BRANA (ብራና): APPLICATION OF AMHARIC SPEECH RCOGNITION SYSTEM FOR DICTATION IN JUDICIAL DOMAIN

By: **Bantegize Addis Alemayhu**

Dedication

ለእቴዋዬ(እናቴ):- you are the reason for all of this

Acknowledgments

All the praises and thanks go to Almighty God. Then, there are a lot of people who helps me in developing my thesis work. Dr. Solomon Teferra, my advisor, I thank you for your help and encouragement to develop this thesis work from start to end and also I sincerely thank you for providing me the necessary data’s. Thank you all the time. Dr. Martha Yifiru thank you for your unlimited help. I sincerely thank the entire researcher’s that work on Automatic Speech recognition which is a big commitment that you all show me with your works which is a ground for my thesis work. W/ro Meseret thank you for your support in helping me get the data’s I need from where you work. You have done a lot for me thank you very much. I really thank workers of Lideta High court transcription department which allows me to use their speech and text collections. Thank you my friends and families who are alongside to “boost” my working mood and those that were with me in the entire work. My Mom you are the one who put me in school, you are the reason where I am now. Thank you is not enough for what you have done for me. I am so lucky in this world that God gave me you Mom. I love you so much.

Table of Contents Page

List of Tables Page

List of Figures Page

Abstract

This thesis work shows the possibility of developing an Amharic speech recognition application for dictation. In order to realize the final work we followed a sequence of works starting from reviewing different literatures about automatic speech recognition and applications of automatic speech recognition for Amharic language and other than Amharic language until developing dictation application prototype.

From the application viewpoint dictation systems represent more technological advances in speech recognition than state-of-the-art. Dictation applications make it possible to use speech instead of typing. A dictation application needs speech recognizer as its core component. To develop and test the required speech recognizer performance 90 min spontaneous speech data was collected; it was segmented and labeled into sentience's with freely available toolkits. Using the prepared corpus we developed a spontaneous speech recognizer based on Hidden Markov Model (HMM). To develop the recognizer we used sphinx which is an open source recognizer engine. In order to complete the full system we developed a python script that generates a canonical pronunciation dictionary and we developed a tri-gram language model with four different smoothing techniques.

In order to test the recognizer performance we prepared 68 utterances. The recognition performance was performed in a batch recognition mode. During batch recognition mode we get speech accuracy of 50% and 53% of WER for spontaneous speech.

Finally we developed dictation application prototype with java programming language and an Add-On extension feature that allow us to use open office writer as a dictation application. Our prototype is tested with three different users to check its performance and communication capability with the core recognizer. The prototype that is developed with java performed better than open office writer extension feature.

We also developed continuous speech recognizer with 20 hours speech corpus and we got 75% word accuracy with 359 utterances. Our dictation prototype was also tested with continuous speech recognizer and it shows a very good performance.

CHAPTER ONE

INTRODUCTION

1.1. General Background

Automatic speech recognition (ASR) means an automated process that inputs human speech and tries to find out what was said. ASR is useful, for example, in speech-to-text applications (dictation, meeting transcription, etc.), speech-controlled interfaces, search engines for large speech or video archives, and speech-to-speech translation [**1**].

Automatic speech recognition (sometimes referred to as just speech recognition, computer speech recognition or erroneously as voice recognition) is the process of converting speech signals uttered by speakers into a sequence of words, which they are intended to represent, by means of an algorithm implemented as a computer program. The recognized words can be the final results, as for applications such as data entry and dictation systems or the words so recognized can be used to trigger specific tasks as in command and control applications.

Automatic speech recognition gives us a new channel for communication with computers. Speech is our natural means of human-to-human communication and it is a clear advantage if we can communicate with computers on our own terms - not the computers. There are also several other arguments for the technique in many applications [**2**].

Speech technology is the technology of today and tomorrow with a developing number of methods and tools for better implementation. Speech recognition has a number of practical implementations for both fun and serious works. Automatic speech recognition has an interesting and useful implementation in expert systems, a technology whereby computers can act as a substitute for a human expert. An intelligent computer that acts, responds or thinks like a human being can be equipped with an automatic speech recognition module that enables it to process spoken information. Medical diagnostic systems, for example, can diagnose a patient by asking him a set of questions, the patient responding with answers, and the system responds with what might be a possible disease [**1**].

Having a machine to understand fluently spoken speech has driven speech research for more than 50 years. Although ASR[1] technology is not yet at the point where machines understand all speech, in any acoustic environment, or by any person, it is used on a day-to-day basis in a number of applications and services.

Figure 1.1 illustrates the major modules of an ASR system and their relation to applications. In feature extraction, signal processing techniques are applied to the speech signal in order to dig out the features that distinguish different phonemes from each other. Given the features extracted from the speech, acoustic modeling provides probabilities for different phonemes at different time instants. Language modeling, on the other hand, defines what kind of phoneme and word sequences are possible in the target language or application at hand, and what are their probabilities. The acoustic models and language models are used in decoding for searching the recognition hypothesis that fits best to the models. Recognition output can then be used in various applications [**3**].

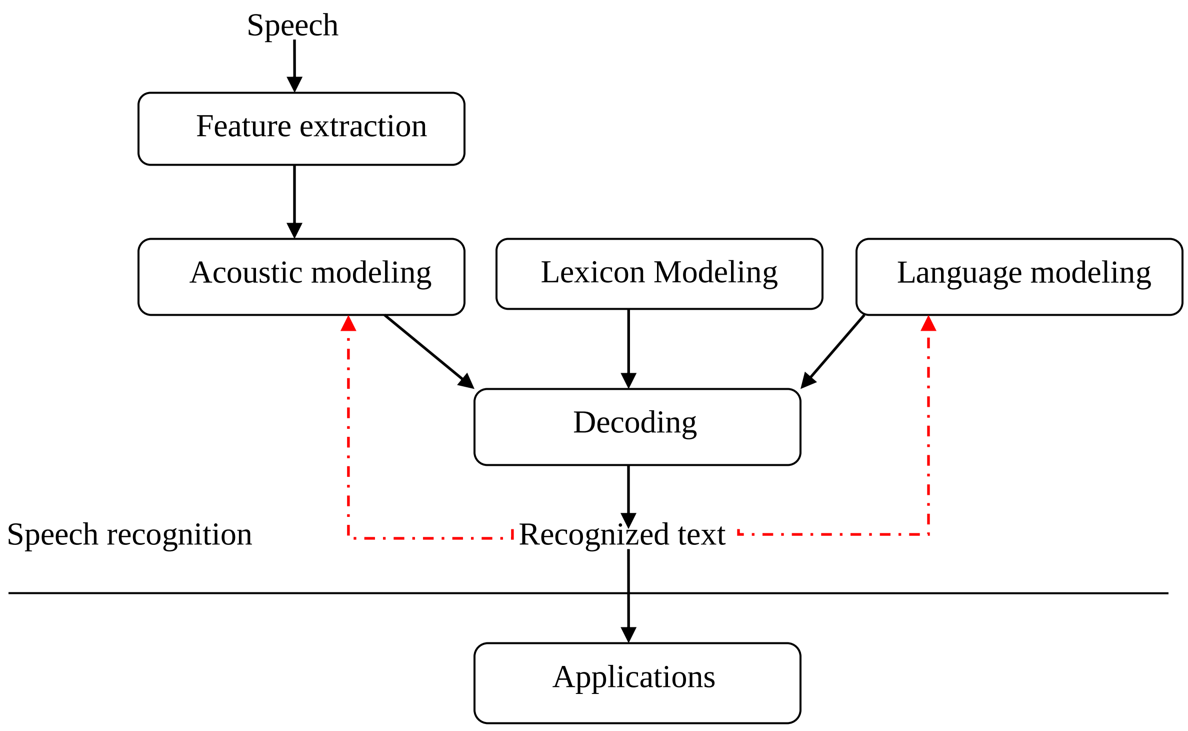

Figure 1.1: The main components of an automatic speech recognition system and their relation to applications.

[1] ASR: Automatic Speech Recognition

[2] IPS: Inter Press Service

Chapter one: Introduction

Speech recognition is mostly applied in command and control, data entry and retrieval, and dictation functions. It is important to note that one application can serve more than one functions. For example, Microsoft Office speech interface serves command and control as well as dictation functions [**4**].

Earlier dictation systems were developed based on isolated word speech recognition. However, since speaking one word at a time is tedious, people prefer to type than dictate in this manner. Nowadays, continuous speech recognition systems make dictation systems attractive. Although it is possible to base dictation systems on small vocabulary, speaker dependent, isolated word speech recognition, the ideal speech recognition system for dictation application is large vocabulary, speaker independent, and continuous recognition system [**4**].

A general-purpose speech recognition system (a dictation system) can be used to encode documents into text. This may be especially useful for languages that have a very large character set. Large characters set languages are difficult to type using keyboards, as they require a number of keystrokes to type for a single character. For such languages typing is usually a very time consuming process. Amharic is an example of this type of languages [**5**].

1.2. Benefits of Speech Recognition

Researchers pointed out a number of benefits that computer speech recognition is coming up with [**6**]. It is used to:-

- Increased productivity by enabling a person to use his/her hands and mouth for different tasks and making hands-free work possible
- Rapid return on investment that apply ASRSs to speed up tasks
- Access to new markets (24-hours service)
- Environment control (by disabled people, e.g)
- The naturalness of communication between man and machine
- Enable us to orally dictate our computers
- Automatically translate spoken language
- Enable us to communicate with remote computers (e.g.: Tele-banking, Expert system, Database-query, Information retrieval, etc)
- Have voice-controlled equipment (e.g.: Car, Airplane, Robotics, etc)

- Automatically receive service requests in different organizations, primarily telecommunications
- Assist/aid persons with disabilities
- Support teaching oral skills
- Develop a multimedia Computer-Assisted-Instruction system and
- Analyze speech evidence for the police and court

1.3. Statement of the Problem and Its Justification

The growth in Automatic speech recognition technology inspired different researchers to explore and investigate applications of automatic speech recognitions for different languages. As a result of the tools for automatic speech recognition that are available freely for research purpose there are lots of researches that are done on automatic speech recognition including Amharic Automatic speech recognition which are developed in different times. But still the researches that are done for Amharic are not satisfactory and enough specially on the applications of Amharic speech recognition.

The work of [**7**] [**8**] developed Amharic consonant vowel (CV) syllable and sub word-based isolated word recognizers. Consequently other researchers also put their effort to develop Automatic speech recognizer for Amharic. But there is only one research [**4**] work done on the applications of Amharic speech recognition system to command and control computer which shows experimental results for 16 commands to control Microsoft word processor by implementing Amharic speech input interface as a communication layer to the speech recognizer.

The experimental results and technical detail explanations in [**4**] motivate us to conduct further research on the area of applications of Amharic speech recognition system by further exploring the possible applications of automatic speech recognition for Amharic language as a motivation of those previously done research works that are done on this research area and we are inspired to delve and work on the application of Amharic speech recognition for dictation on a specific domain.

Dictation has been a common application area for ASRS for a long period. It includes medical transcription, legal and business dictation, as well as general word processing. This is especially helpful for many people who have difficulty in typing due to physical limitations and for those who are very slow in typing. We can even consider using it to allow the illiterates to dictate their ideas [**6**].

To the best of our knowledge there is no any Application of Automatic Speech Recognition for dictation for Amharic language. We try to investigate and experiment the use of implementing an Application of Automatic Speech Recognition in dictation for Amharic language in a specific domain.

We also investigate and experiment the possibilities of developing an Application of Automatic Speech Recognition for Amharic language in a specific domain to help the users to minimize their work load by using their voice for typing rather than using hands to type from the keyboard which will save their time.

The need that we are able to conduct this research work is that since there is no any research or project done in the application of Amharic automatic speech recognition application for dictation we are sure that this research work will put some ground and light for future researchers that are interested to work on applications of Amharic speech recognition.

1.4. Research Questions

At the end of this study this questions are answered with the result obtained from the experiment:

- What are the challenges of developing Amharic dictation applications?
- What changes can be made to BRANA Amharic dictation system to further improve its performance?
- Which editors are more appropriate for Amharic dictation system?

1.5. Objective of the Study

1.5.1. General Objective

The general objective of this research is exploring and conducting experiment to show the possibilities of developing Application of automatic speech recognition for Amharic language for dictation in a specific domain.

1.5.2. Specific Objectives

The specific objectives of the study are:

I. Review about Amharic Speech recognition systems and their relevance for applications
II. Identify the best way of making acoustic model, language model, and pronunciation dictionary for the dictation system
III. Choose the appropriate editor
IV. Building speech corpus for the experimentation in a judicial domain
V. Develop the system (recognizer and prototype of dictation system)
VI. Testing and evaluation of the system developed for experimentation
VII. Forwarding the results obtained and show directions for future studies

1.6. Scope and Limitations of the Study

Because of time constraints and resource available the study only includes prototype and experimentation. It did not implement full Amharic dictation application. The data that we used for training our speech recognizer is limited to about 90 min speech corpus which is a very small corpus. The corpus that we collect is judicial domain corpus speech which is recorded inside a room with lots of audience available which have lots of recording device and environment noises, speaking style of the participants is spontaneous which is a big limitation for the performance degradation of our recognizer which obviously degrades the performance of our system (BRANA), it is advisable to use continuous (read) speech data to make a dictation system but we are unable to get continuous (read) speech corpus in a judicial domain. We used Dr. Solomon Teferra's corpus to test our system performance with continuous (read) speech corpus. Since the aim of the study is to explore and experiment the possibilities of developing Amharic dictation application it may not be used as an actual application. Since dictation application needs a recognizer we modeled our own spontaneous speech recognizer system with a judicial domain speech corpus.

It is experimented with the scope of:

- Small speech corpus of judicial domain
- Developing automatic speech recognition system for Amharic used to decode on the selected domain
- Developing a Prototype dictation system for experimentation

1.7. Methodology of the Study

The study uses different methodologies to realize the research work. These are the methods that we used in the study:-

- ✓ **Literature review**

  literature review is conducted on speech recognition, application of speech recognition, human-computer interaction, Hidden Markov Model and its application in speech recognition. Moreover, researches conducted on Amharic speech recognition are reviewed. To understand how CMU sphinx works different e-books, literatures are reviewed and different internet sites are surfed. Also we reviewed different literatures that are done for developing dictation systems in another language.

- ✓ **Data collection methods**

  - ❖ **Data preparation**

  The speech corpus that is used for the experimentation is collected from Ethiopian judicial authority from lideta Federal High court.

  The speech corpus is segmented and labeled for training and testing. We also tried to prepare text corpus that is used for the language modeling and to prepare lexicon model.

- ✓ **Modeling Technique**

  Prototyping approach is used in the course of developing the dictation system. Prototype of Amharic spontaneous speech recognizer is modeled using CMU sphinx open source speech recognition toolkit.

  To develop the required dictation system, Java programming language is used. This programming language is preferred over others because it is platform independent, user friendly and most of all sphinx4 decoder is entirely developed with java programming language which eases our effort to make a compatible dictation system.

  The dictation system also experimented on apache open office writer which is an open source word processor since our developed dictation editor does not include all the functionalities of word processor we have also tested our recognizer by implementing an Ad-On functionality into apache open office word processor.

- ✓ **Testing and Evaluation**

  The recognition system is evaluated or tested for its accuracy and also we have tested the accuracy of the prototype of our dictation system. To test the recognition system, the

CMU sphinx decoders are used and then the accuracy is reported on the results section with different reporting mechanisms.

The developed dictation system and the recognizers together are evaluated how they are communicating with each other and the performance of the dictator also is evaluated.

1.8. Organization of the Thesis

Chapter 1 introduces all of the contents of the rest of the chapters and it also describes the statement of the problem and its justification, objectives of the research General objective and specific objectives, methodologies that are used for completing the entire study, scope and limitation of the study.

Chapter 2 is literature reviews on the ASR systems, systems that are developed for Amharic language, applications of ASRs, the components of ASR, Automatic speech recognizers that are developed for Amharic, related works that are done on different language etc.

Chapter 3 introduces Amharic language and the linguistic features of Amharic language and its use for the development of ARSs applications.

Chapter 4 describes the methodologies that we used and the tools that we used for the entire research.

Chapter 5 describes prototype of the proposed research work

Chapter 6 shows the results that are obtained from the experiment we conducted

Chapter 7 includes conclusion and recommendations for further researching on the applications of Amharic speech recognition

CHAPTER TWO

LITERATURE REVIEW

In this chapter we have been reviewed different literatures that are related to speech recogniton applications also the basics of speech recognition. It is classified in different sections that describe different concepts in speech recogntion. In the first section we discovered on basics of speech recognition and how it works. In the second section it discovers on the speech recognition applications technologies and its details. Third section discovers on types of speech recognition applications.

2.1. Basics

A speech recognition system aims to find out the most likely word sequence W given the acoustic signal A. With these criteria, the posterior probability P (W|A) is a theoretically accurate measure to judge the correctness of a word. However, in an actual speech recognition system, the maximum likelihood estimation is always used instead by eliminating the constant value P (A), which makes the value assigned to every word a relative rather than an absolute measure [**9**].

According to Markowitz which is pointed on the work of [**4**] emphasizing speech recognition system that plays the role of human-computer interface, indicated that such system performs three primary tasks: preprocessing, recognition and communication.

2.1.1. Preprocessing

Speech is an analog signal “a continuous flow of sound waves and silence”. This form cannot be processed directly by a speech recognizer. Thus, there is a need for preprocessing the speech signal. Preprocessing is the conversion of spoken input into the form that is acceptable by the recognizer. Specifically, it deals with digitizing analog speech and converting the speech wave form to some type of parametric representation. Analog to digital conversion is done using analog to digital converters [**4**] whereas conversion to parametric representation is done by digital signal processing front ends. According to [**9**] signal processing is stated that in which features are extracted from speech waveform, and parameter estimation, in which model parameters are iteratively updated using the training corpus and the pronunciation dictionary. According to C’ernocky which is sited on the work of [**6**] feature extractions consisted of the following steps.

Segmentation: The speech signal is divided into segments where the wave form can be regarded as stationary.

Spectrum: Current methods for feature extractions are mostly based on the short term Fourier spectrum and its changes over time, therefore, the power or magnitude Fourier spectrum is computed in the next step for every speech segement.

Auditory-like modifications: Modifications inspired by physiological and psychological findings concerning human perception of loudness and sensitivity in relation to frequencies, are performed on the spectrum of each speech frame.

Derivatives: Feature vectors are usually completed by the first and second order derivatives of thier time trajectories (Delta and accelaration co-efficients).

2.1.2. Recognition

In this step which the system identifies what has been said after following preprocessing. There are three approaches to automatic speech recognition. These are: **acoustic-phonetic**, **pattern recognition** and **artificial intelligence**.

2.1.2.1. Acoustic Phonetic

Acoustic-phonetic approach is based on the theory of acoustic phonetics. This theory "postulates that there exist finite, distinctive phonetic units in spoken language and that the phonetic units are broadly characterized by a set of properties that are manifest in the speech signal, or its spectrum, over time." It has two steps (segmentation and labeling, and word level recognition) besides the preprocessing step which is common to all other approaches [**4**].

2.1.2.2. Pattern Recognition

Pattern recognition approach to speech recognition, unlike acoustic-phonetic approach, is the one in which the speech patterns are used directly without explicit feature determination and segmentation. This method involves two steps: training of speech patterns and recognition of the pattern via pattern comparison. During training phase speech knowledge is brought into the system. The idea behind the training phase is that if enough versions of a pattern to be recognized are included in the training set that is provided to the algorithm, the training algorithm should be able to adequately characterize the acoustic properties of the pattern. In the second step – pattern recognition – unknown speech or the speech to be recognized is compared with each possible pattern learned in the training phase and classified according to the goodness of match of the patterns [**4**].

2.1.2.3. Artificial Intelligence

AI approach is a hybrid of acoustic-phonetic and pattern recognition approaches since it exploits ideas and concepts of both methods. This approach attempts to simulate the way a person applies his intelligence in visualizing, analyzing and making a decision on a measured acoustic features. To do this, the approach requires different knowledge. In fact, the basic idea of AI approach is to "compile and incorporate knowledge from a variety of knowledge sources and to bring it to bear on the problem at hand." The knowledge required by AI approach is acoustic, lexical, syntactic, semantic and pragmatic. Some of the techniques used in this approach are the use of expert system for segmentation and labeling, learning and adapting over time, and the use of artificial neural network (ANN) for learning the relationship between phonetic events and all known inputs as well as for discrimination between similar sound classes [**4**].

2.2. Speech Application Technology

Although speech is the most natural form of communication between humans, it has several limitations when used in interaction between humans and computers. Some of its advantages which are present in human-human communication cannot be used in human-computer interaction. There are multiple reasons for this, both technical shortcomings and those which are related to the nature of speech as a communication modality. In many cases, communication is not based on the best possible solutions, but instead the technology limits choices and even dictates the design [**10**].

In order to successfully use speech in the interface, we must know the state of the art and reality of speech technologies, as well as what kind of interface element speech is. First, it is important to realize that speech-based human-computer interaction is different from written language and spoken communication between humans. Secondly, speech technology has many technical limitations, and it is not realistic to believe that these will disappear in the near future. In this section those issues of speech technology which affect spoken interaction are presented.

2.2.1. Levels of Speech Processing

Speech can be structured in many different ways depending on its use. One useful categorization is illustrated in Figure 2.1. It contains eight levels, which are organized into three layers. The core speech technologies (speech recognition, speech synthesis) belong mainly to the bottom layer (acoustic, articulator and phoneme levels), while applications deal mainly with the top layer (discourse, pragmatic and semantic levels). The middle layers (syntactical and lexical levels) are needed both in technology and application areas. When speech architectures are considered, we are dealing mainly with interaction methods, but many issues related to the core technologies should be taken into account as well.

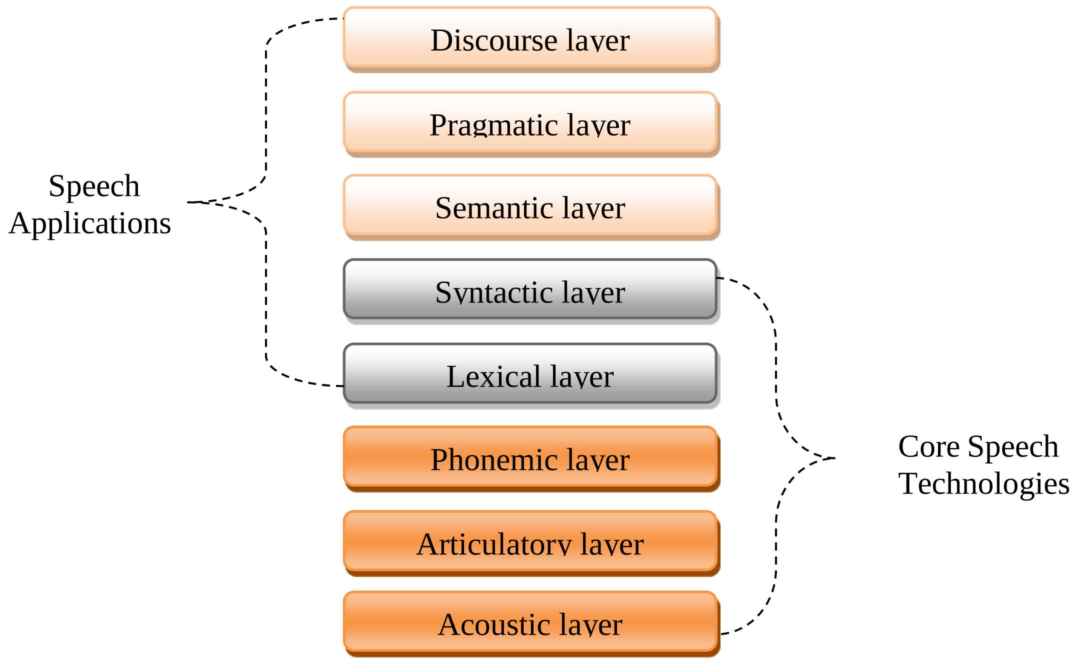

Figure 2.1: Levels of speech processing (adapted from [**10**])

2.2.2. Properties of Speech Systems

Speech recognition systems can be separated in several different classes by describing what types of utterances they have the ability to recognize. These classes are based on the fact that one of the difficulties of ASR is the ability to determine when a speaker starts and finishes an utterance. Most packages can fit into more than one class, depending on which mode they're using [**11**].

Main characteristics of speech systems from the point of view of the speech recognition technology are summarized in table 2.1. It includes those features which are most important when a speech recognizer's suitability for a particular task is evaluated. The properties of the task and the recognizer should match as closely as possible. From the point of view of a speech recognition task, the most difficult properties are in the far right column. Not surprisingly, those are also the most desired properties in speech applications. Next, these properties are presented in more detail.

Vocabulary and language			
Vocabulary size	Small	middle-size	(very) large
Grammar(LM)	Phrases	CFG	n-gram
Extensibility	Fixed	Changeable	dynamic
Communication style			
Speaker	Dependent	Adaptive	independent
Speaking style	Discrete	Continuous	spontaneous
Overlap	half-duplex	barge-in	full-duplex
Usage conditions			
Environment	Clean	Normal	hostile
Channel quality	high-quality	normal-quality	low-quality

Table 2.1: Properties of speech recognition systems [adapted from [**11**]]

2.2.2.1. Vocabulary Size and Recognition Grammars

Vocabulary size and recognition grammars (language models) characterize the interaction possibly better than other properties. By using small vocabulary recognizers (around 10-200 words) it is possible to construct many spoken dialogue applications, even with conversational abilities, but some application areas are not possible. For example, it is possible to construct fairly sophisticated speech-only e-mail applications with a couple of dozen words in the vocabulary. However, if we want to build bus timetable systems we usually need at least a middle-sized vocabulary [**10**].

Recognition grammars also have a great influence on the interaction model. If we can change or dynamically construct grammars, we can design the system to be context-sensitive and utilize dialogue knowledge, interaction history and user profiles to use personalized recognition grammars. In this way, we can use several small and robust vocabularies in different situations. On the other hand, if we cannot change grammars, we should design the grammar and the system interaction model to be suitable for all situations and to all users. Some current speech systems offer both n-gram and context free grammar.

Language models and an option to use large or even very large vocabularies (over 50,000 words).N-gram models are most usable in dictation and other tasks which require very large vocabularies, and in which usage conditions are optimal and the recognizer can be adapted to the speaker. On the other hand, in hostile environments, where speakers are unknown, even small vocabularies and restricted grammars may cause problems.

In practice, most speech recognizers have serious problems with open grammars and large vocabularies. Furthermore, defining large vocabulary grammars or collecting relevant data for n-gram models can be a laborious task. Fortunately, most speech applications do not need large vocabularies and can be designed to work with small or medium sized vocabularies and limited grammars if they are properly designed.

2.2.2.2. Communication Style

Communication style may vary from speaker dependent, discrete speech and half-duplex communication to speaker independent, continuous and full-duplex communication. Each of these parameters has an impact on the interface. For example, the barge-in capability allows users to interrupt system outputs, and this way the outputs can be designed to be longer and more informative than without the barge-in capability. In some applications it is possible to use speaker dependent or adaptive models if the recognizer supports them, but in public applications, where users are not known, speaker and gender independent models are the only possible choices. Speaker adaptive models may be very robust, especially if customized vocabularies are used, which makes their use justified in some application areas [**10**].

With current recognizers there is no need to speak in a discrete manner. However, even if the speaking style can be continuous, it usually helps if words are pronounced clearly and properly. In practice, recognizers are not very robust for spontaneous speech. Word-spotting can be a good choice for some applications, but it is not supported efficiently by most recognizers.

Overlapping speech is not in fact a property of speech recognizers, but rather a property of a system. Barge-in affects prompt design, but also dialogue design. With a half-duplex interface the speech recognition grammar must be designed to support strictly turn-based communication.

With full-duplex support the user and the computer may speak at the same time, regardless of each other. In a typical situation the user wears headphones and uses a close-talk microphone. Barge-in is typical for telephone-based communication and brings more flexibility to the interface than the half-duplex interface. It is important to note that even if the underlying technology supports these features, applications must also provide support.

2.2.2.3. Usage Conditions

Usage conditions range from clean to hostile environments and from high-quality to low-quality channels. High-quality conditions may include close-talk microphones in single-person offices. In low-quality conditions, mobile phones may be used in noisy, public places. More typical conditions can be shared offices and homes in which basic desktop microphones are used. Even with state of-the-art recognizers, the recognition accuracy may be dramatically impaired if usage conditions do not meet those used in the recognizer training phase. Recognizers may contain different acoustic models for different conditions. The lack of a proper acoustic model may make the recognizer unsuitable for certain tasks.

Finally, differences between recognizers for different languages may be considerable, for example there are highly sophisticated recognizers available for English users, but only a couple of limited recognizers are available for Finnish users. When applications for multiple languages are constructed, the limitations of different recognizers must be taken into account and the system interaction must be adapted according to the resources available. This is one of the issues which should be addressed in the system architecture and interaction techniques levels.

Speaker recognition, i.e., speaker identification and speaker verification are not widely used in practical applications, but they are expected to become more important in the future. They are especially useful in pervasive applications, in which speakers must be identified in a transparent manner and the interaction should be adapted to each particular user.

2.3. Types of Speech Applications

Speech application types are classified in different ways with different researchers and books in this section we have been try to see different types of speech applications that are pointed by different researchers. According to S. Furui which is pointed by [**9**] Based on the target of speech (human or computer) and speech interaction style (monologue or dialogue), speech recognition applications can be generally classified into four categories:

- ***Human-human dialogue*****:** speech recognition systems generate transcription of human dialogue, such as conversational telephone speech and meetings. Transcribing human dialogue is a challenging task due to the spontaneity of speech.
- ***Human-computer dialogue*****:** spoken dialogue systems allow humans to talk with computers and to complete certain tasks using their voices. A spoken dialogue system, which includes a speech understanding system and a speech synthesis system, is more than a speech recognition system. Restricted by the limited progress in automatic language understanding, spoken dialogue systems are confined to specific domains such as travel planning services.
- ***Human-human monologue*****:** in this category, human speech is used for conveying information, such as broadcast news, TV shows, and lectures. Speech recognition is the secondary task in which recorded audio data can be transcribed and indexed for future information management.
- ***Human-computer monologue*****:** a human speaks to a computer that transcribes speech into text. Dictation, which is generally used as a document generation tool, is the most prevalent application in this category.

According to Kenny, Nelson, Bodenschatz, and McMonagle which is cited on the work of [**9**] based on the degree of preparation for the dictated text, dictation can be further divided into two sub-categories: transcription and composition. Transcription is the task of reproducing written documents; speakers read written documents to speech recognition systems which transcribe the speech and convert documents into a machine readable format. Compared to transcription, composition offers a more realistic speaking scenario. People directly use dictation software to compose documents such as emails and reports by speaking to computers with little or no preparation. People spontaneously dictate what they want to write down, which makes composition more error prone. Since the final goal of dictation is to generate a written document, the speaking style, as reflected in dictated sentences, is close to the writing style used in written documents.

To produce an error-free speech transcript in monologue applications such as dictation, error correction is important and indispensable. Error correction involves three steps: detecting an error, navigating to the error, and correcting the error. Manual error correction is neither easy nor efficient.

One way to approach speech applications is to divide them into two categories based on the modalities used. In the first category are applications which use speech as their main modality, while to the second category belong multimodal applications where speech is not the dominating modality. An alternative approach is to classify systems into conventional speech applications (telephone applications, dictation system, voice portals) and new application areas (pervasive computing, mobile applications). Here speech applications are divided into conventional, multilingual, multimodal and pervasive applications, because each of these areas has different development and research questions. In practice, it is not important how applications are classified, but it is important to realize that speech applications are much more than traditional spoken dialogue systems and there is a wide range of research problems associated with them beyond the speech recognition [**10**].

As there are multiple application types, applications can also be constructed from many different perspectives by using different approaches. The most advanced systems are usually produced in multidisciplinary projects. A typical project consist of people working with core technologies (speech recognition and synthesis, natural language processing), while others work with interaction technologies (dialogue management, human-computer interaction). People with different backgrounds may have overlapping solutions for the same problems.

The terms speech application and spoken dialogue system are used commonly. The latter is often used to refer to applications which emphasize the use of natural spoken language. According to Fraser on [**10**] often these systems operate on a turn-by-turn basis and the interaction is modeled as a dialogue between the computer and the user. This term is widely used by authors with a background in speech technology or dialogue management research. Sometimes it is also used to distinguish conversational (natural language) systems from more command and control oriented systems. It may also be used to refer to techniques used in dialogue management. Bernsen et al. which is cited by [**10**] have categorized speech systems and use the term “interactive speech system” to emphasize the language processing parts of speech applications in contrast to speech applications which do not use natural language processing techniques. Here the term speech application is used to refer to all speech-based applications regardless of their focus or implementation details.

2.3.1. Conventional Speech Applications

The first speech applications were telephone-based interactive voice response (IVR) systems, which used speech outputs and telephone keys for interaction. These applications are still popular and possibly the most important examples of widely used commercial speech applications. Typically, these applications are designed to replace human operators (i.e. speech is used as a substitute). Problems with these systems include that the DTMF interface may be awkward and user satisfaction poor, if the service does not match the quality offered by human operators. On the other hand, these applications can be very useful for users if they introduce new services which are not possible or affordable with human operators (or in any other way). From the telephone operator's point of view these applications may offer huge savings in the long run. The current trend is that DTMF inputs are replaced by speech inputs [**10**].

In addition to IVR applications, many other forms of telephony applications have dominated the field. These include information services, such as timetable, weather forecasting and banking services, e-mail applications and voice portals. Most of the well-known commercial applications and research prototypes fall into this category [**10**].

Desktop applications from another popular area of speech applications. Most desktop applications are either targeted at dictation or the control of existing graphical applications, such as word-processors or spreadsheet programs. Dictation applications make it possible to use speech instead of typing. Some dictation applications allow also transcribing recorded speech. Many companies offer dictation applications, either for general use or customized for special domains, such as medicine. Dictation applications are relatively popular within special groups. From the application viewpoint dictation systems represent more technological advances in speech recognition than state-of-the-art in interactive system development, although usability is an important issue in dictation applications. There are still many interesting challenges related to these applications (e.g. error correction). One problem with current dictation systems is that they rely quite heavily on visual feedback and multimodal inputs. This limits their usefulness in speech only applications, even if the underlying speech technology makes it possible to dictate.

2.3.2. Multilingual Speech Applications
Translation systems are one example of multilingual speech applications. These systems translate spoken utterances between users. Typical applications include booking systems and other similar applications, where two human participants have a common goal, such as booking a hotel room or a trip, and the computer translates their spoken utterances between languages in real-time.

Another form of multilingual applications is systems allowing users to access information services using multiple languages. A third type of multilingual applications uses multiple languages with the same user. The need for multiple languages arises from the application domain.

2.3.3. Multimodal Speech Applications
Multimodal speech systems have been a popular research subject, but still multimodal systems are rather rare in everyday use.

Among the most studied multimodal speech applications are so-called “talking heads” or “speaking agents”. Good examples are systems built at the KTH, such as August cited on [**10**] and Adapt cited on [**10**]. In these systems the main interface element is audiovisual speech synthesis, which uses anthropomorphic figures to convey facial expressions and head-movements.

Another active area of research in multimodal applications is information kiosks (intelligent kiosks). In these systems modalities such as speech and hap-tics (e.g., touch-screens) are used to provide interface for users in public places.

2.3.4. Pervasive Speech Applications
Pervasive applications, i.e. ubiquitous and mobile computing applications, introduce both new opportunities and challenges for speech as an interface element. Speech is a suitable element for pervasive applications because of the size of the devices, and the lack of large visual displays and familiar interaction devices, such as the mouse and the keyboard. By embedding systems into everyday environments (so-called “intelligent environments”) the focus of speech applications shifts from turn-based, user-initiative conversational systems in a more proactive and distributed direction.

An example of an intelligent environment is the MIT Virtual Room project work of Coen which is cited on [**10**]. The idea of its speech interface is to use context knowledge to allow multiple speech applications to be active at the same time with no need for central control.

Other well-known applications developed by Yankelovich & McLain are Office Monitor and Sawhney & Schmandt Nomadic Radio cited on [**10**]. Office Monitor is a system which serves visitors in an office while its occupant is away. Nomadic Radio is a wearable speech system, which offers several information services, such as listening to e-mail and voice-mail messages, calendar events and news. In addition to synthetic speech, the system uses non-speech audio, speech recognition and tactile inputs.

Systems involving robots are examples of applications in which the context plays an important role. These systems may include multiple participants, which makes the dialogue rather different from traditional single-user and single-system dialogues.

2.4. Related Works

In this section we will present related works on Applications of speech recognition for dictation that are done in other languages and speech recognizers that are done in the past by other researchers for Amharic language. To the best of our knowledge, there is no research work done for Amharic on Applications of speech recognition for dictation except there is only one work of [**4**] application of Amharic speech recognition system to command and control for Microsoft word.

Turkish dictation system for radiology and broadcast news applications: in this work the researches designed a Turkish dictation system for Radiology and Broadcast news applications. Because of Turkish language nature that it is an agglutinative language with free word order the researchers uses classical word-based language models, they achieved 87.06 per cent recognition performance with a small vocabulary size in a speaker independent radiology speech recognition system. However, the same system results in 46.29 per cent recognition rate for the broadcast news dictation due to the large number of out-of-vocabulary (OOV) words. Due to this they parse some of the words to smaller recognition units like stems, endings and morphemes, and introduced these smaller units and the unparsed words to the speech recognizer as lexicon entries. They built LVCSR recognizer for the dictation system they developed. They selected radiology domain to build the dictation system. They designed a graphical user interface for their radiological dictation system [**12**].

Speaker-independent Dictation of Chinese Speech with 32K Vocabulary: they developed speaker-independent, word-based dictation. A deliberately designed 120-speaker database was built for training ; inter-syllable context ,tonal and endpoint dependent acoustic model are applied with promising MFCC feature; Two-pass acoustic matching accelerates the recognition making fully advantage of the monosyllabic structure of Chinese speech; A complete word bigram and trigram serve as language processing module. They reported that the system reaches 90% character accuracy performing in almost real-time on Pentium PC [**13**].

Large Vocabulary Continuous Speech Recognition in Greek: Corpus and an Automatic Dictation System: in this work the researchers presents the creation of the first Greek Speech Corpus and the implementation of a Dictation System for workflow improvement in the field of journalism. They evaluated the word error rate (WER) and the recognition time (xRT) for the three acoustic models: Unisex for the gender-independent model they have got 21.01% 6.16 xRT for Male gender dependent model they have got 19.27% WER and 5.78xRT for Female gender Dependent model they have got 20.85 WER and 5.85xRT. As the final step of their work, they developed a graphical user interface (GUI) in order to complete the Greek Dictation System, named Logotypografos1.0. This tool was created using the MFC C++ library for the graphical interface.

Speech recognizers for Amharic language

To the best of our knowledge there is no any research or project that is done on the applications of Amharic speech recognition for Dictation except only there is one work on command and control application which is done for controlling the functionalities of Microsoft office word processor which is done by [**4**]. We believe that this research work will put some initiation and motivation for the future researchers to focus and work on the applications of Amharic speech recognizers in addition to the other research works that are done before. Some of speech recognizer researches that are done before are:

Isolated Amharic Consonant-Vowel (CV) Syllable Recognition: The first work in Amharic speech recognition was that of Solomon Berhanu's work. He developed speaker dependent and speaker independent HMM-based prototype Amharic CV syllable recognition systems. He used Hidden Markov Model Toolkit (HTK) to develop the recognizers. The speaker dependent CV syllable recognizer recognized on the average 87.68% of the test data accurately. Speaker independent recognizer recognized on the average 72.75% of the test data correctly and an average of 49.21% accuracy was scored for two speakers who have no involvement in the training [**7**].

Sub-word based Amharic Word Recognition: after the work of Solomon [**7**], Kinfe Taddesse [**8**] developed HMM-based speaker dependent and speaker independent sub-word based prototype Amharic word recognizers. He considered three sub-word units - phoneme, tri-phone and consonant-vowel syllables - with the aim of identifying which sub-word unit is better for developing Amharic speech recognition system. Kinfe also used HTK in the course of developing the recognizers.

The speaker dependent phoneme based recognizer recognized on the average 84.80% of testsetI and 83.07% of testsetII correctly. Speaker dependent triphone based recognizer recognized 84.00% of testsetI accurately and 78.00%of testsetII. The speaker dependent CV syllable recognizer recognized 74% of testsetI and 60% of testsetII correctly. Since the performance of speaker dependent CV syllable based recognizer was poor, Kinfe did not develop speaker independent CV syllable based recognizer. Thus, he developed phone based and triphone based speaker independent recognizers. The phone based speaker independent recognizer recognized on the average 77.73% of testsetI and 75.33% of testsetII. The triphone based speaker independent recognizer has better performance than the phone based recognizer; it recognized 91.46 of testsetI and 77.87% of testsetII.

CHAPTER THREE

THE AMHARIC LANGUAGE

In this chapter we will discuss about the general description of Amharic language in related to Speech recognition application.

According to Voigt pointed on the work of [**14**] Amharic is a member of the Ethio-Semitic languages, which belong to the Semitic branch of the Afro-Asiatic super family, and uses the Ethiopic script commonly known as Ge'ez, which is an ancient language currently used for the liturgy of the Ethiopian Orthodox Church.

Amharic uses thirty-three consonant classes form the Ethiopic alphabet, of which twenty seven have unique sounds, these being characterized in terms of their sound creation (labial, dental, palatal, velar, and glottal) and/or their graphic symbols [**15**]. Each consonant has seven forms created by fusion of a consonant and a vowel-fused form of vocalic marker. According to Leslau sited on [**14**] there are a few other characters, called labiovelars that are created by rounding the lips when voicing consonants.

The word units of Amharic are: phoneme, morpheme, root, stem, and word. A phoneme represents a basic sound or unit of sound. Every glyph or consonant form is a phoneme or unit of word. A phoneme or collection of phonemes forms a morpheme, which is the smallest meaningful unit in word [**15**]. A morpheme can be free or bound, where a free morpheme can stand as a word on its own whereas a bond morpheme cannot. An Amharic root is a sequence of consonants, and is the basis for the derivation of verbs; a stem, on the other hand, is a consonant or consonant-vowel sequence. A stem can be free or bond: a free stem can stand as a word on its own whereas a bound stem has a bound morpheme affixed to it. A collection of phonemes or sounds creates a word, which can be as simple as a single morpheme or contain several of them.

Since to build large vocabulary speech recognition system it is recommended to use the word units of that language. This research work is proposed to work on the sub units of the Amharic language. The smallest unit of one language is its phone. Amharic has its own phonemes. Before describing Amharic phonetics, phonology, syntax and writing system it is appropriate to view the basic terms.

Phonetics is the study of speech sounds used in languages of the world. It is concerned with sounds of languages, how these sonds are articulated and how the hearer perceives them.

Phonetics is related to the science of acoustics in that it uses much of the techniques used by acoustics in the analysis of sound [**6**].

Phonology is the study of the sound patterns of a language. It describes the systematic way in which sounds are differently realized in different contexts, and how this system of sounds is related to the rest of the grammar. Phonology is concerned with how sounds are organized in a language. It endeavors to explain what these phonological processes are in terms of formal rules.

Morphology is the study of word formation and structure. It studies how words are put together from their smaller parts and the rules governing this process. The elements that are combined to form words are called morphemes. A morpheme is the smallest meaningful unit you can have in a language.

Syntax is the study of sentence structure. It attempts to describe what is grammatical in a particular language in terms of rules. Syntactic knowledge of a language is given to an ASRS as its language model.

1.1. Basics of Amharic Phonetics

Articulatory phonetics shows that characteristics of sound are determined by the position of the various articulators in the vocal tract and the state of the vocal cords [**6**]. Three aspects are used to show the phonetics of Amharic language: Voicing, Manner and Place of articulation.

Linguists decompose a spoken language into elements of linguistically distinct sounds called phonemes. According to Juang and Furui sited on [**8**] these phonemes are determined and classified according to their corresponding articulatory configurations.

According to the voicing aspect sounds classified as voiced and unvoiced. In the articulation of manner sounds are classified in the classes: Stops, Fricatives, and Approximants. Place of articulations includes Labial, Dental, Palatal, Velar and Glottal [**6**].

Sounds can also be classified as vowels and consonants. Vowel and consonant sounds are produced in fundamentally different ways. According to Clark on [**8**] while consonants are articulated with a substantial degree of obstruction in the oral cavity, vowels are produced with a relatively free airflow.

1.1.1. Articulation of Amharic Consonants

According to Voicing, Manner and Place of articulation the following summary describes Amharic consonants.

Manner of Articulation	Voicing	Place of Articulation				
		Labials	**Dentals**	**Palatals**	**Velars**	**Glottal**
Stops	Voiceless	ፕ [p]	ት[t]	ች[tʃ]	ክ[k]	እ[?]
	voiced	ብ[b]	ድ[d]	ጅ[dʒ]	ግ[g]	
	Glottalized	ጵ[p']	ጥ[t']	ጭ[tʃ']	ቅ[q]	
	Rounded					
Fricatives	Voiceless	ፍ[f]	ስ[s]	ሽ[ʃ]		ህ[h]
	voiced		ዝ[z]	ዥ[ʒ]		
	Glottalized		ጽ[s']			
	Rounded					
Nasals	voiced	ም[m]	ን[n]	ኝ[ŋ]		
Liquids	voiced		ል[l] ር[r]			
Semi-vowels	voiced	ው[w]			ይ[j]	

Table 3. 1: Categories of Amharic Consonants [adapted from [**6**]]

Furthermore, as described on the work of [**6**] Amharic has four labialized consonants which are classified as labio-Velar on the work of [**8**] that are pronounced with a slight rounding of the lips. These are ጐ [g^w], ኰ [k^w], ቈ [q^w], and ኍ [h^w].

1.1.2. Articulation of Amharic Vowels

According to Clark et al., on [**8**] vowels are open sounds, made largely by shaping the vocal tract rather than by interfering with the flow of air stream.

Vowels are most usefully described in terms of the position of the tongue as they are articulated. A vowel articulated with the body of the tongue relatively forward is classified as a *front vowel*; one made with the body of the tongue relatively high is a *high vowel*. Vowels produced with the body of tongue neither high nor low are called mid vowels. Vowels produced with the tongue body front are called front vowels while those made with the tongue body back are called *back* *vowels.* Those vowels made with the tongue body neither front nor back are called central vowels [Ibid].

Vowels accompanied by lip rounding as in (u and o) are called rounded vowels while the other vowels are called unrounded vowels [**8**]. Amharic has 7 (seven) vowels their description is given in the table below.

	Front/Unrounded	**Central/Unrounded**	**Back/Rounded**
High	ኢ[i]	እ[I]	ኡ[u]
Mid	ኤ[e]	ኧ[E]	ኦ[o]
Low		አ[a]	

Table 3. 2: Categories of Amharic vowels [adapted from [**8**]]

1.2. Amharic Writing System

The Amharic script is an abugida, and the graphs of the Amharic writing system are called *fidel(ፊደል).* Each character represents a consonant + vowel sequence, but the basic shape of each character is determined by the consonant, which is modified for the vowel. Some consonant phonemes are written by more than one series of characters: /አ/, /ሰ/, /ጸ /, and /ሀ/ (the last one has *four* distinct letter forms). This is because these *fidel* originally represented distinct sounds, but phonological changes merged them. The citation form for each series is the consonant + ኧ form, i.e. the first column of the *fidel*. [**16**].

There are disagreements among scholars whether the Amharic writing system is syllabic or alphabetic [**8**]. According to Clark et al., on [**8**] writing systems in which one symbol represents one syllable are called syllabic while writing systems in which one symbol represents one sound segment are called alphabetic. According to Bender, Cowley, Mullen on [**8**] argue that the Amharic writing system is syllabic while others Tadesse , Baye say that it is not syllabic.

From the point of view of speech recognition Amharic writing system has some limitations. According to Bender, et al., on [**8**] some of the limitations of the Amharic writing system are:

- There are several duplicate characters that are normally used interchangeably
  For example: [ሀ, ሃ, ሐ, ኻ, ሓ, ኃ],[ሰ, ሠ],[ጸ, ፀ],[አ, ዓ, ዐ, ኣ]

- It lacks a mechanism to mark gemination of consonants.

- There exists ambiguity between sixth-order symbols with and without vowel in different contexts.

Grammatical Arrangement

We will not make a detailed explanation of the Amharic language's grammatical arrangements as it is beyond the scope of this study. We will cover the top level grammatical and morphological structure of the language in this section. The Amharic language has been declared to have word categories as **ስም** (noun), **ግስ** (verb), **ቅፅል** (adjective), **ተውሳከ ግስ** (Adverb), **መስተዋድድ** (preposition), and **ተውላጠ ስም (**pronoun) [**17**].

Noun: a word will be categorized as a noun, if it can be pluralized by adding the suffix ኦች/ዎች and used as nominating something like person, animal, and so on [**15**].

Verb: any word which can be placed at the end of a sentence and which can accept suffixes as /ህ/,/ሁ/,/ሽ/, etc. which is used to indicate masculine, feminine, and plurality is classified as a verb.

Adjective: any word that qualify a noun or an adverb, which actually comes before a noun**(e.g. ጎበዝ ተማሪ)** and after an adverb **(በጣም ጎበዝ).** Other specific property of adjectives is, when pluralized, it will repeat the previous letter of the last letter for the word **(e.g. ረዥም--› ረዣዥም).**

Adverb: it will be used to qualify the verb by adding extra idea on the sentence. The Amharic adverbs are limited in number and include **ትናንት, ገና, ዛሬ, ቶሎ, ምንኛ, ክፉኛ, እንደገና, ጅልኛ,** and **ግምኛ.**

Preposition: preposition is a word which can be placed before a noun and perform adverbial operations related to place, time, cause and so on; which can't accept any suffix or prefix; and which is never used to create a new word. It includes **ከ፣ ለ፣ ወደ፣ ስለ፣እንደ…**

Pronoun: this category further can be divided as deictic specifier, which includes **ይህ, ያ, እሱ, እስዋ, እኔ, አንተ, አንቺ…**; quantitative specifier, which includes **አንድ, አንዳንድ, ብዙ, ጥቂት, በጣም…**; and possession specifier such as **የእኔ, የአንተ, የእሱ…**

The Amharic basic sentence is constructed from noun phrase and verb phrase **(ስማዊ ሀረግ+ ግሳዊ ሀረግ).**

Sentence= noun_phrase + Verb_Phrase.

For example, the sentence: **“ሁለት ትልልቅ ልጆች ትናንት በመኪና ወደ ጎጃም ሄዱ።”** have noun phrase **““ሁለት ትልልቅ ልጆች”** and verb phrase **“ትናንት በመኪና ወደ ጎጃም ሄዱ”.**

The statement **(አረፍተ-ነገር)** will have the noun phrase and verb phrase combinations. The noun phrase and the verb phrase further will be divided to different particles such as other sub noun phrase and verb phrase, noun, adjectives, specifier and so on.

Amharic punctuation marks and numerlas

The Amharic documents collected should be pre-processed before the succeeding ASR components perform further operations. The punctuation marks are here discussed as it will help in pre-processing documents.

Similarly, numerals have greater impact on ASR systems. Since numbers are stored in different formats, there should be some kind of standardization that will help for training acoustic model. If numbers can’t be normalized to one standard, a document that has different number representation will remain irrelevant for the training of acousitc model.

In Amharic, there are different punctuation marks used for different purposes. In the old scripture, a colon (**ሁለት ነጥብ**፡) has been used to separate two words. These days the two dots are replaced with whitespace. An end of a statement is marked with four dots **(አራት ነጥብ** ።) while **ነጠላ ሰረዝ**(፣ or ፥) is used to separate lists or ideas just like the comma in English.

In Amharic, numbers can be represented using Arabic symbols. It has also its own number representations, Ethiopic number representations. Similarly numbers can be represented in a word alphanumerically. Table 3.1 shows the Arabic, Amharic and alphanumeric representation of numbers [**18**].

Arabic	**Ethiopic**	**Alphanumeric**	**Arabic**	**Ethiopic**	**Alphanumeric**
1	፩	**አንድ**	20	፳	**ሃያ**
2	፪	**ሁለት**	30	፴	**ሰላሳ**
3	፫	**ሦስት**	40	፵	**አርባ**
4	፬	**አራት**	50	፶	**አምሳ/ሀምሳ**
5	፭	**አምስት**	60	፷	**ስልሳ/ስድሳ**

6	፮	ስድስት	70	፸	ሰባ
7	፯	ሰባት	80	፹	ሰማኒያ
8	፰	ስምንት	90	፺	ዘጠና
9	፱	ዘጠኝ	100	፻	መቶ
10	፲	አስር	1000	፲፻	ሺ/ሽህ

Table 3. 3: Number Representations in Amharic

Since numbers in Amharic during speech is pronunced not like the exact representaion it must be pre-processed as it is shown on table 3.3.

Fraction	Amharci representation	Ordinals	Representation
½	ግማሽ	1st	አንደኛ/ቀዳማዊ
1/3	ሲሶ	2nd	ሁለተኛ/ዳግማዊ
¼	ሩብ/እርቦ	3rd	ሶስተኛ/ሳልስ
2/3	ሁለት ሲሶ / ሁለት ሶስተኛ	4th	አራተኛ/ራብዕ
¾	ሶስት አራተኛ		
1/10	አስራት		
2X	እጥፍ	9th	ዘጠንኛ/ዘጠነኛ
2.X	ሁለት ነጥብ X	10th	አስረኛ

Table 3. 4: Amharic fraction and Ordinal representation

On the above table we can easily observe that in the collected text if there exist fraction and ordinal representations are found it is normalised into the equivalent Amharic representation.

Gemination

As in most other Ethiopian Semitic languages, gemination is contrastive in Amharic. That is, consonant length can distinguish words from one another; for example, አለ (*alä)* 'he said', አለ (*allä)* 'there is'; ይመታል (*yəmätall)* 'he hits', ይመታል (*yəmmättall)* 'he is hit'. Gemination is not indicated in Amharic orthography, but Amharic readers typically do not find this to be a problem. This property of the writing system is analogous to the vowels of Arabic and Hebrew or the tones of many Bantu languages, which are not normally indicated in writing. The noted Ethiopian novelist Haddis Alemayehu, who was an advocate of Amharic orthography reform, indicated gemination in his novel ፍቅር እስከ መቃብር (*Fəqər Əskä Mäqabər)* by placing a dot above the characters whose consonants were geminated, but this practice is rare [**16**]. Forexmple: “ እኔና’ : እግዚአብሄር : የምናውቀው : እርስዎ : የማያውቁት : አለ’ : አባቴ ? (ፍቅር እስከ መቃብር page ፲፪) (Inenna IgIziabIherI yEmInawIqEwI IrIsIwo yEmayawIqutI *allä* abate?) ‘me and God knows you don’t know there is father?’ grammatically the English translation is wrong but we make it direct word by word translation to help us describe the difference when the words are geminated and not. From the above sentence there are two words that are geminated the word እኔና(Inenna) and አለ(*allä*) from the two words the consonant ‘ና’ in the word እኔና is geminated and it makes the word to give another meaning እኔና’(Inenna) ‘me and’ when it is geminated but if it was not geminated it will give another meaning እኔና (Inena) ‘me come’ and the word አለ’(*allä*) ‘there is’ is geminated but if it was not geminated it will give another meaning አለ (*alä)* 'he said' and it will give another meaning in the sentence. The sentenece will be read as ‘me come God knows you don’t know he said father’.

When we Geminate the words፡ : “እኔና’: እግዚአብሄር : የምናውቀው : እርስዎ : የማያውቁት : አለ’ : አባቴ ?”

Tanslated equivalent result: ”is there anything that me and God knows that you don’t know father?”Or we can say “father is there anything that me and God knows that you don’t know?”

When we don’t geminate the words: “እኔና : እግዚአብሄር : የምናውቀው : እርስዎ : የማያውቁት : አለ : አባቴ?”

Translated equivalen result : “my father says me come God knows that you don’t know”

In the above example when we don’t gemminate the words in the sentence it will create grammatical error both in Amharic and the translated result. The difference between the original and translated result is that the ortographic representation of Amharic sentence is the same whether we geminate it or not that is why Haddis Alemayehu use dot on the top of the gemminated consonant symbol to mark the geminated consonant.

As we can see from the above examples since gemination results in degradation of speech recognition results it is better to use the rule of gemination by inserting special symbols to identify the geminated words that are uttered. But since our transcribed sentence does not have any geminated word we did not consider gemination in our system.

CHAPTER FOUR

CMU SPHINX: THE RECOGNIZER LIBRARY

In this chapter we will try to discuss the tools that are used to build the entire system. The tools that are used for Acoustic model building, language model building and the decoder for speech recognition results. The CMUSphinx tool kit is a speech recognition tool kit with various tools used to build speech applications; it has a number of packages for different tasks and applications.

4.1. History of CMU Sphinx

The history of the speech recognizer's development shows that there are two major inter-related goals of speech recognizer design. One is to optimize the accuracy of speech recognizer. One is to allow a known imperfect speech recognizer to be used in a practical scenario. Usually, this practical scenario imposes constraints to the design of the recognizer such that the recognizer has to run in a limited amount of computation resource (such as amount of random access memory and the computation power). For example, NIST evaluation, a kind of competition between research sites, requires each site to provide the best performing recognition system.

Until this day's sphinx is released in different versions based on the release version numbers we listed the recognizer libraries below.

Sphinx I: was one of the world's first user independent, high performances ASR (Automatic Speech Recognizer) developed by Dr. Kai-Fu Lee in 1987. Sphinx-I support simple word-pair grammars and generalized tri-phones. It was written in the C programming language and was, as described above, the world's first high performance speaker independent continuous speech recognition system. It was based on the viable technology of discrete HMMs, i.e. HMMs that used discrete distributions or simple histograms to model the distributions of the measurements of speech sounds. Since speech itself is a signal that can take a set of values that are continuous in some range, the modeling paradigm required a quantization of speech into a discrete set of symbols. Sphinx-1 accomplished this using a vector quantization algorithm. A primary sequence of LPC-cepstral vectors was derived from the speech signals, and from this sequence, two secondary sequences of difference parameter vectors were derived. The vector quantization algorithm computed codebooks from the vectors in the sequences, and replaced each vector by codeword indices from the codebooks. HMMs were then trained with these sequences of quantized vectors. During recognition, incoming speech was also converted into sequences of codeword indices using the same codebooks.

The sound units that the system modeled with discrete HMMs were called generalized tri-phones. A tri-phone is simply a distinct phonetic unit labeled with its immediately adjacent phonetic contexts. Tri-phones were, and remain, one of the most effective innovations in modeling speech sounds in HMM-based systems. In a system based on generalized tri-phones, a number of tri-phones are modeled by a common HMM. In Sphinx-I, this was done for logistical reasons, since computers in those days were not powerful enough to handle all tri-phones separately. Also, the large stored databases required to train the vast repository of tri-phones found in normal everyday speech did not exist.

In a recognition system, the actual process of recognition is guided by a grammar or language model that encapsulates prior knowledge about the structure of the language. Sphinx-I used a simple word-pair grammar, that indicates which word pairs are permitted in the language and which are not.

Sphinx-II: Within five years of its introduction, technology based on semi-continuous HMMs was ready to be used, and much of it had been developed at CMU. Sphinx-II came into existence in 1992, and was again a pioneer system based on the new technology of semi-continuous HMMs. The system was developed by Xuedong Huang at CMU, then a postdoctoral researcher working with Professor Raj Reddy. Like Sphinx-I, it was written in the C programming language.

The essence of semi-continuous HMM based technology was that speech was no longer required to be modeled as a sequence of quantized vectors. State distributions of a semi-continuous HMM were modeled by mixtures of Gaussian densities. Rather than the speech vectors themselves, it was the parameters of the Gaussian densities that were quantized. Sphinx-II used 4 parallel feature streams, three of which were secondary streams derived from a primary stream of 13-dimensional cepstral vectors computed from the speech signal. All components of the feature streams were permitted to take any real value. For each feature stream, Gaussian density parameters were allowed to take one of 256 values (the number 256 being dictated by the largest number represent-able by an 8-bit number). The actual values of the 256 sets of parameters were themselves learned during training.

In order to achieve real-time speeds, the system was hardwired to use 5-state Bakis topology HMMs for all sound units. Each sound unit was a tri-phone, but unlike Sphinx-1, this system did not use generalized tri-phones. Instead, state distributions of the HMMs for the tri-phones were tied, i.e. states of the HMMs for several tri-phones were constrained to have the same distribution. The state tying was performed using decision trees. This technique of sharing distribution parameters at the state level, invented by Mei-Yuh Hwang, then a doctoral student at CMU under the supervision of Professor Raj Reddy, was yet another major milestone in HMM-based speech recognition technology, as it made it possible to train large numbers of parameters for a recognition system with relatively modest amounts of data.

Sphinx-II also improved upon its predecessor sphinx-1 in being able to use statistical N-gram language models during search, where N could be any number, in principle. An N-gram language model represents the probability of any word in the language, given the N-1 prior words in a sentence, and is significantly superior to the word-pair grammar used by Sphinx-I as a representation of the structure of a language.

The semi-continuous HMM based Sphinx-II required much greater computation to perform recognition than Sphinx-1 did, and consequently the early

Sphinx-II decoders (the part of the recognizer that actually performs the recognition) were relatively slow, and took longer than real time on the hardware of the day. These were soon replaced by the FBS-8 decoder, written by Mosur Ravishankar of CMU which used a lexical-tree-based search strategy that represents the recognizer vocabulary as a tree of phonemes. This lextree decoder could perform recognition in real time on standard computers. The name FBS-8 recalls an interesting history of this decoder. FBS-8 stands for Fast Beam Search version 8. This decoder was actually the 8th in a string of quite different decoders. In fact its most successful predecessor, FBS-6, used a ”flat” search strategy that represents each word in the recognizer’s lexicon separately, and thus was coded very differently from its immediate successor FBS-7, which was lextree based. Sphinx-II was able to achieve an accuracy of about 9030,000 word vocabulary (e.g. Wall Street Journal) recognition task.

Sphinx-III: The next milestone in recognition technology was marked by the creation of Sphinx-III. It was developed four years after Sphinx-II was unveiled and incorporated technology that allowed the modeling of speech with continuous density HMMs in a fully continuous vector space. No vector space quantization were required at any level. Naturally, this resulted in better recognition performance as compared to its predecessor, Sphinx-II. However, at the same time due to statistical requirements imposed by this technology, large amounts of data were required to train such HMMs well. Fortunately, hardware improvements in computers allowed the storage and processing of large amounts of data by this time. Sphinx-III was thus a very viable system at the time it was developed. It had many new features, but was at the same time designed to be backwardly compatible with Sphinx-II. This meant that it could handle both semi-continuous and fully-continuous HMMs. The system was written in the C programming language, and developed jointly by two people at CMU: its HMM-training modules were developed by Eric Thayer, and its decoding, modules were developed by Mosur Ravishankar. Sphinx-III was more versatile in its feature-type handling capacity than its predecessors. The primary feature stream was no longer constrained to be 13 dimensional. It could use single and 4-stream feature sets. The HMM topology was user-specifiable. Like its predecessors, it used tri-phone HMMs, with state sharing information obtained using decision trees.

Sphinx-II currently has two different decoders, both written by Mosur Ravishankar. Both decoders use statistical N-gram language models with N¡=3 during search. They differ in several respects, including their search strategies. One decoder, generally referred to as S3.0, uses the flat search strategy. The second decoder has two variants, usually referred to a S3.2 and S3.3. These differ in their ability to handle streaming input and return partial hypotheses during decoding. S3.2 does not have these capabilities while S3.3 does. Both variants use a lex-tree based search strategy. The use of lex-trees makes these versions much faster than S3.0, and their speed is further enhanced through a sub vector quantization strategy for Gaussian selection. Both decoders, and the trainer, have hardwired requirements to some degree, though to a much lesser extent than the Sphinx-II system. Such requirements are often an essential part of the engineering that goes into implementing speech recognition systems, and are necessitated by the limitations of the computational platforms of the day.

Sphinx-IV: The years after the development of Sphinx-III saw two major advancements in speech research. First, multimodal speech recognition came into existence, with the development of associated algorithms. Second, it became feasible for speech recognition systems to be deployed pervasively in wide-ranging, even mobile, environments.

In multimodal speech recognition, the information in the speech signal is augmented by evidence from other concurrent phenomena, such as gestures, expressions, etc. Although these information streams are related, they differ both in the manner in which they must be modeled and the range of contexts that they capture. The contexts that affect the evidence for any sound can often be highly asymmetric and time dependent. It therefore becomes necessary for the structure and the distributions of the HMMs, and the context dependencies of the sounds they represent, to be variable and user controllable. Also, wide ranging practical applications require the system to function in widely varying language scenarios, and so it also becomes necessary to be able to use language models other than statistical N-grams, such as context free grammars (CFGs), finite state automata (FSAs), or stochastic FSAs, which might be more optimal for the task domain at hand. Statistical language models are just one of a set of different types of well-researched models of human language.

With these considerations, by the beginning of the year 2000, it was clear that Sphinx-III would need to be upgraded. Sphinx-III could only use tri-phone context sound units and required a uniform HMM topology for all sound units. It also required a uniform type of HMM, in the sense that the basic type of statistical density used by each HMM would have to be the same, with the same number of modes, for all sound units, regardless of the amount of training data or the type of feature being used for recognition. Sphinx-III moreover was only partially ready for multimodal speech recognition - although it could use multiple feature streams, it could combine them only at the state level.

Additionally, in the time since the development of Sphinx-III, the world had seen an exponential growth and innovation in computational resources, both hardware and software, available to the general user. Internet, markup languages, programming languages which augment and integrate smoothly with markup languages and the Internet etc. have developed rapidly. Even better recognition algorithms are now available, and internet based technology permits new ways in which these can be integrated into any real medium. High flexibility and high modularity are the norms by which today's pervasive software is being developed.

Thus, in response to the demands and offers of new technology available to speech recognition systems, and to the demands of the rapidly maturing area of multimodal recognition, Sphinx-4 was initiated by a team of researchers in 2001, joining forces from Carnegie Mellon University, Mitsubishi Electric Research Labs and SUN Microsystems. In 2002, researchers from Hewlett Packard Inc. joined the effort. Sphinx-4 is written entirely in the JAVA programming language, which is an extremely powerful language for Internet based applications. It provides vastly superior interfacing capabilities. The system is being developed on an open platform, and is available to researchers, developers, and commercial organizations freely at any stage of development. It is in principle an effort of the world community.

4.2. Why Sphinx

As a developer, the major reason to use Sphinx is because Sphinx currently is the most complete archive which is in Berkeley's license. We are aware of that other very powerful toolkits, are also open-sourced. Unfortunately, some of their licenses disallow to use their code in any project. It may also disallow to use the code and distribute them later in the project.

Another aspect of Sphinx is that based on Sphinx and its related resources, and with enough knowledge, allows building a lot of different types of applications. By speech applications, mean things like dictation, automatic speech server. More importantly, the most common denominator or all these applications, i.e the recognizer is open-sourced in CMU Sphinx. It also gives a lot of bundling resource such as dictionary, phone list from CMU's speech archive. This is something very important if you want to have full-control of speech recognition system.

As a researcher and if the interest is in building HMM-based speech recognition, there are several reasons why we want to use Sphinx 3 and Sphinx 4 as our research tools.

First of all Sphinx assumes that GMM computation and the search process are separate. The important consequence is that we could carry out two different types of research without conflicting each others. For example, we could try to device a new observation probability without touching the source code of the search routine. At the same time, we could also build a new search algorithm without thinking of the GMM computation. This gives us a better advantage in speech recognition research.

The second thing is in training. Most of the time when we were doing modeling research, what we would like to change is the algorithm of estimation. SphinxTrain's Baum-Welch's algorithm solves this problem in two stages. It first dumps the posterior probabilities count in a separate file and can be easily readable by libraries of SphinxTrain. This advantage can greatly shorten our research time from weeks to dates.

The third reason is in the code-clarity; later recognizers such as Sphinx 3 and Sphinx 4 have put readability of the code as one of the most important design goal. Many researchers, not only they want to use the recognizer as a tool. They also want to change the code to suit their purpose. Sphinx 3, for example, has only 4 layers of code from main () to the GMM computation. These become a good advantage for researchers who like to make changes in a speech recognizer.

The final reason is in speed; all Sphinx's decoders are very fast because the speed of the recognizer is always one important concern. We will not only able to build a research system easily but quickly. This is usually an important factor why we would like to use Sphinx.

4.3. Acoustic Modeling Toolkit

SphinxTrain: A good decoder cannot live without a good trainer. SphinxTrain's trainer has an equally interesting development history as in the decoder.

At the age when Sphinx-I and Sphinx-II is created and widely used internally CMU. Experiments of acoustic modeling and language modeling are usually done by "researcher's code". That means individual researchers hack out a tools using C or perl and full fill the aim of training. The obvious problem is data formats might not be exchangeable and caused a lot of problems in advancing speech recognition research.

Two skillful programmers occurred in CMU who solved this problem. First is Dr. Ravi Mosur who is the author of sphinx-II's recognizer and sphinx-III's recognizer(s). We already mentioned him in the Sphinx's history. The second one is Dr. Eric Thayer who is the author of SphinxTrain. At the beginning, the two programmers agreed with the data and model format of the two tools and they start to work on the decoder and trainer separately (Ravi on decoder, Eric on trainer.) At the end, Eric was able to build a set of tools which are essential to the training task. It became the pre-body of SphinxTrain.

A lot of refinement of the trainer was later done by Dr. Rita Singh and PhD students and staffs in robust group led by Prof. Rich Stern. For example, the current version of decision tree-based state-tying for continuous HMM was realized and refined by Rita. She also proposed an novel way to automatically create linguistic questions for decision-tree-base state tying. Bhiksha Raj did a lot of changes to make the s3 aligner program to accept optional silence deletion.

Currently, these research's results in last 10 years are slowly incorporated by the current Sphinx-III maintainers, Arthur Chan and Evandro Gouvea. A java version of the trainer is also under development in Sphinx-IV. These developments will no doubt enrich the software of Sphinx.

4.4. Language Modeling Tool kit

CMU LM Toolkit: The version 1 of CMU LM toolkit is created by Roni Rosenfield. Later Philip Clarkson in Cambridge University improves its caching capability and data structure so that it can handle more data with less memory. It becomes V2 of the program and is open-sourced since 1996.

SRILM: SRILM is a toolkit for building and applying statistical language models (LMs), primarily for use in speech recognition, statistical tagging and segmentation, and machine translation. It has been under development in the SRI Speech Technology and Research Laboratory since 1995.

4.5. Decoding Toolkit

4.5.1. Sphinx-4 Framework

Sphinx-4 is a flexible, modular and pluggable framework to help foster new innovations in the core research of hidden Markov model (HMM) recognition systems. The design of Sphinx-4 is based on patterns that have emerged from the design of past systems as well as new requirements based on areas that researchers currently want to explore. To exercise this framework, and to provide researchers with a "research-ready" system, Sphinx-4 also includes several implementations of both simple and state-of-the-art techniques [**19**].

First and foremost, Sphinx-4 is a modular and pluggable framework that incorporates design patterns from existing systems, with sufficient flexibility to support emerging areas of research interest. The framework is modular in that it comprises separable components dedicated to specific tasks, and it is pluggable in that modules can be easily replaced at runtime. To exercise the framework, and to provide researchers with a working system, Sphinx-4 also includes a variety of modules that implement state-of-the-art speech recognition techniques [**19**].

The Sphinx-4 framework has been designed with a high degree of flexibility and modularity. Figure 4.1 shows the overall architecture of the system. Each labeled element in Figure 5.1 represents a module that can be easily replaced, allowing researchers to experiment with different module implementations without needing to modify other portions of the system.

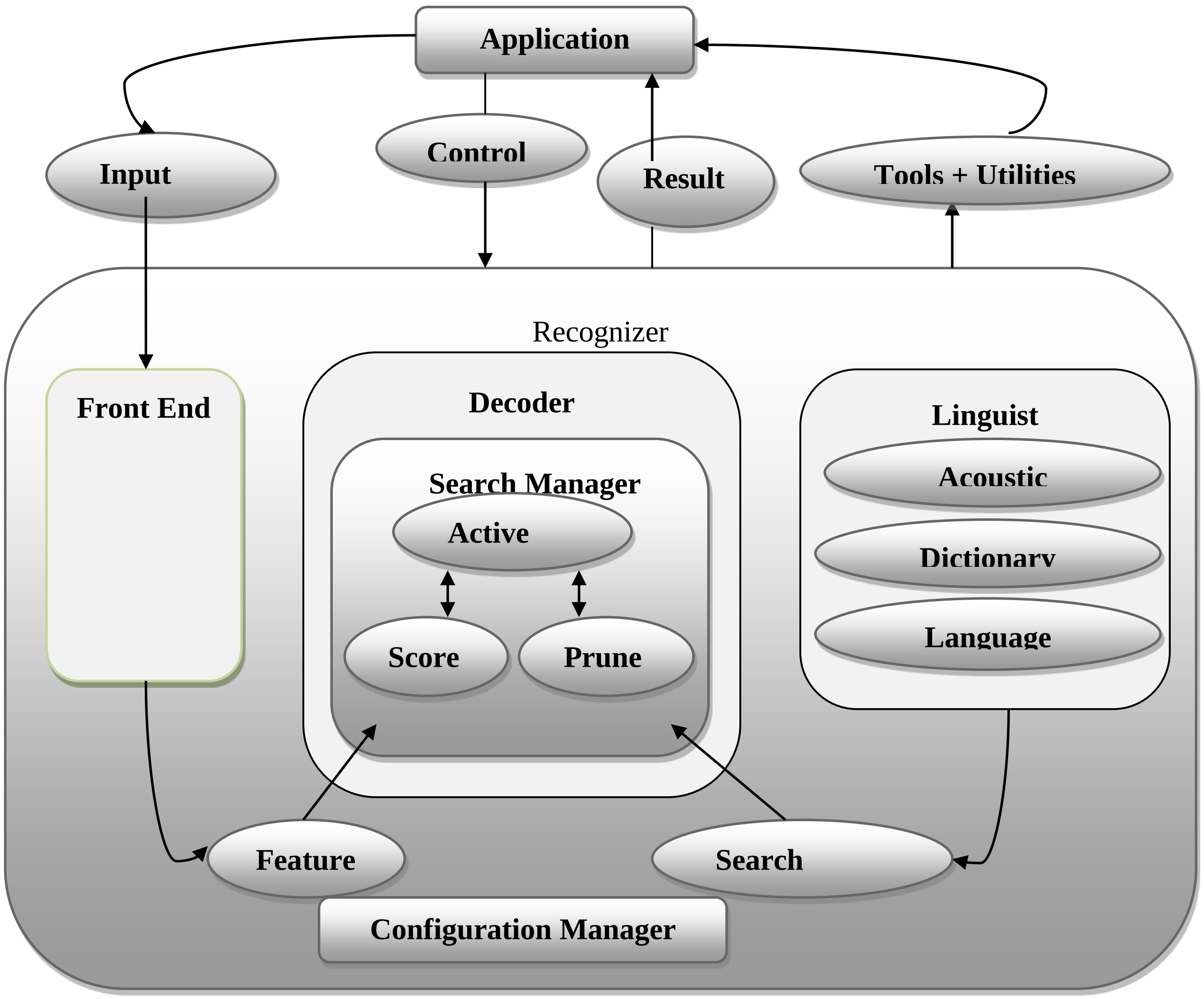

Figure 4.1: Sphinx-4 Decoder Framework. The main blocks are the Front End, the Decoder, and the Linguist. Supporting blocks include the Configuration Manager and the Tools blocks. The communication between the blocks, as well as communication with an application, [Adapted from [**19**]]

There are three primary modules in the Sphinx-4 framework: the *Front End*, the *Decoder*, and the *Linguist*. The Front End takes one or more input signals and parameterizes them into a sequence of Features. The Linguist translates any type of standard language model, along with pronunciation information from the Dictionary and structural information from one or more sets of Acoustic Models, into a Search Graph. The Search Manager in the Decoder uses the Features from the Front End and the Search Graph from the Linguist to perform the actual decoding, generating Results. At any time prior to or during the recognition process, the application can issue Controls to each of the modules, effectively becoming a partner in the recognition process [**19**].

A. *Front End*

The purpose of the Front End is to parameterize an Input signal (e.g., audio) into a sequence of output Features. As illustrated in Figure 4.2, the Front End comprises one or more parallel chains of replaceable communicating signal processing modules called Data Processors. Supporting multiple chains permits simultaneous computation of different types of parameters from the same or different input signals. This enables the creation of systems that can simultaneously decode using different parameter types, such as MFCC and PLP, and even parameter types derived from non-speech signals such as video [**19**].

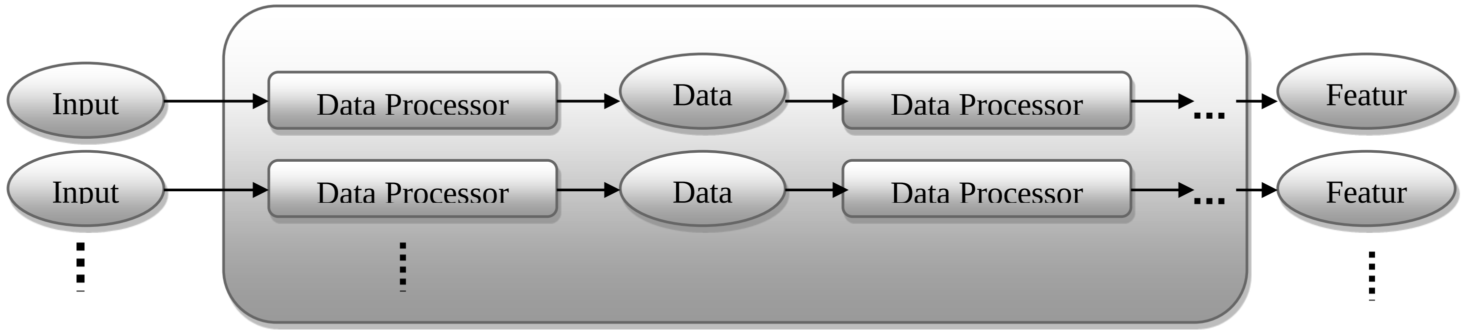

Figure 4. 2: Sphinx-4 Front End. The Front End comprises one or more parallel chains of communicating Data Processors. [Adapted from [**19**]]

B. Linguist

The Linguist generates the Search Graph that is used by the decoder during the search, while at the same time hiding the complexities involved in generating this graph. As is the case throughout Sphinx-4, the Linguist is a pluggable module, allowing people to dynamically configure the system with different Linguist implementations.

A typical Linguist implementation constructs the Search Graph using the language structure as represented by a given Language Model and the topological structure of the Acoustic Model (HMMs for the basic sound units used by the system).The Linguist may also use a Dictionary (typically a pronunciation lexicon) to map words from the Language Model into sequences of Acoustic Model elements. When generating the Search Graph, the Linguist may also incorporate sub-word units with contexts of arbitrary length, if provided.

By allowing different implementations of the Linguist to be plugged in at runtime, Sphinx-4 permits individuals to provide different configurations for different system and recognition requirements. For instance, a simple numerical digits recognition application might use a simple Linguist that keeps the search space entirely in memory. On the other hand, a dictation application with a 100K word vocabulary might use a sophisticated Linguist that keeps only a small portion of the potential search space in memory at a time.

The Linguist itself consists of three pluggable components: the Language Model, the Dictionary, and the Acoustic Model, which are described in the following sections [**19**].

I. Language Model

The Language Model module of the Linguist provides word-level language structure, which can be represented by any number of pluggable implementations. These implementations typically fall into one of two categories: graph-driven grammars and stochastic N-Gram models. The graph-driven grammar represents a directed word graph where each node represents a single word and each arc represents the probability of a word transition taking place. The stochastic N-Gram models provide probabilities for words given the observation of the previous n-1 words.

The Sphinx-4 Language Model implementations support a variety of formats, including the following:

• *Simple Word List Grammar:* defines a grammar based upon a list of words. An optional parameter defines whether the grammar "loops" or not. If the grammar does not loop, then the grammar will be used for isolated word recognition. If the grammar loops, then it will be used to support trivial connected word recognition that is the equivalent of a unigram grammar with equal probabilities.

• *JSGF Grammar:* supports the Java ^{TM} Speech API Grammar Format (JSGF), which defines a BNF-style, platform independent and vendor-independent Unicode representation of grammars.

• *LM Grammar:* defines a grammar based upon a statistical language model. LM Grammar generates one grammar node per word and works well with smaller unigram and bigram grammars of up to approximately 1000 words.

• *FST Grammar:* supports a finite-state transducer (FST) in the ARPA FST grammar format.

• *Simple N-Gram Model:* provides support for ASCII N-Gram models in the ARPA format. The Simple N-Gram Model makes no attempt to optimize memory usage, so it works best with small language models.

• *Large Trigram Model:* provides support for true N-Gram models generated by the CMU-Cambridge Statistical Language Modeling Toolkit. The Large Trigram Model optimizes memory storage, allowing it to work with very large files of 100MB or more.

II. Dictionary

The Dictionary provides pronunciations for words found in the Language Model. The pronunciations break words into sequences of sub-word units found in the Acoustic Model. The Dictionary interface also supports the classification of words and allows for a single word to be in multiple classes.

Sphinx-4 currently provides implementations of the Dictionary interface to support the CMU Pronouncing Dictionary. The various implementations optimize for usage patterns based on the size of the active vocabulary. For example, one implementation will load the entire vocabulary at system initialization time, whereas another implementation will only obtain pronunciations on demand.

III. Acoustic Model

The Acoustic Model module provides a mapping between a unit of speech and an HMM that can be scored against incoming features provided by the Front End. As with other systems, the mapping may also take contextual and word position information into account. For example, in the case of tri-phones, the context represents the single phonemes to the left and right of the given phoneme, and the word position represents whether the tri-phone is at the beginning, middle, or end of a word (or is a word itself). The contextual definition is not fixed by Sphinx-4, allowing for the definition of Acoustic Models that contain allophones as well as Acoustic Models whose contexts do not need to be adjacent to the unit.

Typically, the Linguist breaks each word in the active vocabulary into a sequence of context-dependent sub-word units. The Linguist then passes the units and their contexts to the Acoustic Model, retrieving the HMM graphs associated with those units.

It then uses these HMM graphs in conjunction with the Language Model to construct the Search Graph.

Unlike most speech recognition systems, which represent the HMM graphs as a fixed structure in memory, the Sphinx-4 HMM is merely a directed graph of objects. In this graph, each node corresponds to an HMM state and each arc represents the probability of transitioning from one state to another in the HMM. By representing the HMM as a directed graph of objects instead of a fixed structure, an implementation of the Acoustic Model can easily supply HMMs with different topologies. For example, the Acoustic Model interfaces do not restrict the HMMs in terms of the number of states, the number or transitions out of any state, or the direction of a transition (forward or backward). Furthermore, Sphinx-4 allows the number of states in an HMM to vary from one unit to another in the same Acoustic Model.

Each HMM state is capable of producing a score from an observed feature. The actual code for computing the score is done by the HMM state itself, thus hiding its implementation from the rest of the system, even permitting differing probability density functions to be used per HMM state. The Acoustic Model also allows sharing of various components at all levels. That is, the components that make up a particular HMM state such as Gaussian mixtures, transition matrices, and mixture weights can be shared by any of the HMM states to a very fine degree.

As with the rest of Sphinx-4, individuals can configure Sphinx-4 with different implementations of the Acoustic Model based upon their needs. Sphinx-4 currently provides a single Acoustic Model implementation that is capable of loading and using acoustic models generated by the Sphinx-3 trainer.

IV. Search Graph

Even though Linguists may be implemented in very different ways and the topologies of the search spaces generated by these Linguists can vary greatly, the search spaces are all represented as a Search Graph. Illustrated by example in Figure 4.3., the Search Graph is the primary data structure used during the decoding process.

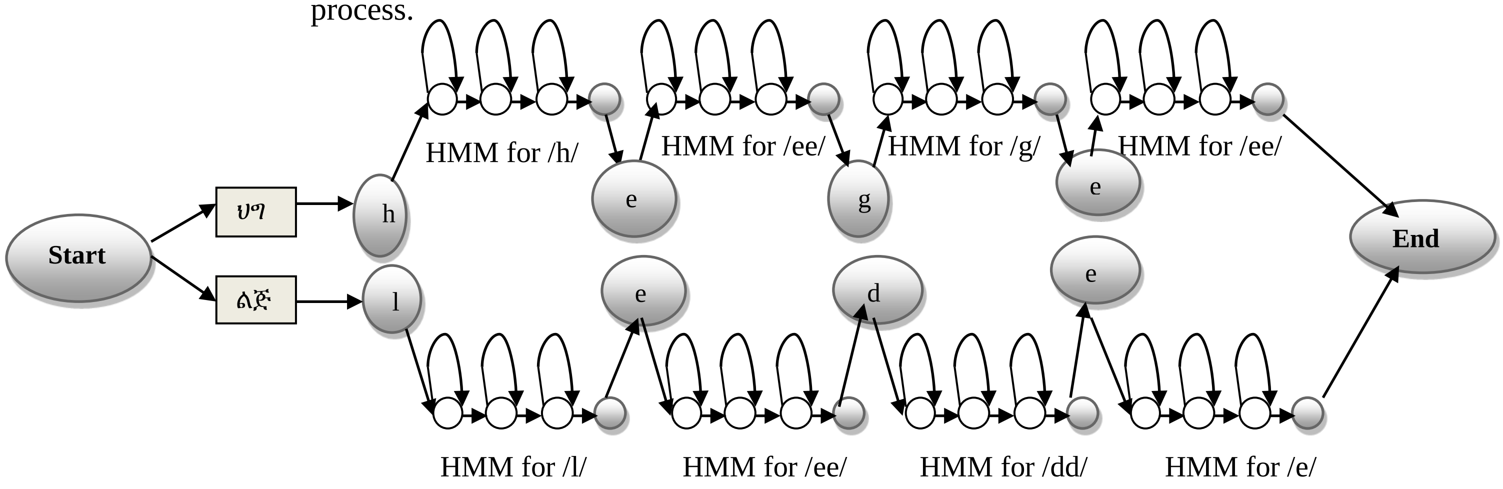

Figure 4. 3: Example Search Graph. The Search Graph is a directed graph composed of optionally emitting Search States and Search State Arcs with transition probabilities. Each state in the graph can represent components from the Language Model (words in rectangles)

The graph is a directed graph in which each node, called a Search State, represents either an emitting or a non-emitting state. Emitting states can be scored against incoming acoustic features while non-emitting states are generally used to represent higher-level linguistic constructs such as words and phonemes that are not directly scored against the incoming features. The arcs between states represent the possible state transitions, each of which has a probability representing the likelihood of transitioning along the arc.

The Search Graph interface is purposely generic to allow for a wide range of implementation choices, relieving the assumptions and hard-wired constraints found in previous recognition systems. In particular, the Linguist places no inherent restrictions on the following:

• Overall search space topology

• Phonetic context size

• Type of grammar (stochastic or rule based)

• N-Gram language model depth

A key feature of the Search Graph is that the implementation of the Search State need not be fixed. As such, each Linguist implementation typically provides its own concrete implementation of the Search State that can vary based upon the characteristics of the particular Linguist. For instance, a simple Linguist may provide an in-memory Search Graph where each Search State is simply a one-to-one mapping onto the nodes of the in-memory graph. A Linguist representing a very large and complex vocabulary, however, may build a compact internal representation of the Search Graph. In this case, the Linguist would generate the set of successor Search States by dynamically expanding this compact representation on demand.

The manner in which the Search Graph is constructed affects the memory footprint, speed, and recognition accuracy. The modularized design of Sphinx-4, however, allows different Search Graph compilation strategies to be used without changing other aspects of the system. The choice between static and dynamic construction of language HMMs depends mainly on the vocabulary size, language model complexity and desired memory footprint of the system, and can be made by the application.

V. *Implementations*

As with the Front End, Sphinx-4 provides several implementations of the Linguist to support different tasks.

The *Flat Linguist* is appropriate for recognition tasks that use context-free grammars (CFG), finite-state grammars (FSG), finite-state transducers (FST) and small N-Gram language models. The Flat Linguist converts any of these external language model formats into an internal Grammar structure. The Grammar represents a directed word graph where each Grammar Node represents a single word, and each arc in the graph represents the probability of a word transition taking place. The Flat Linguist generates the Search Graph directly from this internal Grammar graph, storing the entire Search Graph in memory. As such, the Flat Linguist is very fast, yet has difficulty handling grammars with high branching factors.

The *Dynamic Flat Linguist* is similar to the Flat Linguist in that is appropriate for similar recognition tasks. The main difference is that the Dynamic Flat Linguist dynamically creates the Search Graph on demand, giving it the capability to handle far more perplex grammars. With this capability, however, comes a cost of a modest decrease in run time performance.

The *Lex-Tree Linguist* is appropriate for large vocabulary recognition tasks that use large N-Gram language models. The order of the N-Grams is arbitrary, and the Lex Tree Linguist will support true N-Gram decoding. The Lex Tree Linguist organizes the words in a lex-tree, a compact method of representing large vocabularies. The Lex Tree Linguist uses this lex tree to dynamically generate Search States, enabling it to handle very large vocabularies using only a modest amount of memory. The Lex Tree Linguist supports ASCII and binary language models generated by the CMU-Cambridge Statistical Language Modeling toolkit or SRILM.

C. *Decoder*

The primary role of the Sphinx-4 Decoder block is to use Features from the Front End in conjunction with the Search Graph from the Linguist to generate Result hypotheses. The Decoder block comprises a pluggable Search Manager and other supporting code that simplifies the decoding process for an application. As such, the most interesting component of the Decoder block is the Search Manager.

The Decoder merely tells the Search Manager to recognize a set of Feature frames. At each step of the process, the Search Manager creates a Result object that contains all the paths that have reached a final non-emitting state. To process the result, Sphinx-4 also provides utilities capable of producing a lattice and confidence scores from the Result. Unlike other systems, however, applications can modify the search space and the Result object in between steps, permitting the application to become a partner in the recognition process.

Like the Linguist, the Search Manager is not restricted to any particular implementation. For example, implementations of the Search Manager may perform search algorithms such as frame-synchronous Viterbi, A*, bi-directional, and so on.

Each Search Manager implementation uses a token passing algorithm as described by Young [24]. A Sphinx-4 token is an object that is associated with a Search State and contains the overall acoustic and language scores of the path at a given point, a reference to the Search State, a reference to an input Feature frame, and other relevant information. The Search State reference allows the Search Manager to relate a token to its state output distribution, context-dependent phonetic unit, pronunciation, word, and grammar state. Every partial hypothesis terminates in an active token.

As illustrated in Figure 4.1, implementations of a Search Manager may construct a set of active tokens in the form of an Active List at each time step, though the use of an Active List is not required. As it is a common technique, however, Sphinx-4 provides a sub-framework to support Search Managers composed of an Active List, a Pruner and a Scorer.

The Search Manager sub-framework generates Active Lists from currently active tokens in the search trellis by pruning using a pluggable Pruner. Applications can configure the Sphinx-4 implementations of the Pruner to perform both relative and absolute beam pruning. The implementation of the Pruner is greatly simplified by the garbage collector of the Java platform. With garbage collection, the Pruner can prune a complete path by merely removing the terminal token of the path from the Active List. The act of removing the terminal token identifies the token and any unshared tokens for that path as unused, allowing the garbage collector to reclaim the associated memory.

The Search Manager sub-framework also communicates with the Scorer, a pluggable state probability estimation module that provides state output density values on demand. When the Search Manager requests a score for a given state at a given time, the Scorer accesses the feature vector for that time and performs the mathematical operations to compute the score. In the case of parallel decoding using parallel acoustic models, the Scorer matches the acoustic model set to be used against the feature type.

The Scorer retains all information pertaining to the state output densities. Thus, the Search Manager need not know whether the scoring is done with continuous, semi-continuous or discrete HMMs. Furthermore, the probability density function of each HMM state is isolated in the same fashion. Any heuristic algorithms incorporated into the scoring procedure for speeding it up can also be performed locally within the scorer. In addition, the scorer can take advantage of multiple CPUs if they are available.

The current Sphinx-4 implementation provides pluggable implementations of Search Managers that support frame synchronous Viterbi, Bushderby, and parallel decoding:

• *Simple Breadth First Search Manager:* performs a simple frame synchronous Viterbi search with pluggable Pruner that is called on each frame. The default Pruner manages both absolute and relative beams. This search manager produces a result that contains pointers to active paths at the last frame processed.

• Word Pruning Breadth Search Manager: performs a frame synchronous Viterbi search with a pluggable Pruner that is called on each frame. Instead of managing a single Active List, it manages a set of Active Lists, one for each of the state types defined by the Linguist. Pruning is performed in the decomposition and sequence order of the state types as defined by the Linguist.

• Bushderby Search Manager: performs a generalized frame-synchronous breadth-first search using the Bushderby algorithm, performing classifications based on free energy as opposed to likelihoods.

• Parallel Search Manager: performs a frame synchronous Viterbi search on multiple feature streams using a factored language HMM approach as opposed to the coupled HMM approach used by AVCSR [**3**]. An advantage of the factored search is that it can be much faster and far more compact than a full search over a compound HMM.

CHAPTER FIVE

PROTOTYPE OF AMHARIC DICTATION SYSTEM

In this chapter we will describe the process that we followed to build a simple prototype Dictation system for Amharic language and an Add-On which is used as an extension for apache open office word processor that is used for experimenting dictation. According to our proposal we build the entire prototype with java programming language since we have used sphinx-4 as decoder. Sphinx-4 is entirely built with java programming language. The full sphinx-4 decoder framework is described in the section 4.5.1. The Amharic dictation system prototype mainly used the sphinx-4 decoder as the core recognizer library for getting latest results of test utterances. We have also built models for speech recognition with sphinx-4 decoder.

5.1. General Architecture of Amharic Dictation System Prototype

The entire prototype is built with java programming language. We built an Amharic word processor with java that we used it to demonstrate the recognizer results. Figure 5.1. Shows the general architecture of the prototype.

The prototpe that we develop is a push to talk (PPT) application wihch means in every sentenence the user must push the button listen (አዳምጥ) when the user wants to write sentences using voice. Since the basic idea of this research work is to show the possiblities of developing dictaton application for Amharic language for the next time it will be customized and we will try to make the application fully controlled by speech rather than using a push to talk button. We proposed to make a voice controlled dictator by incorporating different tasks on the existing system we proposed some of the solutions in the future work section.

This prototype is mainly developed for experimentation purpose only due to the functionality that it performs. Since it is just a prototype it does not include all the functionalities of general word processor application.

For the future this prototype will be improved to function like the general word processor and will be available for research works which we believe that it will help future researchers that have an interest on working the applications of Amharic speech recognition research.

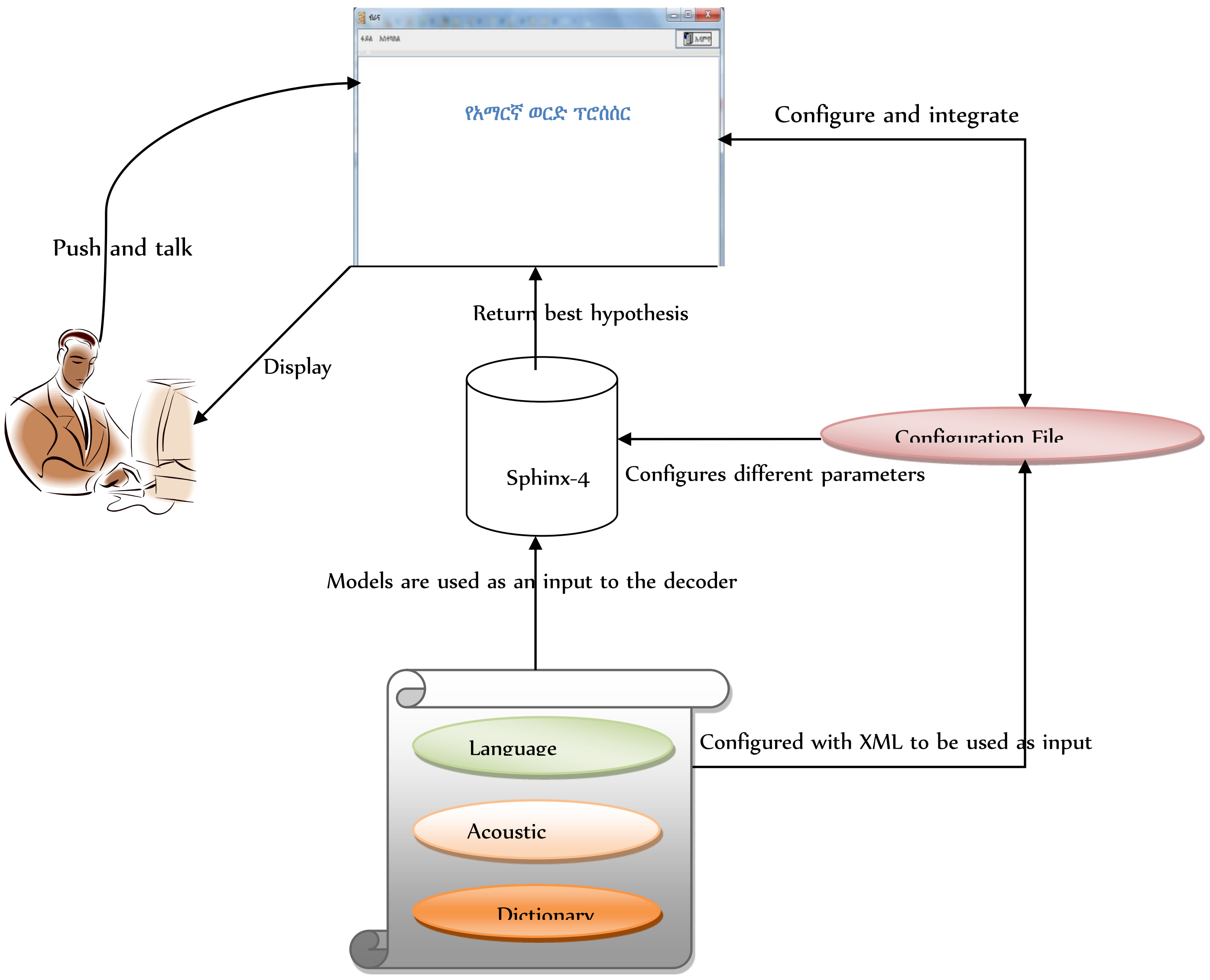

Figure 5. 1: General Architecture

The next figure 5.2. Shows the general overview of the full system and how it works.

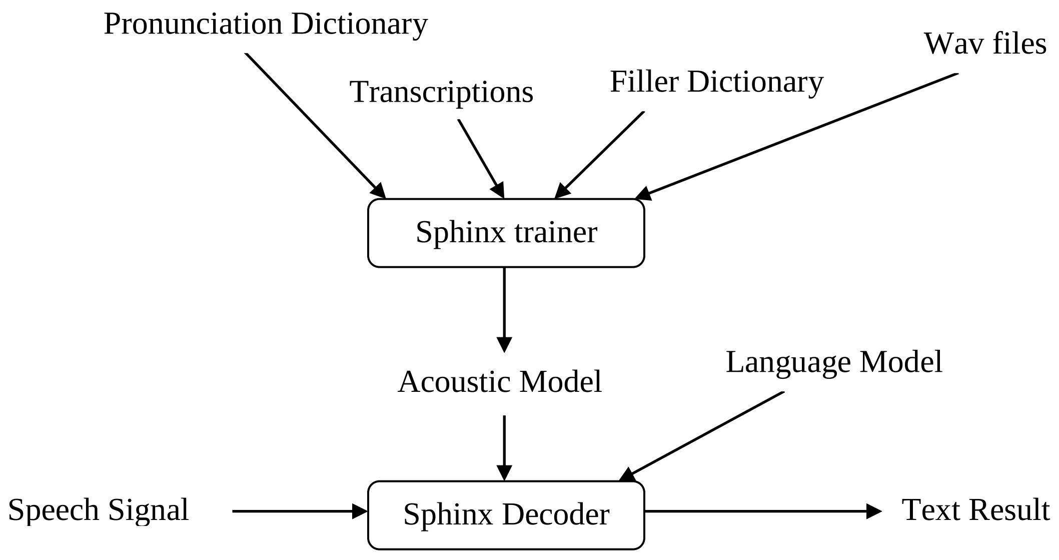

Figure 5. 2: Overview of the full system

As we can see from the general architecture the entire prototype is integrated form different components. The core decoder is sphinx-4; detailed description of sphinx-4 framework is already discussed in section 4.5.1. In the next sections we will try to show other components that are used as input for the prototype we developed.

5.2. Amharic Word Processor

Our prototype dictation system for Amharic language is named Brana(ብራና) or parchment. The term parchment (ብራና) refers to any animal skin, particularly goat, sheep, or cow, that has been scraped or dried under tension. Vellum (from the Old French *velin* or*vellin*, and ultimately from the Latin *vitulus*, meaning a calf) in theory refers exclusively to calfskin, and is used to denote a finer quality of material. It is said to be only in relatively modern times that confusion between the terms has arisen: traditionally the distinction was more strictly observed, for example by lexicographer Samuel Johnson in 1755 and by master calligrapher Edward Johnston in 1906. However, when old books and documents are encountered it may be difficult, without scientific analysis, to determine the precise animal origin of a skin in terms of its species, let alone the age of the animal and for this reason many conservators, librarians and archivists prefer to use either the broader term "parchment", or the neutral term "animal membrane" [**20**].

It is generally agreed that the origin of parchment (ብራና) making found in Ethiopia today likely lies with Christian monks who braved crossing the red sea around the 4th century and brought the bible with them.

Methods exist in Ethiopia that has not been used in European parchment (ብራና) production for over a thousand years, Richard Pankhrust, a renowned authority on Ethiopian manuscript illustration, tells IPS[2]. “This makes Ethiopia unique in keeping the tradition so far into the modern age” [**20**].

Figure 5. 3: Central European (Northern) type of finished parchment made of goatskin stretched on a wooden frame [adapted from [**20**]]

The main goal of this prototype is construction of a fully-functional RTF[3] word processor application. Though Swing's HTML and RTF capabilities are very powerful, they are not yet complete. RTF support is further along than HTML, and this is why we chose to design our word processor for use with RTF documents. The basic component of this word processor uses the capabilities of JTextPane and RTFEditorKit to display and edit RTF documents. It demonstrates very basic word processor functionality, opening and saving an RTF file, and serves as the foundation for our word processor application to be expanded.

The BRANA constructor first instantiates our JTextPane and RTFEditorKit, and assigns the editor kit to the text pane (it is important that this is done before any documents are created). Next

[2] IPS: Inter Press Service
[3] RTF:- Rich Text Format

our Style-Context is instantiated and we build our Default-Styled-Document with it. The Default-Styled-Document is then set as our text pane's current document.

The createMenuBar() method creates a menu bar with a single menu titled "**ፋይል**." Menu items **"አዲስ," "ክፈት," "አስቀምጥ,"** and **"ዘግተህ ውጣ"** are added to the menu. The first three items are duplicated in the toolbar. Full classes and instants of the code that we used to implement our prototype are listed in Appendix D.

If one user wants to use the prototype there are some procedures that the user must follow since this prototype is designed to function in a push-to-talk mechanism the user must push a button on the prototype right top corner which says “አዳምጥ (Listen)” after plugging the microphone.

5.3. Dictation with Apache OpenOffice Writer

Voice capture open source: Sphinx4 project was selected via application programming interface (API) for capturing and processing sound, as a plug-in. By integrating OpenOffice Writer, Sphinx4 with some add-ons, the open source word processor program allows a user to use their sound for typing different Amharic words or sentences on OpenOffice Writer.

The OpenOffice Writer plug-in allows developers to develop plug-in that extend the ability of OpenOffice Writer with Java language. That means the OpenOffice Writer and its plug-in feature are able to operate on multiple OS[4]. Alike the OpenOffice Writer, Sphinx4 which has been chosen as speech recognition system was also been developing entirely with Java.

5.3.1. OpenOffice Writer Plug-In

The plug-in feature of OpenOffice Writer is called Extensions project. Creating Extensions for OpenOffice Writer can be done in various programming language (e.g. Python, Basic, Java, C++). We’ve chosen Java as implementation language. The Sound Commanded OpenOffice Writer Extensions creates a new menu bar item on top of OpenOffice Writer frame Figure 5.4 This menu bar item will be used as gateway to reach the Extension.

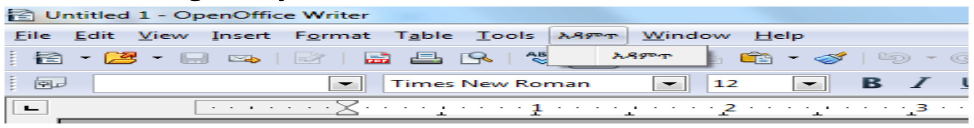
Untitled 1 - OpenOffice Writer
File Edit View Insert Format Table Tools አዳምጥ Window Help
አዳምጥ
Times New Roman
12

Figure 5. 4: New Menu Item in OpenOffice Writer (አዳምጥ)

[4] OS: Operating System

In Figure 5.4 the new menu bar item has been added into OpenOffice Writer. This Menu Bar Item is created from Add-on feature. The Add-on is a part of Extensions feature which allows developer creates visible User Interface (UI) on main OpenOffice Writer window.

5.4.The Recognizer

We built our own speech recognizer by using sphinx-4 as a core decoder. During building of this speech recognizer we prepared domain specific data. The process and tools that we used for data preparation is described in the next sections.

Data preparation

Our data preparation is classified into two data types that used for training the recognizer and data for testing the performance of the recognizer. Since any speech recognizer must have an Acoustic Model, Language Model and Pronunciation Dictionary each model needs its own data.

5.4.1. Acoustic Model

Our acoustic model is developed with Sphinx Trainer. In order to train the acoustic model we used a spontaneous speech of twenty four speakers of twenty male and four female. The number of female speakers is less because of the speech data we have got does not have lots of female speakers. During the preparation of the acoustic model we have used different tools beyond Sphinx Trainer and also we have to delve how to prepare best acoustic model with this tool. Before we have to train using our tools we followed lots of procedures. The speech data was extremely noisy and was not segmented, because of this we have to segment the speech data into sentence level speech to do that we used praat speech processing tool for segmenting the speech. Detailed description is shown in the next sections.

Signal Segmentation and Annotation

The recorded speech is several hours long and is recorded in court room which have lots of background noises and also the recording device was specially used in the court room is used to record speech in a highly compressed form which is DSS[5] file to minimize the size of the file as much as possible which is obvious that quality of the audio is very low the recording device by itself has its own noises which makes our work so difficult to model best acoustic model. Before segmentation of the speech we were supposed to change the DSS file into sphinx supported wav

[5] DSS: Digital Speech Standard

file format in order to do that we used another software tool that converts DSS file into MS-wav[6] file format.

Waveforms need to be segmented to shorter parts and annotated. The recordings have been segmented and annotated fully manually but we suppose to use some automated or semi-automated procedure for the next collection for saving of hard manual work.

Freely distributed speech segmentation, labeling, and transcription software praat [**21**] is used for signal segmentation. It gives full support for the first steps in speech corpus creation. The required steps of signal processing and annotation are described in the following sections.

Segmentation

Each utterance is divided into sentences, which are the most suitable form for the next processing and for training of ASR system. As the transcription is not known, the start- and end-points of the sentences are found manually.

As shown in [**22**] the start-point and end-point of the spontaneous sentences can be very different from fluent read speech. False starts, repairs, and repetitions can appear and the sentence can be also very long. Moreover, the speech can be silent at the end of the sentence. Important parts of speech can be “hidden” in the environmental noise, but it is important to keep this information in the correct segment. It could be useful in further noise compensating algorithms since the speech data we used is extremely noisy.

Orthographic Transcription

The speech content that we segmented is transcribed in the form of orthographic annotation. We used the orthography of Amharic language listed in Appendix for orthographic annotation since Sphinx supports Unicode language unlike HTK uses only ASCII we did not transliterated the annotated text into its equivalent ASCII transcription. The speech is written in the form it is exactly spoken, even in the case of colloquial language or mathematical expressions, which are typically present in technical speech. We did not use any special symbols or representations for spelling, mispronunciations and foreign words. Punctuation is not noted as it is not important for training a phoneme model-based speech recognizer [**22**]. The annotation is not checked by

[6] MS-wav: Microsoft wave

another annotator more than once to guarantee the correct transcription because of time and human resource shortage it is one person work.

Non-speech Event Annotation

Spontaneous speech differs strongly from read speech mainly by the fact that speakers often need to think about succeeding words [**22**]. We have modeled seven non-speech events. The non-speech events are listed in the next table.

Representation Mark	Depiction	Representation Mark	Depiction
Speaker-generated events		**Other Speaker Distortion**	
++FP++	Filled pause	++OTH++	Other Speaker
++BR++	Breath	**Background Noise**	
++LP++	Lip smack	++NOISE++	Stationary Noise
++HES++	Hesitations		
++THC++	Through clear		

Table 5. 1: Non-speech events marks and their description

Speaker-generated events: - are those events generated directly by the speaker while at the time of speaking we used five types of those events in our orthographic transcription. Here we used general representations for each non-speech events. ++FP++ represents all any of filled pauses like 'ah''um','uh' and the like. ++BR++ represents like 'inhale', 'exhale' etc.

For example: - <s> ++BR++ እየመረጠ ሰጠን እሱ </s> as we see in the example we used non-speech events in the orthographic transcription directly as it is spoke by the speaker.

Other Speaker Distortion: - we used the representation mark ++OTH++ for annotating when another speaker is talking or making sound while the first speaker is talking.

For example:- <s> እኔ ሽሚዙን ይዤ ++OTH++ መሀል ላይ ቆምኩኝ ++LP++ </s> we can see form the example that another speaker is interfere while the first speaker was talking.

Background Noise: - we used this mark for representing a background noise since the speech is recorded in a court room there are different background noises like noises that come from outside of the court room, chair and table sounds, murmuring of attending people in the court room.

```
                                Training char for the sphinx trainer
                               ==========================
                                       Type of model
                                             |
                         ------------------------------------------------------
                        |                                                    |
                  CONTINUOUS                                        SEMI-CONTINUOUS
                        |                                                    |
                        |                                           Vector-quantization
                        |                                                    |
                        ------------------------------------------------------
                                             |….make ci mdef[7]
                                             |….flat_initialize CI[8] models
                                       Training CI models
                                             |….make cd untied mdef
                                             |….initialize
                                             |
                                       Training CD[9] untied models
                                             |
                                             |
                                       Decision tree building
                                             |….prune trees
                                             |….tie states
                                             |….make cd tied mdef
                                       Training CD tied models
                                             |
Recursive                ------------------------------------------------------------
Gaussian splitting…….    |                                                          |
                     Continuous models                                  semi-continuous models
                         |                                                          |
                   ---------------------                                  deleted interpolation
                   |                   |                                            |
     Decode with              ADAPT                                                 |….ADAPT
     sphinx3/4                         |                                            |          |
      decoder  <-----------------                                        ------------------------
                                       make cd tied mdef….|----------------------- |
                                       with decode dict and  |                convert to sphinx2
                                       pruned trees          |                       |
                                                    decode with S3/4        decode with s2
```

Figure 5. 5: Acoustic Model Procedure with Sphinx Train

[7] mdef :- Model Definition File
[8] CI:- Context Independent
[9] CD:- Context Dependent

The above chart shows how sphinx train works to model the entire acoustic model. We used for our acoustic model the continuous model procedure because it is recommended for speech data recorded with 16 KHz sampling rate and also it is recommended for sphinx-3 decoder and above. Semi-continuous models are recommended for speech data of telephone conversations and speech that are recorded with 8 KHz sampling rate.

Feature extraction

The feature extraction stage seeks to provide a compact representation of the speech waveform. This form should minimize the loss of information that discriminates between words, and provide a good match with the distributional assumptions made by the acoustic models. For example, if diagonal covariance Gaussian distributions are used for the state-output distributions then the features should be designed to be Gaussian and uncorrelated.

Feature vectors are typically computed every 13 ms using an overlapping analysis window of around 25 ms. One of the simplest and most widely used encoding schemes is based on mel-frequency cepstral coefficients (MFCCs). These are generated by applying a truncated discrete cosine transformation (DCT) to a log spectral estimate computed by smoothing an FFT with around 20 frequency bins distributed non-linearly across the speech spectrum. The nonlinear frequency scale used is called a mel scale and it approximates the response of the human ear. The DCT is applied in order to smooth the spectral estimate and approximately de-correlate the feature elements. After the cosine transform the first element represents the average of the log-energy of the frequency bins. This is sometimes replaced by the log-energy of the frame, or removed completely.

Acoustic Model Training files

During training of our acoustic model we used spontaneous speech corpus of 90 min which is a total of 1550 (one thousand five hundred and fifty) uttered sentences, our pronunciation dictionary is 158 KB canonical dictionary which has a total of 4203 (four thousand two hundred and three) unique words, the total phone coverage we used is 46 (forty six) phones which is 38 (thirty eight) basic Amharic language phones and 8 (eight) non speech event phones including silence which is listed on Appendix A. After extracting the features of recorded speech we organized all the prepared data into two folders the etc and wav folder in which each folders holds different data. During feature extraction stage we got a set of feature files computed from the audio training data we collect for every recording we have in the training corpus. Each recording is transformed into a sequence of feature vectors using a front-end executable provided with SPHINX training package called sphinx_fe and we used parameters for sphinx_fe listed in Appendix C. During feature extraction the feature files are stored in a different folder called feat. A **control file** containing the list of feature-set filenames with full paths to them also provided for the training of acoustic model. An example of the entries in this file:

dictates/tr00-002

dictates/tr00-003

Note that the extensions are not given. They will be provided separately to the trainer in the configuration file. This file is located under the folder etc. A **transcript file** in which the transcripts corresponding to the feature files are listed in exactly the same order as the feature filenames in the control file. An example is listed below.

<s> የወንጀል መዝገብ ቁጥር ሀያ ባዶ አንድ ስልሳ ሶስት </s> (tr00-002)

<s> ፌደራል መጀመሪያ ደረጃ ፍርድ ቤት </s> (tr00-003)

This transcript file is also located in the same folder like the control file in which the trainer configuration file easily locate each files during training. A **main dictionary** which has all acoustic events and words in the transcripts mapped onto the acoustic units. Details of the main dictionary is stated on section 5.4.2.

A **filler dictionary** which usually listed the non-speech events as “words” and maps them to user_defined phones. A **phone list,** which is a list of all acoustic units the we use for training. We used total of 46 phones including phones that are used for mapping non-speech events generally speaking we used 38 Amharic phones and 8 non-speech phones. This are all the files that are used for training and they are stored in the same directory etc. wav directory is used to store all the audio recordings that are used for training. During acoustic model training of our data we followed the procedure that is shown on the sphinx trainer chart.

5.4.2. The Dictionary (lexical Model)

The lexical model is another basic component of speech recognition. To develop our lexical model we used a total of 4203 words that are uttered in the training data. In order to generate the lexical model since there is no any common program that is used for making dictionaries we developed our own program using python programming that is used for generating pronunciation automatically. In order to generate this lexical model we used 38 phones of Amharic. We developed a canonical dictionary. An example is shown below.

ህግ	h ee g ee
ሆነ	h o n aa
ሆነን	h o n aa n ee

5.4.3. Language Model

Language models (LM) are fundamental to many natural language applications such as automatic speech recognition (ASR) and statistical machine translation (SMT).

A statistical language model (SLM) is a probability distribution p(W) over word sequences W that models how often each sequence W occurs as a sentence. Our language models are developed into two modeling tool kits CMU-LM tool kit and SRILM tool kit. We used a text of 30MB normalized text which we get from Dr. Solomon Teferra. In order to make the language model we have followed different smoothing techniques for modeling the best language model since language model has a huge effect in recognition results. Some of the smoothing techniques we used are Absolute Discounting, Modified Kneser-Ney Discounting, Good-Turing discounting, Witten-Bell Discounting.

We developed 4 (four) different language models using the above listed smoothing techniques. The perplexity of each language model is evaluated with 68 sentences with 412 words including start silence (<s>) and end silence (</s>). Perplexity result of the models is described in the table 5.2 below. OOV of each model is 0 (zero) since we used the word lists that we used for training and testing to make language models of 30MB text.

Language Model Code	Smoothing Technique	Perplexity
LM-ABS	Absolute Discounting	2309.08
LM-GT	Good-Turing discounting	7906.64
LM-MKN	Modified Kneser-Ney Discounting	2221.74
LM-WB	Witten-Bell Discounting	4574.5

Table 5. 2: Language model perplexity of different smoothing techniques

CHAPTER SIX

EXPERIMENT OF BRANA (ብራና)

This chapter focuses on the evaluation of our system, recogniton performance, reliablility and trustworthiness and how the dictation system is working well.

Our dictation system has been given a name **ብራና (prachment),** a hisorical writting equipement that is used in Ethiopia. This name is selected in the sense that to memorize our ansistors were very creative and smart to use animals skin to write down different historical, cultural, religious, phylosophic and different books.

The first task we have used for our system evaluation is preparing test data for evaluating the recognizer perfomrance and we have prepared 68 uttered sentences of different speakers which is one female and five male speakers.

6.1. Testing Environment

For this research work, we have used Sphinx as a main component for recognition. The dictation prototypes we have developed are implemented using the Java programming language. The standard Java libraries such as RTF text editor, etc. have been used for text processing. The eclipse Java editor has been used to develop oursystem and net-beans to develop open-office writer extension. The main Java class files developed are shown in Appendix .

We have used Windows 7 Ultimate Edition as an operating system. The hardware component comprises of Intel(R) Core(TM)2 duo CPU of 2.00 GHz, 3 GB memory, and 80 GB hard disk.

6.2. Recognition Evaluation

In this section, we will evaluate the performance of the recognizer that is developed with sphinx-4 decoder and sphinx-3 decoder. The evaluation is classified in to different categories based on the acoustic model developed and based on the language models that are used. The next table shows recognition result of those 68 sentences with CI acoustic model. The acoustic model is developed with no skip transition topology and 8 Gaussian mixtures. The recognition performance on the next table shows stream data recognition.

Speaker	Acoustic Model	Language Model	Recognition Accuracy	Word Error Rate(WER)	Sentence Accuracy
Spk1*	AM-CI[10]	LM-ABS[11]	25%	75%	12.5%
Spk2	AM-CI	LM-ABS	22%	79.6%	0%
Spk3	AM-CI	LM-ABS	9.091%	100%	0%
Spk4	AM-CI	LM-ABS	21.591%	81.818%	0%
Spk5	AM-CI	LM-ABS	26.797%	74.510%	6.667%
Spk6	AM-CI	LM-ABS	35.211%	69.014%	9.091%
Over all result			**25%**	**77.778%**	**4.412%**

Table 6. 1: Recognition Result of Sphinx-4 decoder with CI acoustic model and language Model with Absolute Discounting smoothing

The above table shows recognition result of sphinx-4 decoder which is tested with 68 sentences and speaker which is marked with * sign is participated in the training. Acoustic model is developed with the basic phones only and it is Context Independent acoustic model. The average recognition accuracy is 25%. The CD acoustic model with one (1) Gaussian mixture shows 19.907% difference than that of CI acoustic model.

Speaker	Acoustic Model	Language Model	Recognition Accuracy	Word Error Rate(WER)	Sentence Accuracy
Spk1*	AM-CD_1[12]	LM-ABS	60.714%	39.286%	37.5%
Spk2	AM-CD_1	LM-ABS	42.373%	62.712%	0%
Spk3	AM-CD_1	LM-ABS	33.333%	75.758%	0%
Spk4	AM-CD_1	LM-ABS	34.091%	67.045%	0%
Spk5	AM-CD_1	LM-ABS	43.791%	63.399%	13.333%
Spk6	AM-CD_1	LM-ABS	61.972%	43.662%	36.364%
Over all result			**44.907%**	**60.185%**	**13.235%**

Table 6. 2: Recognition Result of Sphinx-4 decoder with CD acoustic model with one (1) Gaussian mixture and language Model with Absolute Discounting smoothing

[10] AM-CI: Acoustic Model Context Independent Model
[11] LM-ABS: Language Model with Absolute Discounting Smoothing
[12] AM-CD_1: Acoustic Model Context Dependent Model with one (1) Gaussian Mixture

Table 6.2 shows the recognition result of sphinx-4 decoder with an Acoustic model of one Gaussian mixture and the overall recognition accuracy is 44.907%.

Speaker	Acoustic Model	Language Model	Recognition Accuracy	Word Error Rate(WER)	Sentence Accuracy
Spk1*	AM-CD_2[13]	LM-ABS	50%	50%	25%
Spk2	AM-CD_2	LM-ABS	45.763%	57.627%	0%
Spk3	AM-CD_2	LM-ABS	36.364%	72.727%	12.5%
Spk4	AM-CD_2	LM-ABS	35.227%	65.909%	0%
Spk5	AM-CD_2	LM-ABS	49.673%	57.516%	13.333%
Spk6	AM-CD_2	LM-ABS	64.789%	40.845%	54.545%
Over all result			**47.685%**	**57.176%**	**16.176%**

Table 6. 3: Recognition Result of Sphinx-4 decoder with CD acoustic model with two (2) Gaussian mixture and language Model with Absolute Discounting smoothing

Table 6.3 shows the recognition performance of sphinx-4 decoder with CD acoustic model of two Gaussian mixtures and it results an overall recognition accuracy of 47.685% which increases by 2.778% than that of one (1) Gaussian mixture acoustic model.

Speaker	Acoustic Model	Language Model	Recognition Accuracy	Word Error Rate(WER)	Sentence Accuracy
Spk1*	AM-CD_4[14]	LM-ABS	57.143%	42.857%	25%
Spk2	AM-CD_4	LM-ABS	50.847%	50.847%	0%
Spk3	AM-CD_4	LM-ABS	30.303%	75.758%	12.5%
Spk4	AM-CD_4	LM-ABS	34.091%	68.182%	0%
Spk5	AM-CD_4	LM-ABS	49.673%	59.477%	13.333%
Spk6	AM-CD_4	LM-ABS	69.014%	36.620%	54.545%
Over all result			**48.843%**	**56.481%**	**16.176%**

[13] AM-CD_2: Acoustic Model Context Dependent Model with two (2) Gaussian Mixture
[14] AM-CD_4: Acoustic Model Context Dependent Model with four (4) Gaussian Mixture

Table 6. 4: Recognition Result of Sphinx-4 decoder with CD acoustic model with four (4) Gaussian mixture and language Model with Absolute Discounting smoothing

Table 6.4 shows the recognition result of sphinx-4 decoder with a CD acoustic model of four (4) Gaussian mixtures and it results an overall recognition accuracy of 48.843%.

Speaker	**Acoustic Model**	**Language Model**	**Recognition Accuracy**	**Word Error Rate(WER)**	**Sentence Accuracy**
Spk1*	AM-CD_8[15]	LM-ABS	64.286%	35.714%	37.5%
Spk2	AM-CD_8	LM-ABS	50.847%	52.542%	0%
Spk3	AM-CD_8	LM-ABS	30.303%	78.788%	12.5%
Spk4	AM-CD_8	LM-ABS	40.909%	61.364%	0%
Spk5	AM-CD_8	LM-ABS	50.327%	54.902%	13.333%
Spk6	AM-CD_8	LM-ABS	66.197%	38.028%	54.545%
Over all result			**50.463%**	**53.704%**	**17.647%**

Table 6. 5: Recognition Result of Sphinx-4 decoder with CD acoustic model with eight (8) Gaussian mixture and language Model with Absolute Discounting smoothing

Table 6.5 shows recognition result of sphinx-4 decoder with Context Dependent Acoustic Model with eight (8) Gaussian Mixture and the average recognition accuracy of those six speakers is 50.463%. Which we compare it to the context independent acoustic model it shows slightly a big recognition accuracy difference. Form the above results we can observe that when the Gaussian mixture increases the recognition accuracy also increases. From the experiments we conduct acoustic model of CD with eight (8) Gaussian mixtures has better recognition accuracy. From that point we have prepared another language models with different smoothing algorithms and test the recognizer. Below tables shows the results obtained with different language models. We tried to show which language model performs better in speech recognition based on the results we obtain during our experimentation.

The next table shows the results obtained during recognition using sphinx-4 decoder by changing the language model which uses good-Turing smoothing algorithm.

[15] AM-CD_8: Acoustic Model Context Dependent Model with eight (8) Gaussian Mixture

Speaker	**Acoustic Model**	**Language Model**	**Recognition Accuracy**	**Word Error Rate(WER)**	**Sentence Accuracy**
Spk1*	AM-CD_8	LM-GT[16]	57.143%	42.857%	25%
Spk2	AM-CD_8	LM-GT	44.068%	61.017%	0%
Spk3	AM-CD_8	LM-GT	33.333%	75.758%	12.5%
Spk4	AM-CD_8	LM-GT	34.091%	68.182%	0%
Spk5	AM-CD_8	LM-GT	52.941%	51.634%	13.333%
Spk6	AM-CD_8	LM-GT	64.789%	39.437%	45.455%
Over all result			**48.611%**	**55.556%**	**14.706%**

Table 6. 6: Recognition Result of Sphinx-4 decoder with CD acoustic model with eight (8) Gaussian mixture and language Model with Good-Turing smoothing

Speaker	**Acoustic Model**	**Language Model**	**Recognition Accuracy**	**Word Error Rate(WER)**	**Sentence Accuracy**
Spk1*	AM-CD_8	LM-MKN[17]	53.571%	46.429%	12.5%
Spk2	AM-CD_8	LM-MKN	45.763%	61.017%	0%
Spk3	AM-CD_8	LM-MKN	30.303%	81.810%	12.5%
Spk4	AM-CD_8	LM-MKN	36.364%	65.909%	0%
Spk5	AM-CD_8	LM-MKN	49.673%	53.595%	6.667%
Spk6	AM-CD_8	LM-MKN	61.972%	43.662%	45.455%
Over all result			**47.222%**	**57.176%**	**11.765%**

Table 6. 7: Recognition Result of Sphinx-4 decoder with CD acoustic model with eight (8) Gaussian mixture and language Model with Modified Kneser-Ney smoothing

6.3. Dictation systems

6.3.1. BRANA (ብራና)

As the final step of our work, we developed a graphical user interface (GUI) in order to complete the Amharic Dictation System, named BRANA (ብራና). This tool was created using the java programming language swings and different java classes that used to create a user interface. A snapshot of the tool can be seen in the image below.

[16] LM-GT: Language Model with Good-Turing smoothing
[17] LM-MKN: Language Model with Kneser-Ney smoothing

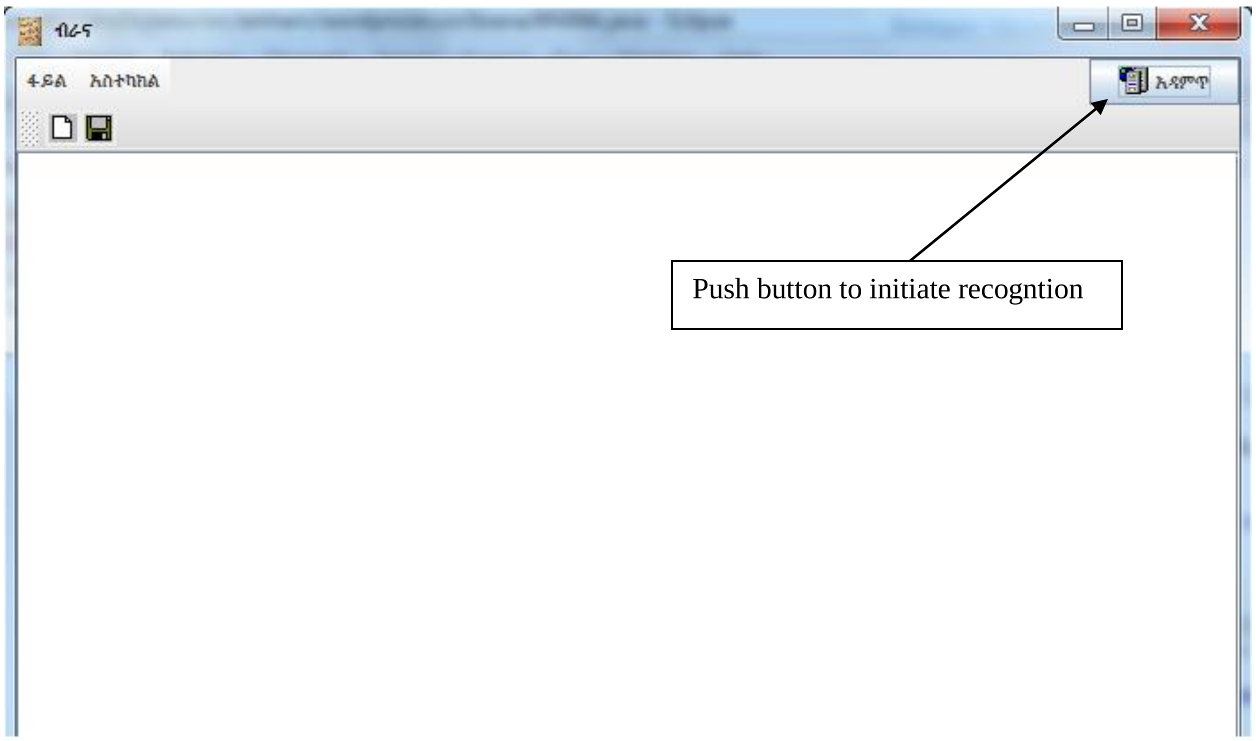

Figure 6. 1: GUI snapshot of BRANA (ብራና)

The tool can be used as a simple editor.Beside the common toolbars(e.g. ፋይል, አስተካክል , etc.) the tool offers a special recognition button. When the "አዳምጥ" button is pressed, the tool starts to listen the dictated sentence uttered by the user and it connects to the core decoder sphinx-4 and writes the recognized hypothesis on the simle RTF editor. The user can initiate a new recognition task by just pressing the "አዳምጥ" button. The result string is created and presented to the user every time when the recogition is initiated and when a user uttered desired sentence or words.

Our prototype system is not capable of executing predefined set of voice commands that are used for dictating like "አዲስ መስመር", "አወፍር (bold)" ,"አጋድም(italic)", "ከታች አስምር(underline)" etc and punctuation marks like "አራት ነጥብ (።)", "ጥያቄ ምልክት(?)" etc.

Also our system is only desigend and implemented to display the uttered senntence or word and it has no any correcton methods rather the used is supposed to correct the errors manually but we propose error correction methods on our future work section.

BRANA is tested in a home environement with three(3) subjects in which each subjects are not participated in any part of the training uttereances which will challange the performance of the system and also we used a high quality sony stereo headphone (MDR-664MV) for testing our prototype system which we belive that the training utterances we used for building the recognizer are recorded in a court environement with a device that highly compress the speech in a very highly compressed manner into a DSS file format this may create also a highly performance degradation to our system because the recording device and testing device are completely different. We used 68 sentences and 298 word for testing the sytem.

Our system is capable of communicating perfectly with the core recognizer we use (sphinx-4) and the processing time to produce an output is fast enough and promicing. From the exiperment we conduct, we tested our prototype with spontaneous speech recognizer that we developed in the entire study. In addition to our spontaneous speech recognizer we developed another speech recognizer that is modeled with read speech corpus which we get the training and testing corpus from Dr. Solomon Teferra. The recognition accuracy of our read speech recognizer we developed is tested with 359 sentences with 4097 words which we get 75% accuracy. We tested our prototype dictation system with both of the recognizers (spontaneous and read) speech recognizer and we observe that the system best perform on the read speech recognizer since it has a high accuracy than that of the spontaneous speech recognizer. From the results we get we belive that if the performace of the spontaneous speech recognizer is improved our system will also perform better.

6.3.2. Dictation with open office writer

As we tried to discuss on section 5.3.1 we developed an add-on that is integrated to open office writter that is used to initiate recogniton and it will communicate with our core recognizer sphinx-4. From the exiperment we conducted using open office writter we have got poor performance when we compare it to the prototype that we discussed on section 6.3.1. we believe this happens because of the unstablity of the code written for implementing the add-on feature which can be improved through time but we proofed that it is possible to make a dictator with open office writer. From our experiments we selected open office writer from the other editors because of its feature that allow users to modify and customize it for different purposes like including and creating an add-on extension that makes open office writer flexible. Also open office writer source code is freely available and can be modified on the interest of the user.

6.4. Discussion

During our experimentation, we were confronted with some problems. We put them as a challenge for making dictation application for Amharic language. Some of them are listed below:

Recognition Performance

Recognition errors are frustrating and cause the user to form incorrect conceptual model of the application's behavior. These recognition errors are the cause to degrade the performance of the recognizer possible causes of recognition errors. These include speaking before the system is ready to listen, background noise, an accent, a cold, an exaggerated tone, and speaking words that are not found in the recognizer's vocabulary (dictionary). Since the training speech data we used have lots of noises and is not transcribed in a controlled manner and is not phonetically rich it became a cause for the degradation of the recognition performance.

Corpus size

Since the basic input for developing good performing speech recognizer the corpus for training must be huge in size. We used a spontaneous speech corpus of 90 min which is very small and not enough to make a best performing spontaneous speech recognizer which directly affects the performance of the dictation system.

Materials used

During developing speech applications it is advisable to use quality equipment's that will make a best application. The materials that we use are not good enough to develop speech application. As we tried to show on section 6.1 our developing environment is very low for developing speech recognizer application, recording devices, headphones etc.

Language nature

The nature of Amharic language by itself is a challenge in order to make an excellent speech recognition application it is better to understand the nature of the language by discussing with the linguist professionals.

Lastly, during the experiment, we have discovered the following additional observations.

- Recognition latency: Recognition latency represents the time lag between the input utterance and output of the recognizer.
- Whenever the Gaussian mixture is increasing the recognition performance increases but we did not tested with the maximum Gaussian mixture that best performs for our recognizer due to time constraint. We experimented with 8 Gaussian mixture.
- There is stability problem on the prototype: our prototype system performs inconsistently through different times we believe that this happened due to the environment constraints during testing and the code that developed for implementing the prototype is not stable (not matured).
- The prototype best performs for read speech inputs by using a continuous speech recognizer due to the recognition performance of the read speech recognizer.
- We observed that open office writer is the best editor for dictation application since it is easily customizable.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

This thesis investigates the possibility of developing an Amharic speech recogniton applicaion for dictation in a specific domain. The use of speech is expected to make the human-computer interaction more natural and efficient than it has been so far.

In this chapter we summarize our conclusions and findings based on our exipermental finding. we review the major contributions of this work and present several suggestions for future work.

7.1. Conclusions

In this thesis we have developed an Amharic speech recogniton application for dictation that has different components. Basically the components are classiffied into two main modules the speech recognizer engine module and the dictation application module which is the front end application that has a direct relation with the end user. The speech recognizer module recognizes the uttered sentence or word which is forwarded by the end user by processing internally with its sub modules that the recogizer has. Our speech recognizer module has also its own sub modules the Acoustic model module, the languge model module and lexical (pronunciation dictionary) model module. The second basic module of the system is dictation application module. It is entirely implemented with java programming language and it is the front end that uses as a user interface for the end user to interact with the speech recognizer module.

This research work attempted to identify the basic language specific issues in speech recogniton application. We have collected spontaneous speech corpus that is used for training of the recognizer and put lots of effort on segmenting and labeling the collected spontaneous speech corpus which is appropriate to use the selected speech recognizer engine in our case CMUsphinx. We have also identified proper non-speech events that happened in the collected speech corpus since the collected speech corpus is spontaneous. During the training phase we followed the standard procedure that is used by SphinxTrain trainer described on page 59 inorder to form our acoustic models which is one component of the speech rcognizer. We implemented our own python program that allows us to create a canonical lexial (pronunciation dictionary) automatically. We used SRILM and CMU-LM language modeling toolkit that allows us to create an ngram language model and since in todays undergoing Automatic Speech recogniton research a tri-gram language model is recommended we developed a tri-gram language model with 30M normalized text corpus with different smoothing techniques.

Finally we implemented two different dictation application systems. The first dictation application is our prototype BRANA(ብራና) in which the entire application is developed with java programming language and it is a PPT(push to talk) prototype which shows a good performance in writing down the recognized results as per the speech recognizer performance. The second dictation application application is an extention application which is part of an Add-On for open office writter but this dictation application does not show such a good performance due to luck of stability of the code.

The evaluation of our system, being the first Speech recogniton application for dictation, shows promising performance. Our speech recognizer for spontaneous speech shows a good performance with small training corpus (90 min). We have got 50% word accuracy which will increase as the training corpus increases. Also we have tested our application with continuous (read) speech recognizer. According to our exiperimental results the performance of continuous speech recognizer is better than that of the spontaneous speech recognizer that shows our system to perform better on the recognizer that have a better recognition accuracy.

7.2. Contribution of the work

The main contributions of this thesis work are summarized as follows:

- We prepared spontaneous speech database that may be useful for researchers that have an interest on doing research on spontaneous speech recognition
- We prepared a manually transcribed text data for the spontaneous speech database
- We implemented a python program that is used to generate a canonical pronunciation dictionary for Amharic words automatically
- We developed a spontaneous speech recognizer that may be useful for future researchers
- We implemented a functional dictation application prototype
- We implemented our recognizer with new recognition tool (sphinx) that is freely available for research and developing an application. To the best of our knowledge it is not used for developing Amharic speech recognizer in previous works.

7.3. Future work

Automatic speech recognition application is a very complex task, which consumes more time, and needs a number of different NLP tools and machine learning algorithms. Hence, there are a number of rooms for improvement and modification for Amharic speech recognition applications. Below are some of the recommendations we propose for future work.

- Automatic Error Correction: The goal of error correction is to produce error-free SR output by repairing recognition errors. Error correction involves three steps: detecting an error, navigating to the error, and correcting the error. Manual error correction is neither easy nor efficient.
- Improve performance of spontaneous speech recognizer: this will be improved by increasing the training speech database to increase the performance of our system since the recognition performance has a direct impact to improve our system.
- Incorporate command and control: by including command and control functionalities to our system it will help to automatically correct errors, used to navigate easily to another line or paragraph or words etc.
- Noise compensation: this will help to improve the performance of the recognizer by compensating different noises that are recorded on the training data.
- Voice activated dictator: since our prototype is working in the form of PPT (Push To Talk) functionality it is better to make it voice activated system that will save users time correcting errors manually, pushing every time to utter which is tedious task and increase processing time for the system.
- Generic dictator: it will be better to make a generic dictator that works for anyone rather than making domain specific dictator.
- Language model: an ngram language model that best performs for making a good generic dictation application.
- Pronunciation dictionary: preparing alternative lexical (pronunciation dictionary) model that best much's for dictation application which supports gemination to increase the performance.

References

[1] Muhirwe Jackson , "Automatic Speech Recognition: Human computer interface for KinyaRwanda language," *M.Sc. Thesis*, 2005.

[2] Mats Blomberg and Kjell Elenius , "Automatic Recognition of speech ," *Department of speech, Music and Hearing (TMH)*, 2005.

[3] Mikko Kurimo et al., "Automatic Speech Recognition," , p. 114.

[4] Martha Yifiru Tachbelie, "Application of Amharic Speech Recognition system to command and control computer: an experiment with Microsoft Word," Addis Ababa University , Addis Ababa, M.Sc. Thesis 2003.

[5] Mesfin Birile Woldetsadik, "Synthetic Speech Trained- Large vocabulary Amharic Speech Recognition system," Addis Ababa University, Addis Ababa, M.Sc. Thesis 2008.

[6] Solomon Teferra , "Automatic Speech Recognition for Amharic," Hamburg Univesrity, Addis Ababa, PhD Dissertation 2005.

[7] Solomon Berhanu , "Isolated Amharic Consonant-Vowel (CV) syllable Recognition: an experiment using Hidden markov model," Addis Ababa University, Addis Ababa, M.Sc. Thesis 2001.

[8] Kinfe Tadesse , "Sub-Word Based Amharic word recognition: an experiment using Hidden markov model (HMM)," Addis Ababa University, Addis Ababa, M.Sc. Thesis 2002.

[9] Yongmei Shi , "An Investigation of linguistic information for speech recognition error detection," University of Maryland, Baltimore County, PhD Dissertation 2008.

[10] Markku Turunen , *Speech Application Design and Development*.: University of Tampere , 2004.

[11] Stephen Cook , *Speech Recognition HOWTO*., 2000.

[12] Ebru Arsoy , "Turkish dictation system for radiology and broadcast news applications," Bogazi University, M.Sc. Thesis 2002.

[13] Xu Bing Bo, Shuwu Zhang Ma, Fei Qu , and Taiyi Huang , "Speaker-independent dictation of chinese speech with 32K vocabulary," *Speech research group*, CF.

[14] Nega Alemayehu and Peter Willett , "Stemming of Amharic words for information retrieval," CF.

[15] ባየ ይማም , *የአማርኛ ሰዋሰው.* አዲስ አባባ, ኢትዮጵያ: ትመማማድ, 1987.

[16] Wikipidia. (2014, July) Wikipidia Web Site. [Online]. http://en.wikipedia.org/wiki/Amharic_language

[17] ጌታሁን አማረ , *የአማርኛ ሰዋሰው በቀላል አቀራረብ.*, 1989.

[18] Seid Muhie Yimam, "TETEYEQ (ተጠየቅ):AMHARIC QUESTION ANSWERING FOR FACTOID QUESTIONS," Addis Ababa University, Addis Ababa , M.Sc Thesis 2009.

[19] Willie Walker et al., "Sphinx-4: A flexible open source framework for speech recognition," *Sun Microsystems inc.*, 2004.

[20] Wikipidia. (2014, July) Wikipidia Web Site. [Online]. http://en.wikipedia.org/wiki/Parchment

[21] Pauline Welby and Kiwako Ito , *Praat Tutorial*. Ohio, Ohio State: Ohio State University, 2002.

[22] Josef Rajnoha and Peter Poll'ak , "Czech spontaneous speech collection and annotation: the database of technical lectures," CF.

Appendices

Appendix A: list of phones

+FP+
+BR+
+LP+
+HES+
+THC+
+OTH+
+NOISE+
SIL
a
aa
b
c
cc
d
dd
e
ee
f
g
h
hh
i
j
k
kk
l
m
n
nn
o
p
pp
q
r
s
ss
t
tt
u
ua
uaa
v
w
x
z
zz

Appendix B: Training configuration

```
# Configuration script for sphinx trainer                          -*-mode:Perl-*-

$CFG_VERBOSE = 1;          # Determines how much goes to the screen.

# These are filled in at configuration time

$CFG_DB_NAME = "Dictation";

$CFG_BASE_DIR = "D:/CS-MSC-MATERIALS-05/Thesis/Sphinx/Dictation";

$CFG_SPHINXTRAIN_DIR = "../../SphinxTrain";

# Directory containing SphinxTrain binaries

$CFG_BIN_DIR = "$CFG_BASE_DIR/bin/Release";

$CFG_GIF_DIR = "$CFG_BASE_DIR/gifs";

$CFG_SCRIPT_DIR = "$CFG_BASE_DIR/scripts_pl";

# Experiment name, will be used to name model files and log files

$CFG_EXPTNAME = "$CFG_DB_NAME";

# Audio waveform and feature file information

$CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav";

$CFG_WAVFILE_EXTENSION = 'wav';

$CFG_WAVFILE_TYPE = 'mswav'; # one of nist, mswav, raw

$CFG_FEATFILES_DIR = "$CFG_BASE_DIR/feat";

$CFG_FEATFILE_EXTENSION = 'mfc';

$CFG_VECTOR_LENGTH = 13;

$CFG_MIN_ITERATIONS = 1;  # BW Iterate at least this many times

$CFG_MAX_ITERATIONS = 10; # BW Don't iterate more than this, somethings likely wrong.

# (none/max) Type of AGC to apply to input files

$CFG_AGC = 'none';

# (current/none) Type of cepstral mean subtraction/normalization

# to apply to input files

$CFG_CMN = 'current';

# (yes/no) Normalize variance of input files to 1.0
```

```
$CFG_VARNORM = 'no';

# (yes/no) Use letter-to-sound rules to guess pronunciations of

# unknown words (English, 40-phone specific)

#$CFG_LTSOOV = 'no';

# (yes/no) Train full covariance matrices

$CFG_FULLVAR = 'no';

# (yes/no) Use diagonals only of full covariance matrices for

# Forward-Backward evaluation (recommended if CFG_FULLVAR is yes)

$CFG_DIAGFULL = 'no';

# (yes/no) Perform vocal tract length normalization in training.  This

# will result in a "normalized" model which requires VTLN to be done

# during decoding as well.

$CFG_VTLN = 'no';

# Starting warp factor for VTLN

$CFG_VTLN_START = 0.80;

# Ending warp factor for VTLN

$CFG_VTLN_END = 1.40;

# Step size of warping factors

$CFG_VTLN_STEP = 0.05;

# Directory to write queue manager logs to

$CFG_QMGR_DIR = "$CFG_BASE_DIR/qmanager";

# Directory to write training logs to

$CFG_LOG_DIR = "$CFG_BASE_DIR/logdir";

# Directory for re-estimation counts

$CFG_BWACCUM_DIR = "$CFG_BASE_DIR/bwaccumdir";

# Directory to write model parameter files to

$CFG_MODEL_DIR = "$CFG_BASE_DIR/model_parameters";
```

```
# Directory containing transcripts and control files for

# speaker-adaptive training

$CFG_LIST_DIR = "$CFG_BASE_DIR/etc";

#*******variables used in main training of models*******

$CFG_DICTIONARY     = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";

$CFG_RAWPHONEFILE   = "$CFG_LIST_DIR/$CFG_DB_NAME.phone";

$CFG_FILLERDICT     = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";

$CFG_LISTOFFILES    = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.fileids";

$CFG_TRANSCRIPTFILE = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.transcription";

$CFG_FEATPARAMS     = "$CFG_LIST_DIR/feat.params";

#*******variables used in characterizing models*******

$CFG_HMM_TYPE = '.cont.'; # Sphinx III

#$CFG_HMM_TYPE  = '.semi.'; # PocketSphinx and Sphinx II

if (($CFG_HMM_TYPE ne ".semi.") and ($CFG_HMM_TYPE ne ".cont.")) {

  die "Please choose one CFG_HMM_TYPE out of '.cont.' or '.semi.', " .

    "currently $CFG_HMM_TYPE\n";

}

# This configuration is fastest and best for most acoustic models in

# PocketSphinx and Sphinx-III.  See below for Sphinx-II.

$CFG_STATESPERHMM = 3;

$CFG_SKIPSTATE = 'yes';




if ($CFG_HMM_TYPE eq '.semi.') {

  $CFG_DIRLABEL = 'semi';

# Four stream features for PocketSphinx

  $CFG_FEATURE = "s2_4x";
  $CFG_NUM_STREAMS = 4;
  $CFG_INITIAL_NUM_DENSITIES = 256;
  $CFG_FINAL_NUM_DENSITIES = 256;
# If you wish to build models for Sphinx-II, uncomment these lines
#  $CFG_STATESPERHMM = 5;
#  $CFG_SKIPSTATE = 'yes';
  die "For semi continuous models, the initial and final models have the same density"
    if ($CFG_INITIAL_NUM_DENSITIES != $CFG_FINAL_NUM_DENSITIES);
} elsif ($CFG_HMM_TYPE eq '.cont.') {
  $CFG_DIRLABEL = 'cont';
# Single stream features - Sphinx 3
  $CFG_FEATURE = "1s_c_d_dd";
  $CFG_NUM_STREAMS = 1;
  $CFG_INITIAL_NUM_DENSITIES = 1;
  $CFG_FINAL_NUM_DENSITIES = 20;
  die "The initial has to be less than the final number of densities"
    if ($CFG_INITIAL_NUM_DENSITIES > $CFG_FINAL_NUM_DENSITIES);
}
# (yes/no) Train multiple-gaussian context-independent models (useful
# for alignment, use 'no' otherwise) in the models created
# specifically for forced alignment
$CFG_FALIGN_CI_MGAU = 'no';
# (yes/no) Train multiple-gaussian context-independent models (useful
# for alignment, use 'no' otherwise)
$CFG_CI_MGAU = 'no';
# Number of tied states (senones) to create in decision-tree clustering
$CFG_N_TIED_STATES = 1000;
```

```
# How many parts to run Forward-Backward estimatinon in

$CFG_NPART = 1;


# (yes/no) Train a single decision tree for all phones (actually one

# per state) (useful for grapheme-based models, use 'no' otherwise)

$CFG_CROSS_PHONE_TREES = 'no';


# Use force-aligned transcripts (if available) as input to training

$CFG_FORCEDALIGN = 'no';


# Use a specific set of models for force alignment.  If not defined,

# context-independent models for the current experiment will be used.

$CFG_FORCE_ALIGN_MDEF="$CFG_BASE_DIR/model_architecture/$CFG_EXPTNAME.falign_ci.
mdef";

if ($CFG_FALIGN_CI_MGAU eq  'yes') {

  $CFG_FORCE_ALIGN_MODELDIR                                                    =
"$CFG_MODEL_DIR/$CFG_EXPTNAME.falign_ci_${CFG_DIRLABEL}_$CFG_FINAL_NUM_DE
NSITIES";

}

else {


$CFG_FORCE_ALIGN_MODELDIR="$CFG_MODEL_DIR/$CFG_EXPTNAME.falign_ci_$CFG_DI
RLABEL";

}


# Use a specific dictionary and filler dictionary for force alignment.

# If these are not defined, a dictionary and filler dictionary will be

# created from $CFG_DICTIONARY and $CFG_FILLERDICT, with noise words

# removed from the filler dictionary and added to the dictionary (this

# is because the force alignment is not very good at inserting them)
```

```
#$CFG_FORCE_ALIGN_DICTIONARY="$ST::CFG_BASE_DIR/falignout$ST::CFG_EXPTNAME.fa
lign.dict";;

#$CFG_FORCE_ALIGN_FILLERDICT="$ST::CFG_BASE_DIR/falignout/$ST::CFG_EXPTNAME.fal
ign.fdict";;

# Use a particular beam width for force alignment.  The wider

# (i.e. smaller numerically) the beam, the fewer sentences will be

# rejected for bad alignment.

$CFG_FORCE_ALIGN_BEAM = 1e-60;

# Calculate an LDA/MLLT transform?

$CFG_LDA_MLLT = 'no';

# Dimensionality of LDA/MLLT output

$CFG_LDA_DIMENSION = 29;


#set convergence_ratio = 0.004

$CFG_CONVERGENCE_RATIO = 0.04;


# Queue::POSIX for multiple CPUs on a local machine

# Queue::PBS to use a PBS/TORQUE queue

$CFG_QUEUE_TYPE = "Queue";


# Name of queue to use for PBS/TORQUE

$CFG_QUEUE_NAME = "workq";


# (yes/no) Build questions for decision tree clustering automatically

$CFG_MAKE_QUESTS = "yes";

# If CFG_MAKE_QUESTS is yes, questions are written to this file.

# If CFG_MAKE_QUESTS is no, questions are read from this file.

$CFG_QUESTION_SET="${CFG_BASE_DIR}/model_architecture/${CFG_EXPTNAME}.tree_questi
ons";
```

```
#$CFG_QUESTION_SET = "${CFG_BASE_DIR}/linguistic_questions";


$CFG_CP_OPERATION= {CFG_BASE_DIR}/model_architecture/${CFG_EXPTNAME}.cpmeanvar";


# This variable has to be defined, otherwise utils.pl will not load.

$CFG_DONE = 1;


return 1;
```

Appendix B: Decoding and Live Recognition configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<config>

   <!-- ******************************************************** -->

   <!-- frequently tuned properties                              -->

   <!-- ******************************************************** -->

   <property name="absoluteBeamWidth"  value="15000"/>

   <property name="relativeBeamWidth"  value="1E-80"/>

   <property name="absoluteWordBeamWidth" value="200"/>

   <property name="relativeWordBeamWidth" value="1E-80"/>

   <property name="wordInsertionProbability" value=".5"/>

   <property name="languageWeight" value="7.5"/>

   <property name="silenceInsertionProbability" value=".1"/>

   <property name="frontend" value="mfcFrontEnd"/>

   <property name="recognizer" value="recognizer"/>

   <property name="showCreations" value="false"/>

   <property name="logLevel" value="INFO"/>


   <!-- ******************************************************** -->

   <!-- batch tool configuration                                 -->

   <!-- ******************************************************** -->


  <component name="batch"

                        type="edu.cmu.sphinx.tools.batch.BatchModeRecognizer">

                <propertylist name="inputDataProcessors">

                        <item>streamDataSource</item>

                </propertylist>
```

```
                   <property name="skip" value="0"/>
                   <property name="recognizer" value="${recognizer}"/>
                 </component>


<!-- ******************************************************** -->
<!-- live mode                                                -->
<!-- ******************************************************** -->


<component name="live" type="edu.cmu.sphinx.tools.live.LiveModeRecognizer">
  <property name="recognizer" value="${recognizer}"/>
  <property name="inputSource" value="concatDataSource"/>
</component>
<!-- ******************************************************** -->
<!-- word recognizer configuration                            -->
<!-- ******************************************************** -->


<component name="recognizer"
                   type="edu.cmu.sphinx.recognizer.Recognizer">
    <property name="decoder" value="decoder"/>
    <propertylist name="monitors">
        <item>accuracyTracker </item>
        <item>speedTracker </item>
        <item>memoryTracker </item>
        <item>recognizerMonitor </item>
    </propertylist>
</component>
```

```
<!-- ******************************************************** -->
<!-- The Decoder   configuration                            -->
<!-- ******************************************************** -->

<component name="decoder" type="edu.cmu.sphinx.decoder.Decoder">
    <property name="searchManager" value="wordPruningSearchManager"/>
    <property name="featureBlockSize" value="50"/>
</component>

<!-- ******************************************************** -->
<!-- The Search Manager                                     -->
<!-- ******************************************************** -->

<component name="wordPruningSearchManager"
type="edu.cmu.sphinx.decoder.search.WordPruningBreadthFirstSearchManager">
    <property name="logMath" value="logMath"/>
    <property name="linguist" value="lexTreeLinguist"/>
    <property name="pruner" value="trivialPruner"/>
    <property name="scorer" value="threadedScorer"/>
    <property name="activeListManager" value="activeListManager"/>
    <property name="growSkipInterval" value="0"/>
    <property name="checkStateOrder" value="false"/>
    <property name="buildWordLattice" value="false"/>
    <property name="acousticLookaheadFrames" value="1.7"/>
    <property name="relativeBeamWidth" value="${relativeBeamWidth}"/>
                  <property name="keepAllTokens" value="true"/>
</component>
```

```
<!-- ******************************************************** -->
<!-- The Active Lists                                         -->
<!-- ******************************************************** -->

<component name="activeListManager"
         type="edu.cmu.sphinx.decoder.search.SimpleActiveListManager">
    <propertylist name="activeListFactories">
  <item>standardActiveListFactory</item>
  <item>wordActiveListFactory</item>
  <item>wordActiveListFactory</item>
  <item>standardActiveListFactory</item>
  <item>standardActiveListFactory</item>
  <item>standardActiveListFactory</item>
</propertylist>
 </component>

<component name="standardActiveListFactory"
         type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
    <property name="logMath" value="logMath"/>
    <property name="absoluteBeamWidth" value="${absoluteBeamWidth}"/>
    <property name="relativeBeamWidth" value="${relativeBeamWidth}"/>
</component>

<component name="wordActiveListFactory"
         type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
    <property name="logMath" value="logMath"/>
    <property name="absoluteBeamWidth" value="${absoluteWordBeamWidth}"/>
    <property name="relativeBeamWidth" value="${relativeWordBeamWidth}"/>
</component>


<!-- ******************************************************** -->
<!-- The Pruner                                  -->
<!-- ******************************************************** -->
<component name="trivialPruner"
        type="edu.cmu.sphinx.decoder.pruner.SimplePruner"/>


<!-- ******************************************************** -->
<!-- TheScorer                                   -->
<!-- ******************************************************** -->
<component name="threadedScorer"
        type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
    <property name="frontend" value="${frontend}"/>
              <property name="isCpuRelative" value="true"/>
    <property name="numThreads" value="0"/>
    <property name="minScoreablesPerThread" value="10"/>
    <property name="scoreablesKeepFeature" value="true"/>
</component>


<!-- ******************************************************** -->
<!-- The linguist  configuration                       -->
<!-- ******************************************************** -->


<component name="lexTreeLinguist"
        type="edu.cmu.sphinx.linguist.lextree.LexTreeLinguist">
        <property name="logMath" value="logMath"/>

        <property name="acousticModel" value="AmharicDictator"/>

        <property name="languageModel" value="trigramModel"/>

                        <!-- property name="languageModel" value="bigramModel"/-->

        <property name="dictionary" value="dictionary"/>

        <property name="addFillerWords" value="false"/>

        <property name="fillerInsertionProbability" value=".2"/>

                        <property name="unitInsertionProbability" value="1.0"/>

        <property name="generateUnitStates" value="false"/>

        <property name="wantUnigramSmear" value="true"/>

        <property name="unigramSmearWeight" value="1"/>

                        <property name="cacheSize" value="0"/>

        <property name="wordInsertionProbability"

            value="${wordInsertionProbability}"/>

        <property name="silenceInsertionProbability"

            value="${silenceInsertionProbability}"/>

        <property name="languageWeight" value="${languageWeight}"/>

        <property name="unitManager" value="unitManager"/>

    </component>


                        <component name="textAlignGrammar"
type="edu.cmu.sphinx.linguist.language.grammar.TextAlignerGrammar">

        <property name="dictionary" value="dictionary"/>

        <property name="logMath" value="logMath"/>

    </component>


    <!-- ******************************************************** -->

    <!-- The Dictionary configuration                             -->
```

```
<!-- ******************************************************** -->
<component name="dictionary"
   type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
   <property name="dictionaryPath"
           value="models/acoustic/Dictation/dict/Dictation.dic"/>
   <property name="fillerPath"
       value="models/acoustic/Dictation/dict/Dictation.filler"/>
   <property name="addSilEndingPronunciation" value="false"/>
   <property name="wordReplacement" value="&lt;sil&gt;"/>
                 <property name="allowMissingWords" value="false"/>
                 <property name="createMissingWords" value="true"/>
   <property name="unitManager" value="unitManager"/>
</component>

<!-- ******************************************************** -->
<!-- The Language Model configuration                         -->
<!-- ******************************************************** -->
<component name="trigramModel"
     type="edu.cmu.sphinx.linguist.language.ngram.large.LargeTrigramModel">
   <property name="unigramWeight" value="0.7"/>
   <property name="maxDepth" value="3"/>
   <property name="logMath" value="logMath"/>
   <property name="dictionary" value="dictionary"/>
   <property name="location"
    value="models/language/Dictation_train_test_tri_sorted.DMP"/>
                 <property name="wordInsertionProbability" value="${wordInsertionProbability}"/>
   <property name="languageWeight" value="${languageWeight}"/>
   </component>


                    <!-- ************************************************** -->
   <!-- bigramModel                                 -->
   <!-- ************************************************** -->
 <!--
   <component name="bigramModel"
         type="edu.cmu.sphinx.linguist.language.ngram.large.LargeTrigramModel">
       <property name="unigramWeight" value=".5"/>
       <property name="maxDepth" value="2"/>
       <property name="logMath" value="logMath"/>
       <property name="dictionary" value="dictionary"/>
       <property name="location"
        value="models/language/Dictation_train_test_tri_sorted.DMP"/>
                      <property name="wordInsertionProbability" value="${wordInsertionProbability}"/>
       <property name="languageWeight" value="${languageWeight}"/>
   </component> -->


   <!-- ******************************************************** -->
   <!-- The acoustic model configuration                        -->
   <!-- ******************************************************** -->
   <component name="AmharicDictator"
              type="edu.cmu.sphinx.linguist.acoustic.tiedstate.TiedStateAcousticModel">
       <property name="loader" value="sphinx3Loader"/>
       <property name="unitManager" value="unitManager"/>
   </component>
```

```
    <component name="sphinx3Loader"
type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loader">

        <property name="logMath" value="logMath"/>

        <property name="unitManager" value="unitManager"/>

        <property name="location" value="models/acoustic/Dictation"/>

                        <property name="dataLocation" value=""/>

    </component>


    <!-- ******************************************************** -->

    <!-- The unit manager configuration                          -->

    <!-- ******************************************************** -->


    <component name="unitManager"

        type="edu.cmu.sphinx.linguist.acoustic.UnitManager"/>


    <!-- ******************************************************** -->

    <!-- The Batch decoding frontend configuration               -->

    <!-- ******************************************************** -->


    <component name="mfcFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">

      <propertylist name="pipeline">

                        <item>microphone </item>

        <item>streamDataSource</item>

        <item>preemphasizer</item>

        <item>windower</item>

        <item>fft</item>

        <item>melFilterBank</item>

        <item>dct</item>
        <item>batchCMN</item>

        <item>featureExtraction</item>

      </propertylist>

    </component>

                    <component name="microphone"

            type="edu.cmu.sphinx.frontend.util.Microphone">

        <property name="closeBetweenUtterances" value="false"/>

    </component>

    <component name="preemphasizer" type="edu.cmu.sphinx.frontend.filter.Preemphasizer"/>

    <component name="windower"
type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower"></component>

    <component name="fft" type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform"/>

    <component name="melFilterBank"
type="edu.cmu.sphinx.frontend.frequencywarp.MelFrequencyFilterBank">

                     <property name="minimumFrequency" value="133.3334"/>

        <property name="maximumFrequency" value="6855.4976"/>

        <property name="numberFilters" value="40"/>

                    </component>

    <component name="dct" type="edu.cmu.sphinx.frontend.transform.DiscreteCosineTransform"/>

    <component name="batchCMN" type="edu.cmu.sphinx.frontend.feature.BatchCMN"/>

    <component name="featureExtraction"
type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>

    <component name="streamDataSource" type="edu.cmu.sphinx.frontend.util.StreamDataSource">

      <property name="sampleRate" value="16000"/>

      <property name="bigEndianData" value="false"/>

      <property name="signedData" value="true"/>

      <property name="bitsPerSample" value="16"/>

    </component>

    <component name="dataDumper" type="edu.cmu.sphinx.frontend.util.DataDumper"/>
```

```
  <!-- ******************************************************** -->
  <!--  monitors                                   -->
  <!-- ******************************************************** -->


  <component name="accuracyTracker"
            type="edu.cmu.sphinx.instrumentation.BestPathAccuracyTracker">
     <property name="recognizer" value="${recognizer}"/>
     <property name="showRawResults" value="true"/>
     <property name="showAlignedResults" value="true"/>
     <property name="logLevel" value="INFO"/>
  </component>


  <component name="memoryTracker"
            type="edu.cmu.sphinx.instrumentation.MemoryTracker">
     <property name="recognizer" value="${recognizer}"/>
<property name="showDetails" value="true"/>
<property name="showSummary" value="true"/>
  </component>


  <component name="speedTracker"
            type="edu.cmu.sphinx.instrumentation.SpeedTracker">
     <property name="recognizer" value="${recognizer}"/>
     <property name="frontend" value="${frontend}"/>
<property name="showDetails" value="false"/>
  </component>
```

```
    <component name="recognizerMonitor"
                type="edu.cmu.sphinx.instrumentation.RecognizerMonitor">
        <property name="recognizer" value="${recognizer}"/>
        <propertylist name="allocatedMonitors">
            <item>configMonitor </item>
        </propertylist>
    </component>

    <component name="configMonitor"
                type="edu.cmu.sphinx.instrumentation.ConfigMonitor">
        <property name="showConfig" value="true"/>
    </component>

    <!-- ******************************************************* -->
    <!--  Miscellaneous components                             -->
    <!-- ******************************************************* -->

    <component name="logMath" type="edu.cmu.sphinx.util.LogMath">
        <property name="logBase" value="1.0001"/>
        <property name="useAddTable" value="true"/>
    </component>
</config>
```

Appendix C: sphinx_fe parameters used to create feature files

-alpha	0.97
-blocksize	2048
-c	D:\CS-MSC-MATERIALS-05\Thesis\Sphinx\Dictation\etc\ Dictation_train.fileids
-di	D:\CS-MSC-MATERIALS-05\Thesis\Sphinx\Dictation\wav
-dither	no
-do	D:\CS-MSC-MATERIALS-05\Thesis\Sphinx\Dictation\feat
-ei	wav
-eo	mfc
-frate	100
-lowerf	133.33334
-mswav	yes
-ncep	13
-nchans	1
-nfft	512
-nfilt	40
-nist	no
-ofmt	sphinx
-raw	no
-runlen	-1
-samprate	16000
-upperf	6855.4976
-wlen	0.025625

Appendix D: List of Main Java class files and instance variables of prototype

No	Class Name/instance Variable	Description
1	BRANA	The main class for the GUI. Have push button that initiates recognition the user dictates a sentence then it will be sent to the recognizer and finally return result to user. It extends JFrame to provide the supporting frame for the word processor
2	SimpleFilter	A class that extends file filter to filter rtf file formats
3	SmallButton	A class that extends Jbutton and implements MouseListener used to create buttons for the prototype
4	Splash	A class that will popup a splash screen when the prototype is activated
5	JTextPane m_monitor	An instance variable that is used to declare text component
6	StyleContext m_context	An insance variable that used for group of styles and their associated resources for the documents in the word processor
7	DefaultStyledDocument m_doc	An instance variable used to represent the current document model
8	RTFEditorKit m_kit	An instance variable for declaring editor kit that knows how to read/write RTF documents
9	JFileChooser m_chooser	An instance variable for file chooser used to load and save RTF files
10	SimpleFilter m_rtfFilter	An instance variable for file filter to ".rtf" files
11	JToolBar m_toolBar	An instance variable for toolbars containing “ክፈት”,”አስቀምጥ”, and “አዲስ” document buttons