was imple-
mented in Java, and also because it was an open source system and therefore
free from licensing issues. The flexible nature of the system, and the fact that
it was configurable to suit a range of different tasks also proved beneficial to
the project.

This report details the implementation, integration and the evaluation
of a speech recognition system. Chapter 2 deals with current advances in
the field of meeting browsing, with Chapter 3 containing the step by step
guide to the implementation and integration of the component. Chapter 4
gives a run down of different technical issues encountered during the project,
and Chapter 5 deals with the evaluation of the system, both of the output
in terms of word error rates and word spotting accuracy, and from a user’s
perspective, with a small usability study. Chapter 6 provides a number of
concluding remarks along with suggestions for improvement and for future work.
Chapter 2

Meeting browsing and ASR
2.1 Background

With the rise of modern video and audio technology, the era of the human minute taker carefully transcribing the agenda and points of reference of a meeting could be soon coming to a close. The modern meeting room may contain an array of microphones and video cameras coupled with a digital projector and computer for displaying presentations.

Having this added information at one’s disposal allows for the off-line reviewing of meetings, with a possible view towards the automation of certain tasks such as topic or speaker segmentation.

Also, there is a need for the efficient browsing of multi-modal meeting recordings, and thus there are a number of different meeting browser systems available, all providing different interfaces and implementations.

2.2 Systems

In their review of the state-of-the-art of meeting browser systems, Luz and Bouamrane (Bouamrane & Luz, 2006a) mention a number of meeting browser systems which use ASR transcripts to aid the browsing of meetings. The FERRET meeting browser provides a speech recognizer transcript as part of the multi-modal interface, which includes video and audio feeds. The transcript is designed to be a rough guide to the meeting and is not expected to provide a word perfect version of the meeting audio stream.

The MeetingBrowser system also uses ASR in an attempt to summarize the main points of a meeting. The summarizer takes the transcript of the meeting as input and attempts to determine relevant parts of the audio, and generate for the user a rough overview of the meeting itself.

In the majority of these systems, the ASR transcript is used as a rough reference for the user. However in the system devised by Rogina and Schaaf (Rogina & Schaaf, 2002), the speech recognizer is adapted ‘on the fly’ using words found in the text of the presentation slides. The project is concerned with the development of an intelligent meeting room system. Information in the form of key words is extracted from the presentation slides, and this is then analyzed and used to aid the speech recognizer. The speech recognizer in this case has a language model which can be easily changed by adding or subtracting certain modules dependent on the subject matter. The recognizer
also works by recognizing the speech of the speaker in real time, and uses the results to provide certain relevant information, using a web search on the topic, which may aid the listeners.

In their experiments, there was a considerable improvement in the word error rate of the recognizer, when it was fed additional information about the topic. This system was aimed toward lectures and presentations, which would normally have a more stable theme than a meeting, though the approach of using a provided text file to 'filter' the output of a speech recognizer is one which shall be examined further.

This chapter dealt with the state of the art of meeting browsing. The next section deals with the implementation of the system.
Chapter 3

System Overview
3.1 The Recognizer

As mentioned previously, the speech recognition engine chosen for the task was the CMU Sphinx system. The version used was Sphinx 4, which was implemented in Java.

Possibly the main reason for choosing Sphinx was flexibility. Many recognizers are developed to do one particular thing, varying in complexity from large vocabulary dictation systems to simple digit recognition programs. The advantage of the Sphinx package is that it is fully customizable for a range of different tasks. Configuration of the system is facilitated through the XML configuration file interface.

3.1.1 Background

The Sphinx recognizer is a Hidden Markov Model based system. The basic principle of these speech recognition systems is that each phoneme in the language has a corresponding model of data for the phoneme, i.e where the phoneme is likely to occur in relation to other phonemes. In the Sphinx system, this data is known as the acoustic model.

During recognition, the audio signal is processed and sets of different features are extracted from it. These features are mapped against the acoustic model to determine which phoneme is likely to be represented by the data.

The next task is to determine which phoneme units were the best fit for the data. This requires a dictionary component to map phonemes to words and then a language model component to work out the probability of certain words occurring before or after other words. In the Sphinx system, these components are fully interchangeable depending on the task at hand. A large vocabulary system may use the N-gram model, which determines the likelihood of a word by considering the N-1 previous words in the sequence. For a simple digit recognition system, this would not be necessary.

3.1.2 Package

The Sphinx source code is freely available under the open-source framework. This was also a factor which influenced its selection as the recognizer used in the program. The package includes a number of demonstration programs
designed to showcase different aspects of the system and to provide programmers with a basis on which to build a system according to their needs. Different programs were provided, for building a word lattice, for continuous live speech recognition, and off-line speech recognition. As the project concerns off-line speech recognition, the latter demo was a perfect choice as basis for the transcription project.

A significant amount of time was spent getting to know the different components and elements of the Sphinx package. The configuration files in particular appeared rather bewildering at first glance, coupled with the lack of documentation, save for a general rundown of which packages were relevant to which task.

The demo implemented off-line recognition of an audio file containing digit data. This data consisted of the digits zero to nine, with zero output as \textit{oh}. The input audio contained a recording of different permutations of these digits spoken in a normal fashion, separated by pauses.

3.1.3 Foundations

The Transcriber demo provided a basic foundation for the program, although there is little left of the original in the final code. The basics which remain are:
• Open audio stream and create Recognizer object
• While the audio stream is still open, recognize new chunk divided by silence
• Print transcription
• When audio is finished, exit

3.2 Expansion

With the original code as a basis, the Transcriber program was created. The acoustic model used was the Sphinx HUB4 acoustic model, trained on US news broadcast news, with a vocabulary of approximately 64,000 words. The language model used was an N-gram model also based on the HUB4 data. Due to their large size (+150mb) these files are not provided with the standard Sphinx source code package and are available for download separately.

Data for the configuration file was put together using examples of other configuration files, for the most part data from the HUB4 regression tests, which measure the performance of the Sphinx Recognizer on different tasks. It involved changing the Linguist component to the LexTreeLinguist, which supported N-gram grammars and the creation of a lattice of all possible results. More information on the different Sphinx components and their construction and implementation is available in the Sphinx Whitepaper (Walker, Lamere, Kwok, Raj, Singh, Gouvea, Wolf, & Woelfel, 2004).

The main additions to the program were the facility to output the results to a file, along with the most important feature, the ability to time-stamp the transcription output.

3.2.1 Sampled Audio Processing

The audio files used in the system are standard sampled audio files. Java supports normal .wav and .au files, as well as the cepstra format which is a form of recorded speech. The .wav format was the one used in the project.

Java provides a number of different classes for dealing with sampled audio. The AudioInputStream class is used by the Sphinx Recognizer object as input. The Recognizer object is the main workhorse of the Sphinx system,
acting as a front end for the different components which make up the Sphinx package.

In order to time stamp the audio file, some understanding of the ordering of sampled audio was needed. A sampled audio file consists normally of a header section, which contains all the necessary information, bit rate, sample rate etc, and then a number of samples. The samples correspond to the sampling rate, e.g if an audio file has a sampling rate of 16000, then there are 16000 samples per second of audio. The number of bits per sample is then needed to work out the bit rate, for example 16 bit audio has 2 bytes per sample.

Armed with this information, it is possible to work out the time offset in an audio file by the number of bits or samples that have been processed. Although Java supports in the input of .wav and .au file formats, the Sphinx recognizer only supports raw audio data, which contains no header file information. This means that only the samples remain, which makes referencing the audio position easier. The AudioInputStream class removes the header information automatically, although this means that the sample rate and bits per sample must be specified in the config file. Because of this, all input files for the project are restricted to being 16bit 16000 samples per second monaural audio data. This is also the sample rate that the HUB4 audio model is recorded in, which is another reason for this restriction. For this reason, the sample rate is hard coded into the program.

The AudioInputStream class provides a number of methods for obtaining information about the audio. Using these methods, it is possible to obtain the current number of samples which have been processed, which divided by the sample rate will yield the audio position. The Recognizer class returns a Result object which represents a recognized chunk of audio, with methods to get information about the result. The result is normally a sentence, although each word can be referenced individually. The start frame and end frame relative to the result sentence itself is accessible, and then the number of overall processed samples is added to these values, giving the position of a single word in relation to the audio file as a whole.

The basic algorithm is as follows.

3.2.2 Basic Outline: Transcriber Algorithm

while(audio still remains){
get recognition result

get the highest scoring path in result

for each word in the result{

    get start frame and add sample counter
    get end frame and add sample counter
    get confidence value
    print to file
}

increment sample counter
    (total samples - number of samples remaining)

}

3.2.3 Explanation

To explain some of the above pseudo-code will involve delving into some of the nuances of the Sphinx Result object and the AudioInputStream object. The sample counter is updated by subtracting the number of samples remaining from the total number of samples. This is because there is no direct way to access the current sample number from the AudioInputStream class.

The confidence value is calculated by the recognizer, when it determines the highest scoring path. The function of the confidence value is to provide information about how sure the recognizer is about the particular word. It is originally returned as a logarithmic value, which is then converted into a standard base 10 value between 0 and 1. The original plan for the confidence value was to implement some form of threshold, with words scoring under the threshold being discarded in the final representation. Upon examination of the confidence values produced by the recognizer, it became apparent that the majority of words scored a 1 on the scale, thus rendering the confidence value redundant. This behavior led to the idea of confidence scoring being reevaluated, ultimately leading to the improvement of the output produced by the program.
CHAPTER 3. SYSTEM OVERVIEW

It is worth pointing out that the program returns the time values as fractions of a second, not in standard time format consisting of minutes and seconds. This is because these values are easier to deal with when adding offset values. Obviously, these values could be easily converted to minutes and seconds should the need arise, indeed the opposite is implemented in the front end, where offsets are entered in standard hr:min:sec format, and then converted to seconds for use in the program.

```plaintext
<s> (confidence:1) 0 0
<sil> (confidence:1) 0 161
++cough++ (confidence:1) 161 458
</s> (confidence:1) 458 458
<s> (confidence:1) 594 594
++noise++ (confidence:1) 594 594
++cough++ (confidence:1) 594 594
they (confidence:1) 594 594
</s> (confidence:1) 573 573
<s> (confidence:1) 7370 7370
<sil> (confidence:1) 7370 7377
++noise++ (confidence:1) 7377 7387
++uh++ (confidence:1) 7387 7409
++noise++ (confidence:1) 7409 7415
++uh++ (confidence:1) 7415 7432
end (confidence:1) 7432 7455
</s> (confidence:1) 7455 7455
<s> (confidence:1) 11151 11151
<sil> (confidence:1) 11151 11151
++cough++ (confidence:1) 11151 11179
end (confidence:1) 11179 11209
</s> (confidence:1) 11209 11209
<s> (confidence:1) 11482 11482
++uh++ (confidence:1) 11482 11487
today (confidence:1) 11487 11524
++noise++ (confidence:1) 11524 11560
```

Figure 3.2: Basic Transcriber Output

3.2.4 Improving the Output

With the confidence values proving unusable, it became necessary to develop a new theory for ranking the recognizer output. This involved the creation of a word lattice which represents every possible path in a result object. Using the lattice, it was possible to output the first N best results from a Result object, providing a number of hypotheses which can be processed at a later stage.
3.2.5 Word Lattice

A word lattice is a directed graph which contains all the possible paths through the recognized result. It contains a start node and a number of end nodes, with the arcs between the nodes containing different weights. The main purpose of a lattice is to output alternate results, and it is for this purpose that it is used in the system. Sphinx contains a Lattice class which provides methods for the manipulation and extraction of information from a lattice. The LatticeOptimizer class eliminates redundancy in a lattice by collapsing all nodes that refer to the same word.

The use of the lattice meant that it was now possible to have alternative results, which could later be utilised. In some cases, a huge number of alternatives were possible, which led to massive output files. It was decided to limit the number of alternative hypotheses to five, which was deemed a sufficient number. Of course, some results had less than five alternatives.

![Word Lattice Diagram](image)

Figure 3.3: Graphical Representation of a Word Lattice

3.2.6 Reorganizing the Result

With the new possibilities provided by the word lattice, it became necessary to change the output. The original output remains available and in addition a new output file was created. Originally it was proposed to print out each hypothesis with time stamp values for each word. However, due to the nature of the output from the Lattice class, which gives the list of alternatives as a list of lists, this was problematic.
In Java, a List is a linked list data structure, with some additional features. Due to the nature of Java as an object oriented programming language, every object must have a type, with all types being an instance of the super type, Object. In the case of the list data structure, to facilitate the storing of different types in the same list, the list will only store the Object type. In order to obtain the original type which was input into the list, the Object must be cast into the required type.

Although the documentation for List mentions that it is possible to have a list of lists, it warns that the behavior of the stored list may be erratic. This was certainly found to be the case. When attempting to cast the stored list, a ClassCastException was thrown. This Java exception occurs when a casting operation is attempted on an object with the incorrect type, i.e. trying to cast an object of type Apple into an object of type Orange.

After much effort was expended attempting to solve this problem, a compromise was reached. Instead of attempting to access the list word by word, and with each word the time offset value for the word, the native Java toString() method was invoked on the list, and this was then printed to a file. The disadvantage of this approach was that the time stamping accuracy is now only available at a sentence level, i.e. in the new output, individual words cannot be referenced.

Another addition to the output was the dumping of lattice files. Each result generates a lattice, which can be output in a number of different formats. One of these formats is designed to be viewed by a graph viewing program, AiSEE, which provides a pictorial view of the lattice. The lattice output is designed to aid the understanding of how the lattice is actually built. One possible implementation could be to develop a parser which can read the raw lattice output and use it directly to represent alternative hypotheses.

### 3.2.7 Front End

To enable the easy processing of audio files, a simple front end for the program was developed. This allowed the input of audio files into the transcriber, along with the setting of an offset value. This feature was added to facilitate the partial transcription of audio files. Due to the memory and processor intensive nature of the task, the processing of longer files often took a considerable amount of time and often resulted in OutOfMemory errors from the Java Virtual Machine. As a result, it became necessary to transcribe
Another issue with the Transcriber program was the fact that it proved to be difficult to get the program to exit when the audio had finished transcribing. When the audio stream ran out, a Java `IOException` occurs, and the program terminates. One way to solve this would be to ‘catch’ the exception using a Java try catch block, which handles errors without resulting in program termination. This and other issues encountered will be discussed in full in the next chapter. The next section deals with the integration of the output into a meeting browser system.

### 3.3 Integration

With the speech recognition output now usable, it was necessary to display the result to the user. A general draft of a design proposal was made, and because this design closely resembled that of an existing system, the Meeting Miner, it was decided that the speech recognition output would be integrated into this existing system, rather than attempting to develop a standalone program. This was partially due to the limited time constraints which the project was under but it also added an extra element to the Meeting Browser system itself. The author wishes to acknowledge the assistance of Mr Matt Bouamrane in this regard.
3.3.1 The Meeting Miner

The Meeting Miner is a multi modal meeting browser tool developed by Matt-Mouley Bouamrane at Trinity College Dublin. It consists of several components which include an audio stream, a shared document field for displaying the shared documents from the Shared Editor, a keyword search bar for user queries, and an indexed keywords bar which showed the different keywords which were edited and typed by the participants. A full description of the components of the Meeting Miner, along with annotated screen shots can be found in (Bouamrane & Luz, 2006b)

3.3.2 Post Processing

Before the results could be incorporated into the Meeting Miner, they first had to be cleared up of any filler elements. The Sphinx recognizer outputs filler tokens which are easily separable from actual words by use of certain symbols, for example ++um++, and ++noise++. The first hypothesis was chosen from the multiple hypothesis output, and was cleaned of any filler tokens. This was then displayed for the user.

The next step was to make sure that the time offsets were correct when
compared with the general audio recording. Because each speaker was recorded separately on a portable hard disk recorder, the transcription files were not in sync with the RTP audio which contained both speakers. For this task, the offset was measured manually by comparing the two audio files and all of the offset values in the input file were incremented or decremented accordingly.

### 3.3.3 Displaying the Output

A new display pane was added to the Meeting Miner interface to incorporate the speech recognition transcripts. The transcripts were displayed in a dialog fashion, being preceded by the name of each speaker. Further functionality was added by allowing the user to click on the speech recognition output and be taken to the corresponding point in the audio. Additionally, when browsing the audio, the speech recognition pane dynamically updates the recognition result to correspond with the current section of the audio. Some small bugs have been noticed with the component, for example, when first clicked on, the speech recognition output jumps back to the start. There also seems to be a slight lag in the speech recognition output in regard to the corresponding audio. Chapter 5 details a usability study in which users are asked to test the system and also describes how the system provides the user with keywords from the speech recognition output.

### 3.4 Conclusion

In this section, the general procedure for implementing annotated speech recognition was detailed. The basic algorithms involved were presented, and reference to the relevant classes and components of the speech recognition package was made. The next section deals with the different issues experienced during the implementation and how these issues were resolved.
Figure 3.6: Meeting Miner with integrated speech recognition
Chapter 4

Performance and Implementation Issues
4.1 Introduction

Before any of the implementation was carried out, it was necessary to become acquainted with the Sphinx package. This was no easy matter, as the source code was supplied with minimal documentation. There was a programmers guide available on the main CMU Sphinx website which detailed the basics on how to build a simple system, but this did not provide much help on the more detailed aspects of configuration. A considerable amount of time was spent trawling through javadoc files for different components to determine the exact function of each part of the system.

4.2 Installation Issues

The Sphinx system was downloaded and installed as a project in the Eclipse IDE. This first step proved to be less than trivial, because of the extended filenames of many of the components, specifically the acoustic model files, an example of which is HUB4 8gau 13dCep 16k 40mel 133Hz 6855Hz.jar.

When the project zipfile was unzipped into the Eclipse directory, which was located some way down the directory tree, some of the files, namely the acoustic models were not correctly extracted. This seemed to be because the maximum file name length restriction of 255 characters on Windows machines was exceeded by this operation. Due to this, the system exhibited some strange behaviour, and eventually it became necessary to do a clean install of the Eclipse IDE and the Sphinx system in a directory closer to the root directory, which solved all of these preliminary installation problems. The next set of issues were related to the running time and performance of the project.

4.3 Performance Issues

Large vocabulary speech recognition is a complicated task involving many different processes. The Sphinx 4 recognizer is built in Java, and running under Eclipse creates even more overhead, which can affect performance. As can be seen from the following results, a powerful, dedicated system is needed to make the system viable for use.
On their website, the Sphinx team display results from a number of different performance measurements, known as regression tests. The test system used in the tests is a dual CPU UltraSPARC(R)-III running at 1015 MHz with 2G of memory. This machine achieved a score of 3.95 on a regression test with the HUB4 system using the two processors, and a score of roughly 4.5 using a single processor. This score measures how much time it took to process the file in relation to the length of the file itself, a measure of processing time against real time.

In comparison, the machine used in the project was a 2.8ghz Pentium 4 with 512 mb of memory. On the test data used in the project, the performance score was roughly 10.5. This score is not completely accurate but reflects the amount of processing time needed to transcribe an audio file during the project. From this it is gathered that running speed is highly dependent on system resources and available memory.

### 4.3.1 Word Lattice

The aforementioned data concerned simply a best result transcription of an audio file using the HUB4 language and acoustic model. In this case, only one version of each result was provided. When the need to create a word lattice became apparent, the running time of the program increased almost exponentially. The creation of a word lattice involved a considerable amount of backtracking, and all alternative hypotheses were taken into account, unlike the earlier best path result version where the lower scoring paths are simply discarded.

There is no data on the Sphinx website which details the amount of time needed to create lattices for each result and provide all possible hypotheses, so no comparisons between systems can be made, but a rough example from the project can be given. A 30 minute chunk of audio took over 3 days (60 hours) to process using the aforementioned test system, a 2.8ghz Pentium 4 with 512 mb of memory.

In producing a word lattice for each result, it is difficult to determine how many alternative hypotheses will be produced. It depends very much on the length of the result and how many variations can be found by each phoneme to word mapping. Originally, dumping every possible hypothesis often resulted in system memory shortages and the program was often prematurely terminated by Eclipse. On examination of the configuration file, an option
was discovered which set a limit on the maximum number of edges allowed in the lattice. This was then set to 50 to avoid overly large lattice files. With this option set, the memory errors ceased.

4.4 Offset option

With processing times extending into days, the idea of being able to transcribe large files in sections became even more attractive. With this in mind, an extra offset parameter was added to the front end to enable files to be transcribed from any point, enabling the audio file to be manually split into sections and then each section transcribed separately.

4.5 Issues with Sphinx system

In general, once the installation issues had been resolved, the Sphinx system was found to be a relatively flexible speech recognition solution. There were some minor issues with some of the demo programs which resulted in Java OutOfMemory errors. The Sphinx forum on Sourceforge.net provided the solution to these problems, which was to set the Java heap size, the amount of memory available to the Java Virtual Machine, to a higher value.

The audio file limitation meant that many of the input files had to be downsampled to conform to the 16 bit 16000hz limit. This downsampling was carried out by the sound editing program Cool Edit Pro, and in took on average half as long as the audio file itself to process. The online documentation mentions that future versions of the Sphinx system may in fact be able to get sample rate data directly from the audio file itself.

4.6 Conclusion

As is often the case with software projects, it seems that more time was spent getting to grips with the basic provided software than was spent actually writing new code. Another aspect which was not ideal was the extended running time of the program, which meant that if changes needed to be made, it was sometimes necessary to wait a long time before the results of these changes could be observed. In this case, a small test utterance of
roughly 27 seconds long was often used to test the system, which meant that time would not be lost waiting for the program to finish running. It soon became clear however, that the test machine was definitely not up to the task and a more powerful machine was certainly required.

This section dealt with the difficulties experienced in the implementation of the project, in the next section, the output of the recogniser is evaluated and a small usability study is carried out.
Chapter 5

Evaluation
CHAPTER 5. EVALUATION

5.1 Introduction

According to (Bouamrane & Luz, 2006a), meeting browser systems are not easy to evaluate. There are a multitude of different criteria involved in such a task, and clear guidelines have yet to be set out as how to proceed. In this report however, it is the speech recognition component which is of paramount importance in the evaluation process, therefore we are only concerned with the feedback which is relevant to this component.

In that regard, a two pronged evaluation approach was agreed upon. An evaluation of the raw data produced by the speech recogniser was made, alongside a small usability study of the MeetingBrowser system as a whole, containing the speech recognition output. The first section looks at the results of the Transcriber program with regard to word error rate and word spotting accuracy, while the second section deals with the results of the usability tests on selected volunteers.

5.2 Part One

5.2.1 Word Error Rate

In the field of speech recognition, the WER, or word error rate, is used to measure the accuracy of a speech recognition system. The word error rate is measured as a percentage and consists of the amount of incorrectly recognized words divided by the total number of uttered words and expressed as a percentage

\[
\frac{IncorrectlyRecognisedWords}{TotalNumberofUtteredWords} \times 100 = WER
\]

Depending on the application, different word error rates may be allowable, dictation, for example would require a very low word error rate to properly function, maybe 5 or 10 percent would be permissible, but summarization may not require as low a word error rate to still be useful. In the recognition of spontaneous speech, word error rates as high as 50 or 60 percent are not uncommon.\(^1\)

\(^1\)according to (Foote, 1999)
5.2.2 The Participants

Since both of the speakers were transcribed separately, it was possible to obtain a separate word error rate for each speaker. However, time constraints made it extremely difficult to transcribe the entire two files, each 52 minutes in length, therefore a representative proportion of each file was transcribed by hand in order to generate an approximate WER.

First Participant: The first participant was a male fluent non native speaker of English. His speech was characteristic of a fluent non native speaker, carefully enunciated and clear, with various non native inflections. His speech also contained a large number of interjections, fillers and false starts.

Second Participant: The second participant was myself, a male fluent native speaker of Irish English. My speech was characteristic of a native speaker, fast paced with a good deal of interjections and abbreviations.

5.2.3 Analysis

At first glance it seemed probable that the first participant would have a lower word error rate, as a glance at the transcript revealed certain passages which were transcribed with what looked like a high degree of accuracy. However, after the sample sections were transcribed, approximately 15 mins in length starting from the beginning of the file, the results showed that this was not the case.

An approximate WER of 74 percent was recorded for the first participant, and 79.5 percent was the result for the second participant. These rates are extremely high, and in the case of most applications these would be deemed so inaccurate that they are actually unusable, however in this case the concern is not the production of word-perfect transcriptions, rather the end-goal is the successful extraction of keywords from the transcribed audio.

There are many reasons for these high values. The meeting was an example of unrehearsed, spontaneous speech, which means that it contained a large number of filler segments, which are often misrecognised by the system. In comparing the two speakers, although the first participant spoke at a moderate rate, his speech contained several non-native speaker inflections which could possibly contribute to misrecognition, along with a substantial amount of fillers and false starts. In comparison, the native speaker’s speech
was much more fast paced, which also caused difficulties for the recogniser. When transcribing the audio from both speakers it became apparent that the speed at which they spoke was a strong determining factor in the recognition result. Passages where the speech was slowed down, for one reason or another, possibly for emphasis or to clear up an earlier misunderstanding seemed to improve the recognition result. It could also possibly explain the five percent difference in WER between the two speakers. Of course, other factors must be taken into account such as the time spent talking by each participant, the first participant had considerably more speaking time than the second participant. Obviously, a complete transcription of the meeting would provide a more comprehensive WER.

5.2.4 Further Considerations

It also became clear during the transcription of the audio that perhaps the language model used by the recogniser wasn’t quite suited to the task at hand. The HUB4 language model was the only large vocabulary language model provided by Sphinx, therefore there was no choice available when it came to choosing the language model. With a 64,000 word vocabulary, it was created from transcriptions of news broadcasts. It appeared to contain a large number of proper names and surnames which often appeared in misrecognised segments, examples of these include

1. **Correct:** so go on with the pros and cons then.
   **Recognised:** it’s a country shows goleman wasn’t proven combs

2. **Correct:** depending on the mood some people like an intimate.
   **Recognised:** depending on lucinda bulletins

3. **Correct:** our cost so ill just put the costs at the very end
   **Recognised:** jones ist decosta the very low

From these and other examples, it can be construed that the language model used possibly also contributed somewhat to the high error rate.

Other considerations include microphone noise and background noise. The nature of the recording environment meant that typing would often occur during speech and this was definitely audible during the recording. In
the case of the first speaker, some noises created by microphone adjustment in the first few minutes were recognised as words. Although the high word error rate would seem to render the output almost useless, this is in fact not the case. The next section details the idea of keyword spotting in the speech recognition output, with keywords taken from the shared document created by the Shared Editor used to find more occurrences of keywords in the speech recognition output.

### 5.2.5 Keyword Spotting

In the Meeting Miner system, the keyword bar which is located above the segmented audio display is made up of keywords extracted from the document. In the updated version with the speech recognition component, the keywords are first extracted from the text document and then used to search for corresponding occurrences in the speech recognition transcription. This can improve accuracy, because the keywords taken from the edited document do not always correspond to when a word was spoken, but rather to when it was typed or pointed at.

In order to facilitate the ranking of the spotted keywords, an keyword spotting output file is created when the Meeting Miner system starts. This contains a log of at what time offset the keyword was spotted and by which speaker it was uttered. The task of rating the accuracy involved referencing the keywords with the given time value and then checking whether the word had been correctly recognised or not.

The table shows the most relevant keywords that have been taken from the speech recognition output. Analysis of these results yields a keyword spotting error rate of approximately 24 percent, approx 50 percent lower than the actual WER of the transcription results. This is a clear improvement and shows that although the overall WER for the transcriptions is high, the keyword spotting results are still very much usable.

Of course, 24 percent is still far from perfect, and it must be noted that in many cases, the number of occurrences for a word from the speech recognition transcript matches the number of occurrences in the shared document, which means that in this case new information may not have been gained. It is also worth bearing in mind that we still have not referenced the occurrences of keywords in the speech recognition output to the user entered occurrences to determine whether the times of occurrence match up, in some cases this
<table>
<thead>
<tr>
<th>Word</th>
<th>before ASR</th>
<th>ASR Matches</th>
<th>False ASR Matches</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>ticket</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>friday</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>support</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>promotion</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>sound</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>radio</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>death</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>thursday</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>euro</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>small</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>night</td>
<td>3</td>
<td>18</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>venue</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>modern</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>village</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>cons</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>price</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>finishes</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>break</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>pay</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>play</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>budget</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>provided</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>people</td>
<td>2</td>
<td>16</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>march</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>public</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>probably</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>expected</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>high</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>free</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>date</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>security</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>music</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>open</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>doors</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>time</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>tree</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>good</td>
<td>1</td>
<td>14</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>engineering</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>minimum</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>week</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>instrumental</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.1: Keyword Spotting Accuracy Table
may be true and in others this may not.

Some of the keyword spotting errors are not just to do with the recognition output. The word spotting algorithm doesn’t check whether the chosen word occurs on its own as a single word, or is actually part of another word. An example is the word \textit{cons}, from \textit{pros and cons}. The main reason why all of the ASR results for this word are incorrect is because the algorithm returns sentences such as \textit{the consideration of ......} as correct hits. Then again, the opposite can also occur, for example a possible recognition of \textit{for the night} may be \textit{fortnight}, which the word spotting algorithm will still return as an instance of \textit{night}. In this case although the recognizer has produced the wrong output, the word spotting algorithm will still report the correct keyword.

With this added information available to the user, it was time to test the whole system with some participants.

5.3 Part Two

5.3.1 The Usability Study

In order to test the system, a small usability study was carried out. Six volunteers were chosen, three males and three females, all college students in the 19-24 age group. Each participant was schooled in the functionalities of the Meeting Browser and then given 15 multiple choice questions to answer about the meeting. The theme of the meeting that was used was organizing a gig in Dublin. This theme was chosen in order to provide a theme that would hopefully be easier to relate to than a more technical topic.

The participants were given a time limit of 30 minutes to answer the questions about the meeting. The following table shows the results.

5.3.2 The Questions

On review of the questions, it became apparent that a number of the answers were rather ambiguous. The first question asked users why newspaper advertising was not used to promote the gig. During the meeting, the cost of advertising was discussed, and the price was found to be too expensive. It was also mentioned that flyers would be a sufficient means of promotion, and
both of these statements were given as possible answers. Every participant chose the former answer, therefore although the latter was considered to be the most correct when compiling the list of questions, the former answer was also accepted as correct.

Questions which were not strictly worded as they were spoken in the meeting were found to be difficult. For example, no participant answered the question relating to the percentage of bar sales that would be earned by the concert promoters. In the meeting, the question of how much money will be made from the bar is posed, the answer was none. Because the question was phrased in percentages, none of the participants managed to find an answer.

5.3.3 Analysis of Results

When looking at the results, some outside criteria must be taken into account. For example, participant number three was a non-native speaker of English. Although she answered all the questions, half of the answers were deemed to be incorrect and from the answer sheet she seemed to have been unsure to whether there was more than one answer to each question or not. This could have been the fault of myself, the organiser of the study, for not explaining the instructions more clearly.

5.3.4 Questionnaire

The real aim of the usability study was to get some feedback on the user interface, in particular how the speech recognition result was presented and how useful it was. The questionnaire contained a number of questions in
relation to the task and the user interface of the meeting browser system. The users were asked to score the different components out of ten.

The first participant found the speech recognition to be inaccurate and remarked that a possible improvement of the system would be the improvement of the speech recognition accuracy. The keyword search option was found to be the most useful component. Another issue was the fact that the two audio streams were not equal in volume, the second speaker was not always audible. When asked whether he would use such a system, his answer was that he would rather listen to the entire recording and be thorough about it, than risk missing out on information.

The second participant remarked that the speech recognition was slightly out of sync with the recording. This was an issue in development that had already been observed. She also nominated the keyword search as the most useful tool, and noticed that the second speaker volume was low. She remarked that she would use such a system if the need to browse a meeting arose, although she felt that the system had more potential.

The third participant remarked that she did not understand the speech recognition output. She found the blue keyword bar above the audio the most useful component. As an improvement, she suggested an option that allowed searching for a phrase. To an extent, this is already supported within the search box, which allows searching for multiple words, which she may not have understood. Again, this may have been due to an incomplete explanation of the system.

The fourth participant rated the speech recognition display the lowest of all the components. As improvements, she suggested a more accurate search option which allowed a user to type in the desired word, instead of the selection option currently available.

The fifth participant also rated the speech recognition display the lowest out of all the components, with the keyword search being the most important.

The sixth participant also rated the speech recognition output the lowest of out all the components, again with the keyword search being cited as the most useful component.

5.3.5 Summary

From these results and feedback, a few points have been clarified. The inclusion of speech transcripts in sentence form is not really useful, due to the
low accuracy rate. On the other hand, the keyword search was seen to be the most useful component, and this contains a large number of keywords which have been extracted from the speech recognition output.

Looking at the study in terms of the system, in the 30 minutes that the participants had to answer the questions, none of them managed to answer all 15 correctly. This could be due to a number of factors, clearly some of the questions were slightly ambiguous and this should be addressed in any future experiments. Some sort of comparative study is required, where one group is presented with the audio file alone and given a certain length of time to answer the questions, while another group is given the Meeting Browser system and a shorter length of time to complete the task.

The next chapter is the conclusion of the report and contains suggestions for future work on the subject.
Chapter 6

Conclusion
6.1 Summary

This report has dealt with the creation of a speech transcription program, the integration of this system into an existing meeting browser and the evaluation of the resulting system. Using a freely available speech recognition engine, it has been possible to create a system that when faced with the task of spontaneous speech recognition, manages to extract a large amount of relevant key words that allow the browsing of an audio meeting recording.

Like any software project, there was a certain amount of work needed at the beginning to get to grips with the packages being used, and to some extent this may have taken longer than expected. This was overcome in the end, and many lessons were learned in regard to various operating system aspects, programming language nuances and audio format issues.

Many of the difficulties in the speech recognition domain which were detailed in the introduction have been experienced during the implementation. Examples of noise and disfluencies in the audio stream, differing accents and styles of discourse have been observed first hand. It is now clear that the system resources needed for efficient large vocabulary spontaneous speech recognition are still far beyond the reach of the average home computer, and that a dedicated server running the transcription program is the clearly the only choice if the processing time is to be in any way viable.

Although the WER of the transcripts was high, the results of the keyword spotting were encouraging, and there is definite scope for further streamlining of the system.

The usability study provided useful feedback on the user interface. As was expected, the speech recognition output presented in conversational form was found to be of little use, and the keyword search option was found to be the most useful. The only possible advantage of having the transcriptions would be for the hearing impaired, but given the high word error rate this too would be of little use.
6.2 Future Work

6.2.1 Further Evaluation

To some extent, the evaluation aspect of the project could be extended. The original proposal for the evaluation of the system involved a comparative study, with one group being presented with the keywords from the shared document alone, and one being presented with the keywords which were taken from the speech recognition, and then to compare the results of each study to see if there was an improvement in the number of questions answered, or the accuracy of questions answered between the two groups. Given the time constraints and the work involved in setting up an experimental study involving participants, this was unfortunately not carried out.

Another aspect of the evaluation process could be concerned with comparing the keywords from the recognition output with the keywords from the shared document to see whether the recognised keywords actually filled in the gaps missing in the keyword bar or whether they simply reinforced the information that was already there.

It is also worth bearing in mind that all of the accuracy tests results were obtained from one meeting recording. To get an overall average accuracy rating would require the transcription and analysis of a number of different meetings with several different participants.

6.2.2 Interface Improvement

Another piece of information that was gained from the usability study was the need for a more intuitive keyword search option. This poses many possibilities, the system could be connected to an ontology database, for example, WordNet, and could provide more detailed responses to user based queries on words or phrases. This could also be used for topic detection. For example, a user searching for the word *man* could be also presented with optional results for *person* and *male*. A more user friendly query system could be developed which could take a natural language question and return suggested information as to the possible answer.
6.2.3 Improvement of Speech Recognition

Another idea is the improvement of the speech recognition output by referencing keywords that are present in the shared document. This was another feature that was to be implemented but was not due to time constraints. The speech recogniser outputs the best five versions of each transcription sentence, if five are available. The idea was to search for the different keywords in the sentence in the shared document and to rank each hypothesis accordingly.

While analysing the output, it seemed clear that the language model used was not the most appropriate language model for the task. Another possible extension would be to create language and acoustic models based on a corpus of meetings, which might be more useful for the task of meeting transcription.

Of course, one option would be to use a different speech recognition package altogether. CMU Sphinx 4 was chosen because it was free to use and implemented in Java, but there are alternatives such as the HTK speech recognition toolkit, which could also be used. The output from several different speech recognisers could be compared to find the best performing system for large vocabulary spontaneous speech recognition.
Bibliography


41
Appendix A

Transcriber Code
package demo.sphinx.transcriber;

import edu.cmu.sphinx.frontend.util.StreamDataSource;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
import edu.cmu.sphinx.util.props.PropertyException;
import edu.cmu.sphinx.result.ConfidenceResult;
import edu.cmu.sphinx.result.ConfidenceScorer;
import edu.cmu.sphinx.result.Lattice;
import edu.cmu.sphinx.result.LatticeOptimizer;
import edu.cmu.sphinx.result.MAPConfidenceScorer;
import edu.cmu.sphinx.result.Path;
import edu.cmu.sphinx.result.WordResult;
import edu.cmu.sphinx.result.TokenGraphDumper;
import edu.cmu.sphinx.result.Node;

import java.io.*;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;
import java.text.DecimalFormat;
/**
 * A simple example that shows how to transcribe a continuous audio file
 * that has multiple utterances in it.
 */
public class Transcriber {

    private static DecimalFormat format = new DecimalFormat("#.#####");
    public String inputfile = "";
    public long available = 0;
    public long samplecount = 0;
    public FileOutputStream alt;
    public PrintStream pg;
    public int off;

    //main method for creating Transcriber object

    public Transcriber(String in, int offset) {
        inputfile = in;
        off = offset;
    }

    public void setAvailable(long s) {
        available = s;
    }
}
public long getAvailable(){
    return available;
}

public void setSampleCount(long c){
    samplecount = c;
}

public long getSampleCount(){
    return samplecount;
}

//returns the current offset value
public long getOffSet(){
    return (((getSampleCount() - getAvailable()) / 160 ) + off);
}

/**
* method for running the Transcriber.
*/
public void go() {
    try {

        URL audioURL = new File(inputfile).toURI().toURL();

        URL configURL = Transcriber.class.getResource("config.xml");

        ConfigurationManager cm = new ConfigurationManager(configURL);
        Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
/* allocate the resource necessary for the recognizer */
recognizer.allocate();

AudioInputStream ais = AudioSystem.getAudioInputStream(audioURL);
StreamDataSource reader =
(StreamDataSource) cm.lookup("streamDataSource");
reader.setInputStream(ais, audioURL.getFile());

long trackstart;
long trackend;

trackend = 0;
trackstart = 0;
boolean done = false;

long start = System.currentTimeMillis();
long trans = System.currentTimeMillis();
long current = start;

Lattice lat;
LatticeOptimizer opt;
TokenGraphDumper tdd;

int filecounter = 1;
System.out.println(current / 1000);
setSampleCount(ais.getFrameLength());
setAvailable(samplecount);
System.out.println("Filesize = " + ais.getFrameLength() + " frames");
// get the label for the output file,
// by changing the extension of the inputfile to .txt
StringBuffer outfile = new StringBuffer(inputfile);
outfile = outfile.replace((outfile.length() - 3), outfile.length(), "txt");
System.out.println(outfile.toString());
    //initialize the outputfilewriters
    FileOutputStream out = new FileOutputStream(outfile.toString());
    PrintStream ps = new PrintStream(out);
    //create an output file for the graph representation with the .alt suffix
    outfile = outfile.replace(outfile.length() - 3, outfile.length(), "alt");
    alt = new FileOutputStream(outfile.toString());
    pg = new PrintStream(alt);
    while (!done) {

        Result result = recognizer.recognize();
        if (result != null) {
            String resultText = result.getBestFinalResultNoFiller();
            System.out.println(resultText);
            lat = new Lattice(result);
            opt = new LatticeOptimizer(lat);
            lat.dumpAISee("ResultLattice" + filecounter + ".gpl","Lattice" + filecounter);
            //This prints the alternative hypotheses
            printAllPaths(lat);
            //
            lat.dumpAllPaths();

            filecounter++;

            ConfidenceScorer cs = (ConfidenceScorer) cm.lookup
                        ("confidenceScorer");
            ConfidenceResult cr = cs.score(result);
            Path best = cr.getBestHypothesis();

            /*
            * print out confidence of individual words
            * in the best path
            */
            WordResult[] words = best.getWords();
for (int i = 0; i < words.length; i++) {
    WordResult wr = (WordResult) words[i];
    trackstart = wr.getStartFrame();
    trackend = wr.getEndFrame();
    /*
     * This prints out the result with filler and
     * their corresponding time offset in seconds
     * */
    System.out.println(wr + " (confidence:" +
                        format.format(best.getLogMath().logToLinear
                                        ((float)wr.getConfidence())
                        + "") + " " + (trackstart + getOffSet()));

    /*This writes out the result with filler to a file
    * taking care that the value for the end sentence token
    * is not invalid,
    */
    System.out.println(wr);
    if(wr.toString().equals("</s>")){
        ps.println( wr + " (confidence:" +
                        format.format(best.getLogMath().logToLinear
                                        ((float)wr.getConfidence())
                        + "") + " " + (trackstart + getOffSet()) + " " + (trackstart + getOffSet())
        }else{
            ps.println( wr + " (confidence:" +
                        format.format(best.getLogMath().logToLinear
                                        ((float)wr.getConfidence())
                        + "") + " " + (trackstart + getOffSet()) + " " + (trackend + getOffSet()))
        // ps.println(printResult(wr,samplecount,available,trackstart,trackend,best));
    }
}

System.out.println();
if(((getSampleCount() / 16000) - (trackstart / 100)) <= 0) {
    done = true;
} else {
    System.out.println(ais.available() / 2);
    //ps.println(ais.available() /2);
    setAvailable(ais.available() /2);
}
trans = System.currentTimeMillis();
current = trans - start;
System.out.println(current / 1000);
if(resultText == null){
    done = true;
    System.out.println("end");
} else {
    done = true;
}
// This is the main method for the Transcriber

public static void main(String args[]){
    FileSelector fs = new FileSelector();
}

// This method prints all 5 of the alternative hypotheses from a lattice

void printAllPaths(Lattice l){
    List allpaths;
    Object temp;
    List ln;
    Node store;

    allpaths = l.allPaths();
    pg.println("<resultstart>");
    pg.println("(" + (getOffSet()) + ")");
    int counter = 0;

    for(Iterator i = allpaths.iterator();i.hasNext();){
        if (counter > 5 ){
            break;
        }
        counter ++;

        temp = i.next();
        pg.println(temp);
    }
    pg.println("<resultend>");
}
}