University of Cape Town

The copyright of this thesis rests with the University of Cape Town. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

An Artificial Intelligence Approach to Improving Speech Recognition

Prepared by: Luis Ramos dos Santos Lopes

Supervised by: Prof. Daniel Mashao and Neco Ventura

Speech Technology and Research Group
Department of Electrical Engineering
University of Cape Town

March 2009

University of Cape Town

This dissertation is submitted to the University of Cape Town in fulfilment of the academic requirements for the Degree of Master of Science in Engineering

Statement of Originality

The work in this thesis is based on research carried out in the Speech Technology and Research Group, UCT, South Africa. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Signature of Author : ....
Luis Ramos dos Santos Lopes
March 2009

University of Cape Town

Copyright © 2009 by Luis Ramos dos Santos Lopes.
"The copyright of this thesis rests with the author. No quotations from it should be published without the author's prior written consent and information derived from it should be acknowledged".

Acknowledgements

I would like to thank my supervisors Prof. Daniel Mashao and Neco Ventura for their support and for making this research possible; the Speech Technology and Research (STAR) group and the Centre for Broadband Networks at the University of Cape Town for supporting this research; and Phillippa Wilson who kindly provided her assistance in submitting this thesis remotely.

University of Cape Town

Abstract

Speech Recognition is a technology with promising applications. However, the performance of current speech recognizers greatly limit their widespread use. Approaches to reducing the word error rate have mainly been associated with statistical techniques. As a consequence, speech recognition results can still contain sentences that are nonsensical. The method proposed here, is to analize the output of any chosen speech recognition system, in order to determine whether a sentence contains syntactic or semantic errors. This is done via a software agent that uses the information from its knowledge base to attempt to correct the errors found. A system was implemented with a small vocabulary speaker-independent continuous speech recognition system, with limited sentence structures. The achieved increase in speech recognition accuracy, shows that there are benefits in using this approach.

University of Cape Town

Contents

Declaration ii

Acknowledgements iii

Abstract iv

1 Introduction 1

1.1 Research Motivation 2

1.2 Research Objectives 3

1.3 Overview of Proposed System 4

1.4 Thesis Outline 5

2 Background on Speech Recognition Technology 7

2.1 Overview 7

2.2 The Front End 8

2.3 The Decoder 9

2.4 Speech Recognition Errors 10

2.5 The Challenges 11

3 Methods for Improving Speech Recognition 13

3.1 Enhancing Speech Recognition Engines 13

3.1.1 Language Models and Grammars 13

3.1.2 Using dialogue context and knowledge 16

3.1.3 Imitating Spreading Activation 16

3.1.4 Hybrid Artificial Neural Networks and HMM models 17

3.1.5 Automatic Diagnosis of Errors 18

University of Cape Town

3.2 Post-Processing Speech Recognition Results 18
3.2.1 Identifying the Best Recognition Hypothesis with a knowledge base 18
3.2.2 Error Detection Using Semantic Similarity 18
3.2.3 Error Correction Based on Co-Occurrence Statistics 19
3.2.4 Error Correction with a Noisy Channel Model 20
3.2.5 Semantic Oriented Error Correction 22
3.2.6 Correction with Common Sense Knowledge 22
3.3 Other Approaches 23

4 Implementation of the Sphinx-4 Speech Recognition System 25
4.1 Introduction to Sphinx-4 25
4.2 Installing Sphinx-4 27
4.3 The Acoustic Model and Dictionary 27
4.4 The Language Model 28
4.5 Configuration of the Sphinx-4 Applications 29

5 The Software Agent 33
5.1 Overview 33
5.2 Scope and Limitations 34
5.3 Use Case Model 35
5.3.1 Use Case Diagram 36
5.3.2 Process Sentence 36
5.3.3 Check Syntax 37
5.3.4 Check Logic 37
5.3.5 Correct Sentence 37
5.3.6 Read Results 37
5.4 The Genese Package 38
5.5 Knowledge Base 39
5.5.1 Meaning Representation 41
5.5.2 Dictionary 42
5.5.3 Classification Database 42

University of Cape Town

5.5.4 Verb Database . . . 43
5.5.5 Validation Knowledge Base . . . 43
5.5.6 Common Sense Knowledge Base . . . 45
5.6 System Activity Diagram . . . 45
5.6.1 Initial Processing . . . 46
5.6.2 Deletion Error . . . 48
5.6.3 Deletion Case 1 . . . 48
5.6.4 Deletion Case 2 . . . 50
5.6.5 Deletion Case 3 . . . 51
5.6.6 Correct Sentence . . . 53
5.6.7 Find Matching Word . . . 54
5.6.8 Check and Fix Logic Error . . . 56
5.7 Processing Examples . . . 58
5.7.1 Syntax Error Example . . . 58
5.7.2 Deletion Error Case 1 Example . . . 60
5.7.3 Deletion Error Case 2 Example . . . 61
5.7.4 Deletion Error Case 3 Example . . . 62
5.7.5 Semantic Error Example . . . 62
5.8 Source Code . . . 63
5.9 The Genese Applications . . . 67

6 Integrating the Intelligent Agent with Sphinx-4 71

6.1 Integration with Sphinx-4 . . . 71
6.2 The PlainSpeech Application . . . 72
6.3 The SmartSpeech Application . . . 72
6.4 The WavToText Application . . . 74

7 Experimental Results 78

7.1 User Survey . . . 78
7.2 Experiments with SmartSpeech . . . 80
7.3 Experiments with WavToText . . . 82
7.4 Adjusting the Knowledge Base . . . 83

University of Cape Town

8 Conclusions **85**

8.1 Conclusions 85

8.2 Comparison with Other Research 86

8.3 Recommendations 86

Bibliography **88**

Appendix **94**

A Sample of Language Model **94**

B Sphinx-4 Configuration **100**

C Common Sense Knowledge Base **105**

D Test Sentences **110**

E Contents of CD-ROM **112**

University of Cape Town

List of Figures

1.1 Block Diagram of Proposed System 5

2.1 Basic Diagram of a Speech Recognition System 8

4.1 Architecture of the Sphinx-4 speech recognition engine [47] 26

5.1 Class Diagram for the Genese Package 40
5.2 Activity Diagram for Initial Processing 47
5.3 Deletion Error Activity Diagram 49
5.4 Deletion Case 1 Activity Diagram 50
5.5 Deletion Case 2 Activity Diagram 52
5.6 Deletion Case 3 Activity Diagram 53
5.7 Correct Sentence Activity Diagram 54
5.8 Find Matching Word 57
5.9 Check and Fix Logic Error 59
5.10 Example of Syntax Error Detection 59
5.11 Snapshot of ConsoleInteraction 69
5.12 Snapshot of VisualInteraction 70

6.1 Snapshot of the SmartSpeech Application 73

7.1 Comparison of Original WER and SmartSpeech WER 81
7.2 Original WER and SmartSpeech Reduction 81

University of Cape Town

List of Tables

2.1 Example of contents of dictionary 9

5.1 Knowledge Base Files 41

5.2 Ranking of Common Sense Knowledge Base Files 56

5.3 CSKB Lookup to Correct Syntax Error 60

5.4 VKB Lookup for Deletion Error Case 1 Example 60

5.5 CSKB Lookup For Deletion Error Case 1 Example 61

5.6 Matches for Deletion Error Case 1 Example 61

5.7 Search and Matches for Deletion Error Case 2 Example 62

5.8 Search and Matches for Deletion Error Case 3 Example 62

5.9 VKB Lookup for Semantic Error Example 63

5.10 CSKB Lookup for Semantic Error Example 63

5.11 Matches for Semantic Error Example 63

7.1 Examples of Corrections Made by SmartSpeech 79

7.2 Word Error Rate Reduction for Several Tests 80

7.3 Sentence Error Rate Reduction for Several Tests 81

7.4 Original WER and WavToText Reduction 82

University of Cape Town

Chapter 1

Introduction

University of Cape Town

Electronic Engineering and Computer Science are the main fields that deal with automatic speech recognition (ASR) by machine. This is evident considering, for example, that signal processing and computer algorithms are essential technologies for this application. Given that speech recognition enables spoken language to be recognized by machines, it is not difficult to associate it with linguistics as well. Nevertheless, speech recognition can arguably benefit from many other branches of knowledge, such as physics, physiology or even psychology. The interdisciplinary nature of speech recognition, is thought to be one of the reasons that make it difficult to reach the level of desired perfection [43]. In particular, better collaboration between experts in linguistics, artificial intelligence, including natural language processing and understanding, statistics and digital signal processing, would likely be beneficial.

The speech recognition research community seems to have acknowledged the importance of taking into account the complex characteristics of spoken language, and we can see in chapter 3, several efforts in incorporating syntax and semantic constraints in speech recognition, with encouraging results. However, many modern speech recognizers still produce nonsensical sentences, even when there is an obvious better choice of output. Although some work has been done to directly address this deficiency, as discussed in 3.2.6; by comparison with the amount of effort put on other methods, we are not aware of much research being done to impart ASR systems the ability to recognize speech using human knowledge in a logical manner.

The research work described in this thesis, is another step in this direction.

This chapter describes the motivation behind the approach taken on this research, the research objectives, and gives an overview of the proposed system.

1.1 Research Motivation

After several decades of research by universities, institutions, commercial companies and other organizations, the performance of speech recognition systems in many real-world applications still leaves much to be desired. Since speech is a human ability, we contemplated on the idea of incorporating some of the mechanisms by which we recognize speech, into ASR systems. In particular, we considered applying human knowledge and common sense reasoning to improve recognition accuracy.

Intuitively, one knows that listening and transcribing a discourse in a topic that we are familiar with, is much easier than carrying out the same task on a topic that we have little knowledge of, even if all or most of the words are part of our mental vocabulary. If a person is being dictated a list of unconnected words over the telephone, she could easily confuse two similar sounding words, such as "crown" and "brown". However, if a list of sentences are being dictated, context would make it easier to disambiguate between the two words. It became apparent, therefore, that common sense reasoning, seems to play an important part in the speech recognition process by humans. Replicating this ability artificially, could lead to improvement in the accuracy of ASR systems.

Several months were spent studying and researching the various techniques used to improve speech recognition, while learning more about artificial intelligence and linguistics, with the objective of applying this knowledge to the task of reducing speech recognition errors. While studying linguistics, we found an interesting paragraph on the subject of deaf people being able to understand speech. Fromkim *et al.* [45] stated that as much as seventy-five percent of English words cannot be accurately read on the lips. They comment on how remarkable it is that many deaf individuals are able to comprehend speech, by combining lip reading with knowledge of the structure of language, context, and the meaning redundancies that language

University of Cape Town

has.

It is obvious from the preceding paragraph, that knowledge of language, and reasoning based on context is essential for language understanding, even when many of the spoken words cannot be correctly recognized. This is particularly relevant for dialogue systems, where the meaning of a user's utterance is more important than the actual string of words spoken, as it can be seen in 3.1.2 on page 16. The work of Savage *et al.* [27] also demonstrates this point. We reasoned that the same principles could be applied to improving the word accuracy of speech recognition systems and we decided to focus our research on this approach.

1.2 Research Objectives

The main purpose of this research was to identify a method to improve speech recognition accuracy, currently receiving less attention from the speech recognition research community. Motivated by the factors given in the preceding section, we decided to concentrate our research in developing a technique that made use of common sense knowledge and language structure, to improve speech recognition. During the course of the research, the idea eventually evolved into a specific project, aimed at investigating the benefits of this approach, and implementing it in a live, real time speech recognition system. The goals of the project were as follows:

- Design and develop an intelligent software agent, or system, to analyze the final output of a given speech recognizer, identify any possible error in syntax and semantics, and correct the error, if an obvious, logical sentence exists.
- Design the specifications of the above system with realistic constraints, taking into account the limited development time available, while still proving the usefulness of key concepts.
- Design and implement a knowledge base, to be used by the intelligent agent, for the purpose of detecting semantic errors, and for correcting errors when possible.

University of Cape Town

- Integrate the intelligent agent with a speech recognizer; develop an application to transcribe recorded speech, and a dictation application made to recognize speech in real time, both using the integrated system in a way that is transparent to the user.

- Test the system, using both the transcriber and dictation applications above, in order to determine how much improvement, if any, is made with the system, by comparing the output of the software agent, which is seen by the user, with the output of the baseline recognizer, which serves as the input to the software agent.

The speech recognition engine chosen was Sphinx-4. Taking into consideration the time constraints, a restriction was placed on the sentence structure that can be processed by the intelligent agent, with only two specific grammar rules allowed. Since the associated knowledge base would have to be built from scratch, it was expected that the tests would be conducted on a small vocabulary set.

University of Cape Town

1.3 Overview of Proposed System

Given the objectives outlined in the preceding section, we designed a system with a block diagram shown in Figure 1.1 on page 5. The speech recognition engine converts a spoken utterance into text. The block labeled "software agent" is the module developed from this research. It processes the output of the speech recognizer to detect syntax or semantic errors, and attempts to correct any errors found, using the data in the knowledge base. For example, if the input text is "the cat ate the that" the software agent will inform the application that a syntax error was detected. It will also provide a suggested correct sentence. Using the information on our knowledge base the corrected sentence would become "the cat ate the rat".

Throughout this document the terms "the intelligent software agent", "the software agent", "the intelligent agent", "the agent", or within certain contexts simply "the software"; may all be used interchangeably to refer to the software component developed from this research, generally in line with the definition of a "rational agent" used by Russell and Norvig [49].

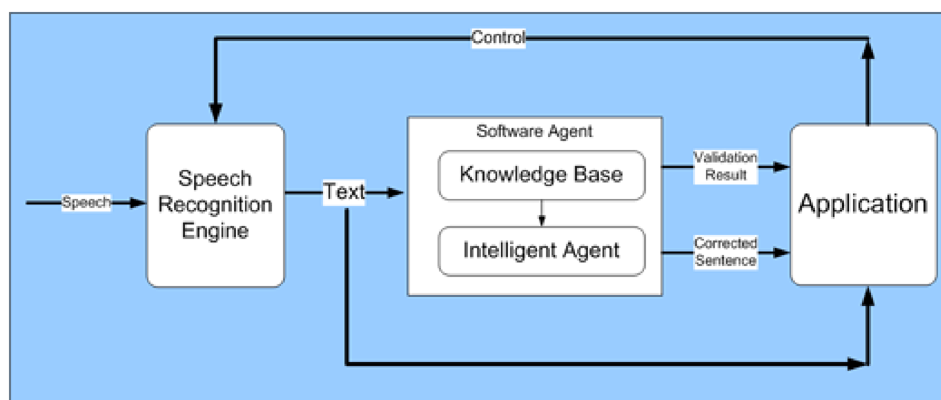


Figure 1.1: Block Diagram of Proposed System

The purpose of the software agent is to detect recognition errors, and to repair nonsensical sentences by guessing what the original utterance was using common sense knowledge, in a similar way that a human would. The result, is an overall increase in speech recognition accuracy. The speech application which makes use of the software agent, controls the ASR engine, and can request the user to repeat the sentence, if an intelligent guess could not be made.

1.4 Thesis Outline

A brief description of the content of this thesis follows. In Chapter 2 an overview of speech recognition technology is given, to provide the necessary background for the remainder of the document.

In Chapter 3 we provide a literature review of research that has been done to improve speech recognition accuracy, with particular emphasis on methods that attempt to use syntactic and semantic constraints. In Chapter 4 we give a brief overview of the Sphinx-4 speech recognition system, and provide details of how it was implemented and configured for our specific application.

A detailed description of the software agent designed to detect and correct speech recognition errors is given in chapter 5, with reference to diagrams and the accompanying source code. A discussion of the integration of the software agent with Sphinx-4, and the two applications developed to test the system is given in chapter 6.

University of Cape Town

Experimental results from tests conducted with the system using the two applications are presented in chapter 7, and conclusions given in chapter 8. The appendices contain additional information.

University of Cape Town

Chapter 2

Background on Speech Recognition Technology

University of Cape Town

In this chapter, a brief and high-level overview of speech recognition technology is presented, in order to provide some background to the rest of this thesis.

2.1 Overview

Automatic speech recognition (ASR) by machine, consists of finding the string of words that correspond to an utterance captured with a microphone. A simplified diagram for speech recognition is shown in figure 2.1 on the next page.

After being converted into an electronic signal by a microphone, the speech is first processed by the front end. All of the digital signal processing and feature extraction is done by this block. Next, the features, which could be in the form of mel frequency cepstrum coefficients (MFCC) are sent to the decoder, which searches for the most likely sequence of words, based on the information given by the dictionary, acoustic model and language model. The text can then be further processed for errors, or sent directly to the application, such as a document generator or a language understanding system.

The above description is for a hidden Markov model (HMM) based system, which is the paradigm used by most modern speech recognizers. In mathematical terms, the speech recognition problem can therefore be expressed as follows [46]:

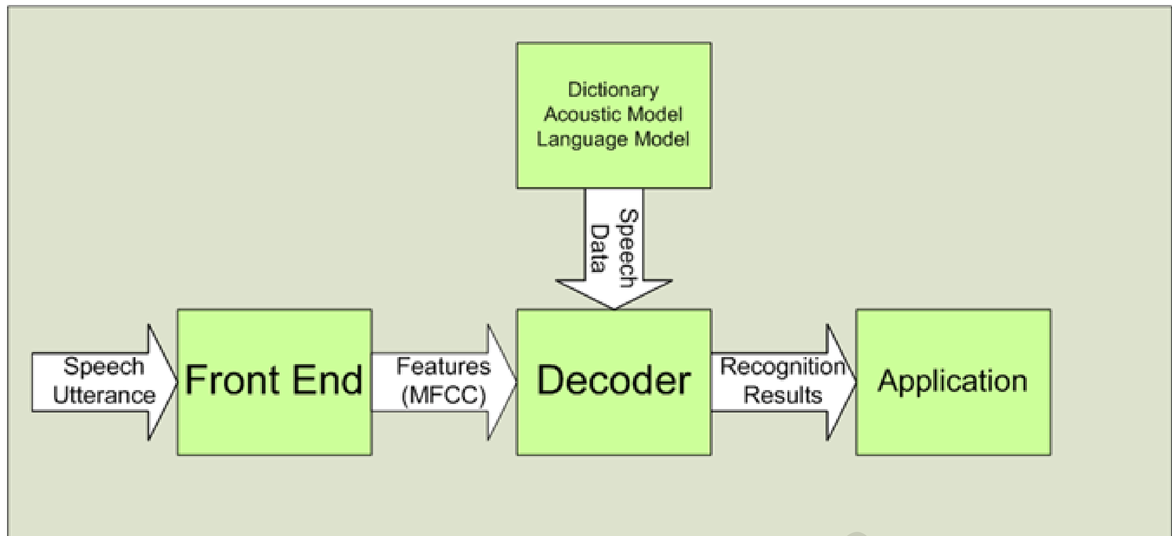


Figure 2.1: Basic Diagram of a Speech Recognition System

$$\hat{W} = arg\,\max_{W}\, P(W \mid A) \tag{2.1}$$

Equation 2.1 means that the speech recognizer will search for the most likely sequence of words $\hat{W}$ given the acoustic evidence A.

Other approaches to speech recognition include the acoustic-phonetic approach and artificial neural networks (ANNs) [43].

2.2 The Front End

The front end is responsible for the initial processing of the signal in order to extract the features of the utterance. The sound pressures of the spoken words are converted into a corresponding analog electronic signal, which is sampled and converted into a digital stream. For speech recognition software running on desktop or portable computers, the sampling and analog to digital conversion is done by the hardware on the host computer, usually via a microphone plugged directly into a sound card, or via a USB port. The front end in such cases, will therefore consist only of software that applies digital signal processing (DSP) techniques on the digital signal.

For speech recognition, it is not the varying amplitude of the signal corresponding to the sound wave form, or in other words, the time domain characteristics of the

University of Cape Town

BANANA	B AH N AE N AH
MONKEY	M AH NG K IY
ULTRASOUND	AH L T R AH S AW N D
PRESIDENT	P R EH Z AH D EH N T
PEACE	P IY S

Table 2.1: Example of contents of dictionary

signal, that are important, as it might be intuitively thought at first. Rather, a frequency domain representation of the signal is obtained, in order to determine the energy content of relevant frequencies. This is the essence of feature extraction done by the front end, which will typically produce a vector of mel frequency cepstrum coefficients (MFCC), although other representations are possible.

University of Cape Town

2.3 The Decoder

In the acoustic model of a typical HMM based ASR system, an HMM is used for each acoustic unit, usually a phoneme. A phoneme is a sub-unit of the sound required to constitute a word. In some cases, each entire word is modeled by an HMM, but this is more practical in small vocabulary applications. The dictionary of the system maps a set of phonemes to each individual word in the vocabulary. An example is shown in table 2.1.

The HMM parameters of an acoustic model are estimated through a process called training, which uses a speech corpus, typically consisting of a large amount of recorded sentences and their corresponding transcriptions. A larger amount of quality training data provides a better acoustic model, with a resulting increase in recognition accuracy. Speech corpora is usually created by having a person read several sentences, in a quiet room, with a high quality microphone, in a normal tone of voice. In order to create a speaker independent system, speech data is collected from different males and females.

From equation 2.1 on the preceding page, after applying Bayes' theorem, the following expression can be derived [46]:

$$\hat{W} = arg\,\max_{W} P(A\,|\,W)\,.\,P(W) \tag{2.1}$$

This means that the decoder takes into account the acoustic evidence observed, and the probability that the user uttered a set of words W. The latter is done by the language model. The simplest type of language model can take the form of a finite state grammar, in which every possible sequence of words is specified. This is obviously practical only for a very small vocabulary set. A context-free grammar expresses possible word sequences by a set of rules.

The most commonly used type of language models are *n-gram* models, which consist of establishing the probability of a word, based on the previously uttered words. For example, in a trigram (n=3) model, the probability of a word being spoken given that two specific words were previously spoken, is calculated. These probabilities are established by analyzing a corpus of text. For optimal performance, the topic of the chosen text corpus should be related to the intended application. For example, if a speech recognition system is to be used to transcribe news reports, a text corpus of past news reports should be used.

The decoder searches through the possible word sequences, using the grammar or language model to restrict the search space, and produces a word lattice or N-best list of hypotheses. The output is usually the 1-best hypothesis, although, as we can see in chapter 3, some implementations use further algorithms to re-rank the list and pick the best hypothesis through the use of additional constraints or knowledge sources.

University of Cape Town

2.4 Speech Recognition Errors

The types of errors that automatic speech recognition engines make, can be classified into three categories: substitution errors, deletion errors, and insertion errors.

A substitution error occurs when the ASR engine incorrectly recognizes a word. For example, if the speaker uttered the sentence “I have a dent in my car” but the ASR engine produces the text “I have a tent in my car”, the word ’tent’ is a substitution error. Substitution errors occur more often for words of phonetic similarity.

Deletion errors refer to words that are missing from the recognized speech. For

the utterance above, if the output of the ASR engine is “have a dent in car”, then the words ’I’ and ’my’ constitute deletion errors. Short words, or words at the beginning of a sentence, may generate deletion errors more often.

An insertion error refers to an extra word in the recognition result. In our example above, if the ASR produced the text ’I will have a dent in my car’ then the word ’will’ is an insertion error.

The accuracy performance of speech recognition systems is usually measured by the word error rate (WER), which is the total number of substitution errors, deletion errors or insertion errors usually expressed as a percentage of the total number of uttered words in a recognition task. For example, if 100 words were dictated to a speech recognition system, and the result contained 12 substitution errors, 1 insertion error and and 2 deletion errors, then the word error rate would be 15%. The ideal target for improving the accuracy of a speech recognition engine would be to reduce the WER to 0% under all conditions.

Another measure of accuracy is the sentence error rate, which is the number of sentences that had at least one error, usually expressed as a percentage of the total number of uttered sentences.

University of Cape Town

2.5 The Challenges

To understand the nature of speech recognition errors, it may be useful to think about the reasons humans sometimes misunderstand words. Ambient noise, different accents, people talking in the background, a noisy telephone line or similar sounding words are examples of factors that make it difficult for a person to accurately recognize speech. All of these factors also impact the accuracy of ASR engines, often to a much greater extent.

Current speech recognition systems do not handle adverse conditions as well as humans, although there has been considerable research on robust speech recognition. For instance, humans have the ability to follow a conversation in a party in the presence of background music, or speech from other talkers, by somehow filtering out the unwanted sounds and concentrating on the voice of the target speaker. This

remarkable ability is known as the 'cocktail party effect' [45].

In addition to intuitively obvious factors affecting recognition performance, speech recognition by machine is inherently vulnerable to other aspects of human speech. For example, speech corpora used to train acoustic models, are typically created with adults. As a result, speech recognition accuracy in these systems decreases considerably with children. Likewise, spontaneous speech, characterized by abrupt pauses, false starts, prosody and variations in pace can cause degradation in performance.

As another example, when a recognition error occurs, many users tend to repeat the sentence speaking very slowly or emphasizing the incorrectly recognized words. While this approach may be effective for a human listener, it actually makes recognition worse in the case of an ASR system, because the acoustic models were trained with normal speech. Humans are also less likely to make deletion or insertion errors when recognizing speech. A speech recognition engine, on the other hand, could decode an utterance of the word 'insult' as 'in salt'.

The seemingly random nature of speech, due to the virtually infinite number of variations it can take, is therefore the main reason behind the difficulty in creating a perfect speech recognition system that is accurate for all speakers in all conditions. Arguably, it is also the reason statistical methods have proved to be more successful in practical applications than other techniques.

University of Cape Town

Chapter 3

Methods for Improving Speech Recognition

University of Cape Town

A number of approaches have been taken to solve the difficult problems in speech recognition. In this chapter we provide an overview of some of the research that has been done to improve the accuracy of state-of-the-art speech recognition systems. It is not meant to be an exhaustive treatment of the different methods and techniques found in the literature, but rather, it is intended to provide an appreciation for the different ideas and possible research directions. There is some emphasis on methods that attempt to take advantage of lexical and semantic properties of natural languages to improve recognition accuracy.

3.1 Enhancing Speech Recognition Engines

3.1.1 Language Models and Grammars

Rosenfeld [1] mentions how current language models are sensitive to domain changes. If a language model is trained with corpora of a particular genre, using it for speech recognition on a different topic, often results in an increase in the word error rate. He discusses how several techniques are used to address some of the deficiencies of simple n-gram models, such as using decision tree models, exponential models, and adaptive models. He also discusses some promising new directions, including

dimensionality reduction, which takes into account the fact that many words are semantically related. Using this motivation for topic adaptation, Bellegarda [34] combined it with an n-gram and developed it further to obtain a 16% reduction in recognition errors over a trigram baseline.

In order to exploit syntactic and semantic properties in language, Wang *et al.* [8] have used a unified generative directed Markov random field model framework that combines n-gram models, probabilistic context free grammars and probabilistic latent semantic analysis. Experimental results on the Wall Street Journal corpus have shown that their composite language model can produce a significant reduction in perplexity over a state-of-the-art trigram model with Good-Turing and Kneser-Ney smoothing.

Grau *et al.* [10] have incorporated semantic knowledge into a language model for the speech recognition engine used in a dialogue system which provided slightly better word accuracy, and overall improved results in the understanding system. To incorporate the semantic knowledge, they used a methodology called Morphic Generator Grammatical Inference (MGGI) [42].

Ward and Issar [11] use a Recursive Transition Network (RTN) to integrate semantic constraints into the SPHINX-II recognizer. The RTN is used in the A* portion of the search, during the decoding process. The RTN scoring is combined with the scoring from the n-gram to produce the N-best list of hypotheses, which biases the system to prefer sentences accepted by the RTN but still allows word sequences that go against it, provided that there is strong evidence. The RTN grammar was optimized for the speech recognizer to be used for spontaneous speech in an air travel dialogue system, and to allow for better compatibility between the decoding and the understanding portion of the system. Experimental results have shown a 7% relative reduction in word error rate, a 8.8% reduction in sentence error, and more importantly, a 23% reduction in the understanding error.

The Gemini natural-language processing system was used by Dowding *et al.* [9] at SRI International, to incorporate natural language restrictions into the DECI-PHER speech recognition system, used in spoken language applications such as travel planning. Gemini parsed each of the N-best hypotheses returned by the rec-

University of Cape Town

ognizer, and detected the minimal number of grammatical fragments needed to span the hypothesis, in order to calculate a natural language score based on this number. A natural language score for each recognition hypothesis is calculated in this manner and combined with the recognizer score. The recognized sentence would be the one with highest overall score. Evaluation on the Air Travel Information Service (ATIS) task showed a 5% reduction in recognition error [12]. Later, Moore *et al.* [6] modified this approach slightly, by changing the way the natural language score is calculated. The analysis from Gemini is used to estimate the probability of the hypothesis with n-gram statistics, and the natural language score is based on the logarithm of this probability estimate. Under various tests, this kind of multi-level n-gram model, showed improvements over the previous approach, with error rate reductions of about 15% over the baseline recognizer.

University of Cape Town

Qualitative language models, such as context-free grammars and unification grammars, can impose natural-language constraints, but the common methods used to integrate them into the recognition search, namely word lattice parsing, N-best filtering or re-scoring, and dynamic generation of partial grammar networks, do not necessarily yield much better recognition accuracy than a well-trained n-gram model. Moore [19] presents a method that might be extended to incorporate all the constraints of a complex unification-based grammar into a fully statistical model. Acero *et al.* [35] proposed a Semantically Structured Language Model (SSLM), which takes advantage of both n-gram statistical models and context free grammars (CFGs). They concluded that it can reduce the authoring load of creating a speech understanding system, while providing better understanding accuracy and usability. Bühler [18] discusses a method for using language modeling to integrate speech recognition with a flat semantic analysis, for spoken language understanding systems.

For dialogue systems, it is possible to parse the sentence during the decoding stage, by means of a stochastic context free grammar (SCFG), similar to a stochastic recursive network (SRTN), which allows the decoder to output semantic concepts along with the words. A maximum entropy (ME) based semantic language model can also be used to reduce speech recognition errors [20].

3.1.2 Using dialogue context and knowledge

Sheryl Young [3] showed in the MINDS system how knowledge sources can enhance speech recognition accuracy. More specifically, she used dialogue-level contextual knowledge, to generate conceptual predictions about the next utterance. In order to prevent system rigidity, layered predictions are created, ranging from very specific to more general. The conceptual predictions are expanded into words in a grammar that is used to guide a modified version of the SPHINX speech recognition system. Since speech recognition engines use grammars to constrain the search space of possible words, the scope of the recognition results is confined to words that are more relevant to the context of the dialogue. Each new utterance provides a new context for another set of predictions about what is going to be said next, so the system generates grammars dynamically throughout the dialogue.

The MINDS system was designed to be used in a resource management domain, featuring information about ships in the United States Navy. The knowledge sources come from the dialogue, general world knowledge, a model of the task semantics and models of individual users [3].

Performance for the MINDS system was evaluated by using the speech recognition system with a standard fixed grammar as a baseline, and comparing the results obtained with the dynamically generated grammar using the layered predictions. The accuracy of the system using predictions was 96.5% against 85% with the standard grammar, which represents a decrease of 80% in the error rate. In addition, the semantic accuracy, defined as a correct SQL (Standard Query Language) query to the database, increased from 85% to 100% [3]. These results show that predicting the user's intention based on dialogue and task-related context, is a very effective technique for reducing speech recognition errors, and more importantly, for providing a more accurate input to the dialogue system.

3.1.3 Imitating Spreading Activation

Another way of using word co-occurrence statistics is for imitating spreading activation, which in psycholinguistic theories, could be the mechanism through which

University of Cape Town

the human mind perceives the most likely meaning of a word and makes semantic predictions about the words that are supposed to follow [5]. The concept behind the spreading activation theory is that in the human brain, activation of one node spreads to the immediate semantically or phonologically related neighbour nodes, but the most highly activated lexical node is selected, in order to avoid the polysemy problem. Stefan and Denisa [5] give an example of the word "star" in a sentence related to computer networking. The activation spreading from the nodes of the words "node", "connected", and "hub" reach the node "star" with the computer related meaning. Other meanings of the word "star", such as a celestial object, or celebrity, are not selected due to poor activation from their neighbouring nodes.

To imitate this effect, an analysis is performed on text corpora, in order to determine the co-occurrence statistics of words within a sentence and build semantic clusters of words that belong to the same topic or context. There will be overlaps as one word can exist in more than one cluster. The beginning words of a sentence can be used as input words. In another step, all clusters that are related to the topic of the sentence, are merged to form one large cluster [5]. It is assumed that this model could be incorporated into a speech recognition system.

University of Cape Town

3.1.4 Hybrid Artificial Neural Networks and HMM models

The use of artificial neural networks (ANN) has been as an interesting alternative to the HMM paradigm, however, practical implementations remain difficult due to their inability to model long sequences of acoustic observations required for speech recognition. Different architectures have therefore been proposed to implement a hybrid ANN/HMM system. Compared to standard HMM systems, reductions in WER ranging from about 16% to 26% have been reported in the literature. Perhaps the main advantage of these hybrid systems, is that ANNs require less parameters than HMMs and as such, are much easier to implement in hardware with VLSI technology [22].

3.1.5 Automatic Diagnosis of Errors

Nanjo *et al.* [25] have developed a method that provides an automatic diagnosis of speech recognition errors, by identifying the modules that cause the errors and pointing out the errors in detail. This information was used successfully to improve their decoder and to direct future research.

3.2 Post-Processing Speech Recognition Results

3.2.1 Identifying the Best Recognition Hypothesis with a knowledge base

Gurevych and Porzel [17] have used a knowledge-based scoring system to identify the best recognition hypothesis for a spoken dialogue system. The natural language parser that interprets the user's intention, is used to select the N-best list from the speech recognizer's word lattice, and assigns a score to each hypothesis. The list is passed to a knowledge-based scoring system which re-scores the list of hypotheses, based on how well the concepts represented by a sentence agree with the knowledge present in the system's ontology. The system also takes into account the previous utterance, to evaluate the context of the hypothesis.

To measure the system performance, human annotators were used to identify the best hypothesis, from a list of alternatives, for each sentence in a transcript. It was found that the speech recognition system correctly identified 83.88% of the best hypotheses, compared to 87.56% for the parsing module and 88.07% for the contextually enhanced knowledge-based scoring system [17].

3.2.2 Error Detection Using Semantic Similarity

Inkpen and Désilets [15] developed a system intended to filter out incorrectly recognized words from automatic audio transcripts. The idea was to completely remove from the transcript, content words that do not appear to be semantically coherent with the context, termed "semantic outliers", in order to make easier the task of obtaining the gist, or using the document to browse through the recorded audio.

University of Cape Town

They give as an example, the portion of text transcribed from the audio of a meeting with a commercial speech recognizer, "Weenie to decide quickly whether local for large expensive plasma screen aura for a bunch of smaller and cheaper ones and Holland together", for which the filtered transcript produced with their algorithm is " ... to decide quickly whether ... large expensive plasma screen ... for a bunch of smaller and cheaper ones and ... together". They believe that the filtered transcript will be more useful for getting the gist or browsing the audio recording, for which the correct transcript is "We need to decide quickly whether we will go for a large expensive plasma screen or for a bunch of smaller and cheaper ones and tile them together" [15].

To identify the semantic outliers, the algorithm makes use of the semantic similarity measure between words calculated in a window. Two methods were considered to obtain the semantic similarity measure. One is dictionary based, which can use path length in a semantic network such as WordNet, described in 3.2.6. The other is based on probabilities obtained from a large corpus. The PMI corpus based measure was preferred. From their experimental results, a 50% reduction of content word error rate[1] is obtained at the sacrifice of losing 50% of the good content words, for a certain threshold. They note that the algorithm is computationally expensive, which makes it less suitable for real-time applications. One possible direction mentioned, is to correct the errors by looking at the N-best list from the recognizer and choosing the hypothesis that maximizes semantic coherence [15].

Another somewhat related work was done by Hazen [24], who developed a system to align and correct approximate human generated transcripts for long speech recordings, using ASR. Preliminary experiments have produced a 12% reduction in error rate.

3.2.3 Error Correction Based on Co-Occurrence Statistics

A technique of error detection and correction applied for user-centered tasks such as information retrieval is used by Sarma and Palmer [2]. The detection and correction

[1] The concept of content word error rate or cWER is explained in [15].

University of Cape Town

is achieved by taking into account that certain context words tend to appear with a given query word, and with an incorrectly recognized phonetically similar version, of the same query word. The output of a given ASR is analyzed and co-occurrence statistics calculated for every word in the system's dictionary. This results in obtaining a set of context words associated with each word in the vocabulary. Given a target word, such as a query word in the case of an information retrieval system, they identify regions in the corpus, containing a large number of expected context words, associated to the query word. They use context analysis to identify regions in the corpus that are likely to contain the query word, and are therefore, also likely to contain a mis-recognition of the given query word. These different words that share a similar context as the query word, are considered candidate error words. By measuring the phonetic distance between a candidate error word and the query word, a decision can be made whether or not to correct it. Given that they share the same context, if the candidate error word is phonetically similar to the query word, there is a high likelihood of it being a mis-recognition of the query word.

As an example of the above, consider the phrase "...the reconstruction of a rock proceeds despite Chirac's refusal...". The word "rock" is at the centre of top-ranked words for "Iraq" making it a candidate error word. The strong phonetic similarity between "Iraq" and "a rock" is a good indication that "a rock" should be corrected to "Iraq" [2].

University of Cape Town

3.2.4 Error Correction with a Noisy Channel Model

Ringger and Allen [7] propose using a noisy channel model to process the output of a speech recognizer to reduce recognition errors. This technique, adapted from machine translation, consists of assuming that the system that begins with the speaker, and ends with the speech recognition engine which is seen in a black box fashion, is a noisy channel that may introduce errors, such that the actual sequence of spoken words $\underline{e}_{1,n} = (w1, w2, ..., wn)$, may be different to the sequence of words $\underline{e}'_{1,n'}$ at the output of the channel. An error correcting post-processor is therefore used to attempt to recover the original sequence of uttered words and thereby correct some of the recognition errors. A simple expression for the corrected sentence, which

is the most likely pre-channel sequence of words $\hat{\underline{e}}_{1,n}$, is derived using Bayes' rule [7]:

$$\hat{\underline{e}}_{1,n} = \underset{\underline{e}_{1,n}}{argmax}\, P[\underline{e}_{1,n}]\ .\ P[\underline{e}'_{1,n'}\,|\,\underline{e}_{1,n}] \tag{3.1}$$

In equation 3.1, $P[\underline{e}_{1,n}]$ models the formation of English utterances by the speaker (the language model), while $P[\underline{e}'_{1,n'}\,|\,\underline{e}_{1,n}]$ models the behaviour of the channel [7].

For the implementation of the post-processor, hand transcriptions of a dialogue system were used to train a language model. Data for training the channel model was obtained by aligning the output of the SPHINX-II speech recognition system with the hand transcriptions and tabulating substitutions. The post-processor was designed to use the one-for-one channel model and the back-off bi-gram language model. Testing the system shows that an increase in overall speech recognition accuracy is possible, even after tuning the speech recognition engine [7].

Jeong *et al.* [16] proposed a system that applies a maximum entropy (ME) language model to re-score the N-best hypotheses returned from a first level correction using the noisy channel model. The ME language model has the advantage of allowing the incorporation of higher level linguistic knowledge, thereby making the error correction process more sensitive to syntactic and semantic level errors. In experimental results on the domain of in-vehicle telematics information retrieval (IR), they reported a 39% WER for a baseline ASR compared to 27% WER with noisy channel error correction, and 22.6% after re-scoring with maximum entropy language model (MELM), which represented an overall reduction of 42%.

Later, Jeong and Lee [13] developed a more complex method tailored to spoken language systems. The N-best hypotheses from the ASR module are corrected and re-generated using the noisy channel paradigm described above. Using a statistical domain-specific spoken language understanding model, the generated hypotheses are analyzed for semantic content. Finally, the hypotheses are re-ranked and an algorithm selects the best sentence based on the correction score and the semantic score. Testing the system in air travel and telebanking domains, reductions of up to 9.7% and 16.8% of word error rate were obtained, respectively.

University of Cape Town

3.2.5 Semantic Oriented Error Correction

Jeong *et al.* [14] developed a technique for correcting speech used as a query to an information retrieval (IR) system. The output sentence from the ASR engine is first converted into a lexico-semantic pattern (LSP), which is essentially a sequence of symbolic words used to represent the meaning of the sentence, with certain words abstracted to their respective classes. If the LSP does not form a correct statement due to speech recognition errors, it is corrected with a semantic confusion table and a template database. Next, a lexical correction is made by aligning the corrected LSP with the original recognized sentence, and using a lexical confusion table, domain dictionary and ontology dictionary to obtain the best replacement words from the replacement semantic classes, based on the minimum edit distance.

As an example, if the user spoke "take a train from Chicago to Toledo" but the output from the recognizer is "ticket train from Chicago to to leave", the conversion to LSP will be "<u>%ticket</u>@vehicle%from%location%to<u>%to%leave</u>". The LSP will be corrected to "<u>%take</u>@vehicle%from%location%to<u>%location</u>" from which the corrected sentence "take train from Chicago to Toledo" will be derived, as described above [14].

Experiments conducted to evaluate the performance of this system have shown a 12.97% word error rate, compared to 20.49% for the baseline speech recognition system. They also compared the performance with an error correction system using the noisy channel model, which had an overall word error rate of 14.21%, thus showing that this system provided slightly better results [14].

3.2.6 Correction with Common Sense Knowledge

The Open Mind Common Sense (OMCS) project, which started at the Massachusetts Institute of Technology (MIT) media laboratory, has been collecting data from the public about facts from everyday life, through a website. Over 700,000 statements have been collected from untrained Internet volunteers, since 2000. This data has been used to create ConceptNet, a semantic network containing more than 150,000 nodes of related concepts [30]. Lieberman *et al.* [4] have used ConceptNet to re-rank

University of Cape Town

the hypotheses returned by a speech recognizer, to filter out nonsensical phrases, and place at the top of the list, the hypotheses that make more sense, according to the context. They have shown that this method could be used to reduce speech recognition errors and to improve error-correction interfaces.

They discuss how a speech recognizer may have trouble distinguishing between two phonetically identical words like “break” and “brake”. By taking context into account, as in the example “my bicycle has a squeaky brake”, the system knows from ConceptNet, that a bicycle has brakes, and can therefore make the correct choice. The approach also works with phonetically identical words, such as “earned” and “learned”. Many commercial dictation systems allow the user to correct a recognition error by choosing from a list of possible words. Using this method, the list of alternatives can be re-arranged, with the ones that make more sense appearing first. They established that having the most obvious words at the top can reduce dictation time by providing a better error-correction interface, since the user is less likely to have to read the entire list to find the correct word [4].

Compared to WordNet [31], a widely used and popular semantic resource in the computational linguistics community [36], which began at Princeton University, ConceptNet contains common sense knowledge from everyday life, as opposed to more formal and taxonomic knowledge in WordNet. For example, in WordNet, a dog is identified as a type of canine, whereas in ConceptNet, a dog is a type of pet [4]. This makes ConceptNet a more suitable source of knowledge for general domains, although its use was envisaged to be broader than speech applications, as demonstrated by Edward *et al.* [32].

University of Cape Town

3.3 Other Approaches

There has been ongoing research on new or existing methods to improve speech recognition that have not been mentioned in this chapter, including Bayesian network structures, which can model several characteristics of natural speech in realistic environments in a unified way, such as ambient noise, microphone and channel variability, age, gender and pronunciation differences. Some existing techniques address

these problems fairly directly, as in vocal-tract length normalization (VTLN) [44], an attempt to compensate for variability in vocal-tract length, or parallel model combination (PMC), a way to separate out the effects of ambient noise by building separate noise and speech models. Maximum likelihood linear regression (MLLR), feature-space MLLR (FMLLR) and discriminant transforms such as LDA, are examples of techniques that can be used to address a wide and undefined range of phenomena simultaneously [26].

Stolcke [38] and a team of researchers from SRI International at Menlo Park, California, the International Computer Science Institute (ICSI) and the University of Washington, developed a system that produces "rich transcripts", that is, transcripts from conversational telephone speech and broadcast news, with additional structural information corresponding to sentence boundaries and disfluencies. They have succeeded in obtaining improved speech recognition accuracy, by working on all components of speech recognition and using or adapting some of the methods listed in the preceding paragraph. In the front end, they experimented with non-standard features including discriminative phone posterior features estimated by multilayer perceptrons, various measures of voicing, and an innovative phone-level macro-averaging for cepstral normalization. Improvements in acoustic modeling and language modeling were also achieved.

University of Cape Town

Chapter 4

Implementation of the Sphinx-4 Speech Recognition System

University of Cape Town

As it was described in chapter 1, we developed a software agent that post processes the result of any speech recognition system. In our prototype we have chosen to integrate the software agent with the Sphinx-4 speech recognition engine. This chapter briefly describes Sphinx-4 and provides details on the specific configurations used on our experimental systems.

4.1 Introduction to Sphinx-4

Sphinx-4 was developed after version 3 of the Sphinx speech recognition system, created at Carnegie Melon University (CMU). Unlike Sphinx-3 which was developed in C, Sphinx-4 was written entirely in the Java^{TM} programming language, and was designed to be an improvement over the previous version. Sphinx-4 was a joint collaboration between the Sphinx group at CMU, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT) [47].

The architecture of Sphinx-4 can be seen in figure 4.1 on the next page. When the recognizer starts up it creates the three main components: the FrontEnd, which performs the digital signal processing on the audio input and produces the resulting

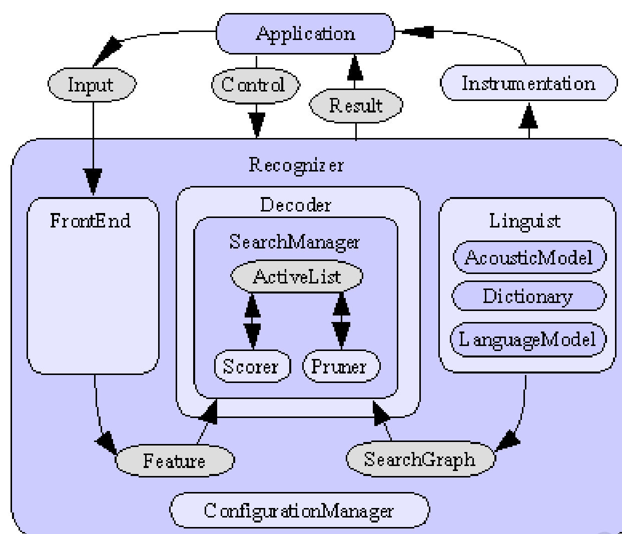


Figure 4.1: Architecture of the Sphinx-4 speech recognition engine [47]

University of Cape Town

features; the Decoder, which searches for the most likely sequence of words based on the features and the search graph; and the Linguist, which generates the search graph. These components, will in turn construct their own sub-components. For example, the Linguist creates the AcousticModel, Dictionary and LanguageModel sub-components. The Application controls the recognizer, provides the audio input, and receives the recognition Result and optionally the Instrumentation. The Instrumentation is additional information that can be generated by the recognizer, and may include information such as warning and error messages, accuracy statistics or search space plots [47].

The components and sub-components are created according to the configuration specified by the user for a specific task. For instance, a large vocabulary task requires a different linguist than that required by a small vocabulary task. The Configuration Manager of the recognizer is responsible for reading the user configuration from an XML based configuration file [47].

There isn't one single configuration file for Sphinx-4. Each application has its own configuration file which is read at run time. The name of the configuration file is specified on the Java code of the application.

We have created three Sphinx-4 applications for this research. They all share the same dictionary, acoustic model and language model, so these will be discussed

independently, followed by a more detailed description of the configuration files and how they differ for the individual applications.

4.2 Installing Sphinx-4

Instructions for downloading and installing Sphinx-4 can be found on the Sphinx-4 website [47], or by opening the "index.html" file under the Sphinx4 folder on the accompanying CD. This file is part of the Sphinx-4 distribution which was released as open source. We have used the source package of the beta 1.0 release of Sphinx-4. The entire source code for Sphinx-4 is thus available on the accompanying CD.

To build and run Sphinx-4 we used version 1.4 of the Java 2 SDK, standard edition, and Ant 1.6. Ant uses the "build.xml" file to build Sphinx-4 with the Java SDK. It also builds the Java archive (JAR) files for the acoustic models and reads the "demo.xml" file to build the demonstration programs.

University of Cape Town

4.3 The Acoustic Model and Dictionary

For our three experimental applications we decided to use the Wall Street Journal (WSJ) acoustic model and dictionary, contained in the "WSJ_8gau_13dCep_16k_-40mel_130Hz_6800Hz.jar" file under the "Sphinx4/lib" folder. The WSJ model was created from an American English speech corpus, and is distributed with Sphinx-4. It was decided that it was an ideal acoustic model due to its quality. In addition, the fact that it was created with an American accent, would allow us to test the performance of our system with different (e.g. South African) pronunciations.

Sphinx4 can use acoustic models created with SphinxTrain, which is a tool developed at CMU for this purpose. To train an acoustic model, SphinxTrain needs to be provided with the recorded speech files, a transcript containing the text corresponding to each recorded speech file, a dictionary mapping every word in the training corpus to the corresponding phonemes, a filler dictionary, and a list of phonemes.

Training an acoustic model essentially consists of adjusting the model parameters, given the observation, i.e. the recorded speech and corresponding transcripts.

This is one of the hardest problems of HMMs [23]. SphinxTrain adopts the Baum-Welch algorithm for estimating the model parameters. It then generates a number of model files which should be used to build a JAR file for the acoustic model. For example, we have created an acoustic model to experiment with SphinxTrain, and used the following files:

```
cd_continuous_8gau/means
cd_continuous_8gau/mixture_weights
cd_continuous_8gau/variances
cd_continuous_8gau/transition_matrices
dict/genese.dic
dict/fillerdict
etc/genese_13dCep_16k_40mel_130Hz_6800Hz.1000.mdef
etc/genese_13dCep_16k_40mel_130Hz_6800Hz.ci.mdef
```

All of the above folders and files should be located under one directory, which is specified in the build.xml file, so that the JAR file can be built with the directory contents. In addition, a file called "model.props" should be created, containing information about the acoustic model. For the WSJ model this file is located under "sphinx4/models/acoustic/wsj". Some of the most important technical characteristics are shown below:

```
numberFftPoints = 512
numberFilters = 40
gaussians = 8
minimumFrequency = 130
maximumFrequency = 6800
sampleRate = 16000
```

4.4 The Language Model

For our task, we used a statistical trigram language model, created with the Carnegie Melon University Statistical Language Modeling Toolkit (CMU SLM toolkit). At

University of Cape Town

the time of this writing, the toolkit was available at *"http://www.speech.cs.cmu-.edu/tools/lmtool.html"*.

To create the language model, the toolkit needs to be supplied with a training transcription. The transcription is essentially a list of likely sentences. Not all valid combination of sentences need to be provided, however, all dictionary words that are to be used in recognition should be contained in the transcription. Since the Sphinx-4 Linguist uses the language model to build the search graph, if a word is not part of the language model, it won't be recognized, even if it exists in the dictionary. We took advantage of this fact and used the "cmudict.0.6d" dictionary in the WSJ model JAR file, which contains many words that are not used in our experiment. This gave us the flexibility of adding new words already in the dictionary, by simply updating the language model.

In order to create a good training transcription, we developed a Java program that automatically generated over 5000 sentences from a common sense knowledge base. This training transcription can be found on the accompanying CD. Since the knowledge base is central to functioning of the software agent, we will defer the discussion of this program to Chapter 5. The language models created are located on the same directory as the Java class files of their respective applications. They are also included in the JAR file of these applications. A sample of the language model for the SmartSpeech application can be seen in Appendix A.

University of Cape Town

4.5 Configuration of the Sphinx-4 Applications

As previously mentioned, sphinx-4 uses an XML based configuration file, which allows the application programmer to configure sphinx-4 for a particular task. In our case we have created three applications and they each have their own configuration file. These files, which have a ".xml" extension, are located in the same folder as their applications, and are included in the JAR files of their respective applications.

The "PlainSpeech" application is a live dictation program that converts a spoken utterance into text and writes each line to a text file. The "SmartSpeech" application does exactly the same, except that the output of the Sphinx-4 recognition engine

is first processed for syntax and semantic errors, optionally corrected, and only valid sentences are written to the text file. Since the speech recognition process for these two applications are the same, their configuration files are identical. The third application, called "WavToText" is a sound file transcriber. It transcribes the speech recorded in an audio file. The different configuration required will be shown here. The applications are described in more detail in Chapter 6.

Looking inside the "smartspeech.config.xml" configuration file for the Smart-Speech application found in Appendix B, which as previously explained, is identical to the configuration for the PlainSpeech application, we find the following section:

```
<!-- ****************************************************** -->
<!--               The Dictionary configuration             -->
<!-- ****************************************************** -->
<component name="dictionary" type="edu.cmu.sphinx.linguist.
dictionary.FastDictionary">
  <property name="dictionaryPath" value="resource:/edu.cmu.sphinx.
  model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model!/
  edu/cmu/sphinx/model/acoustic/WSJ_8gau_13dCep_16k_40mel_130Hz_
  6800Hz/dict/cmudict.0.6d"/>
  <property name="fillerPath" value="resource:/edu.cmu.sphinx.model.
  acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model!/edu/cmu/
  sphinx/model/acoustic/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/
  dict/fillerdict"/>
  <property name="addSilEndingPronunciation" value="false"/>
  <property name="wordReplacement" value="&lt;sil&gt;"/>
  <property name="unitManager" value="unitManager"/>
</component>
```

University of Cape Town

The section above is the dictionary configuration, and it can be seen that the location of the dictionary is indicated here. The notation used points to a path inside the JAR file of the acoustic model. First, the path to the acoustic model is given by the string that precedes the "!" character. The string after the "!" is the path inside the actual model JAR file.

The following section of the XML file configures the language model. It should be noted that the "trigramModel" component was specified in the linguist configuration section. Using the same notation previously explained, the "location" property specifies the path to the language model file inside the JAR file.

```
<!-- ****************************************************** -->
<!--            The Language Model configuration            -->
<!-- ****************************************************** -->
<component name="trigramModel" type="edu.cmu.sphinx.linguist.
language.ngram.SimpleNGramModel">
  <property name="location" value="resource:/demo.sphinx.
  smartspeech.SmartSpeech!/demo/sphinx/smartspeech/smartspeech.lm"/>
  <property name="logMath" value="logMath"/>
  <property name="dictionary" value="dictionary"/>
  <property name="maxDepth" value="3"/>
  <property name="unigramWeight" value=".7"/>
</component>
```

Looking at the configuration file for the SmartSpeech and PlainSpeech applications, we find the following section, which corresponds to the configuration of the front end.

```
<!-- ******************************************************** -->
<!--        The live frontend configuration                   -->
<!-- ******************************************************** -->
<component name="epFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
 <propertylist name="pipeline">
   <item>microphone </item>
   <item>speechClassifier </item>
   <item>speechMarker </item>
   <item>nonSpeechDataFilter </item>
   <item>premphasizer </item>
   <item>windower </item>
   <item>fft </item>
```

University of Cape Town

```
    <item>melFilterBank </item>
    <item>dct </item>
    <item>liveCMN </item>
    <item>featureExtraction </item>
  </propertylist>
</component>
```

The above portion of XML lists the front end components (the front end pipeline) that are given specific attributes further down on the configuration file. The WavToText application also contains a front end configuration, but instead of the "<item> microphone </item>" tag we have the "<item>streamDataSource </item>" tag. This is because the speech data for the WavToText application doesn't come from a microphone, but from a recorded file which is read as a data stream by the "streamDataSource" component. This component is configured as follows.

```
<component name="streamDataSource"
type="edu.cmu.sphinx.frontend.util.StreamDataSource">
  <property name="sampleRate" value="16000"/>
  <property name="bitsPerSample" value="16"/>
  <property name="bigEndianData" value="false"/>
  <property name="signedData" value="true"/>
  <property name="bytesPerRead" value="320"/>
</component>
```

It can be seen above that the "streamDataSource" component properties such as the sampling rate are specified in the configuration file.

University of Cape Town

Chapter 5

The Software Agent

University of Cape Town

The software agent shown in figure 1.1 on page 5 is at the heart of the proposed solution for improving speech recognition accuracy. In this chapter we describe in detail the prototype of the software agent developed during the course of this research, using the Unified Modeling Language (UML) where appropriate. This software agent is used by the SmartSpeech and WavToText speech recognition applications based on Sphinx-4, which were briefly described in the previous chapter. The "Genese" applications, developed to test the behaviour of the software agent are also discussed here.

5.1 Overview

The software agent is in essence an intelligent text processor. The goal of the software agent is to analyze a line of text to identify possible syntax or semantic errors and correct them, if possible. In artificial intelligence (AI) terms, the world of the agent is made up of all the words in the dictionary. The information about the agent's world contained in the knowledge base, provides the agent with the data needed to achieve its goal.

A syntax error means a violation of language rules that would make a sentence ungrammatical. For example, the phrase "dog eat win runs the" is not a valid English sentence. There are grammatical rules that govern the sequence in which nouns, verbs, adjectives, conjunctions and other types of words can be used together

to construct valid sentences. Checking the syntax of a sentence, can therefore be done by evaluating the sequence of words and determining if the different word types are in a correct order. This is the approach used by our software agent.

A sentence can be syntactically correct but completely meaningless. For example, the sentence “the wall runs the ear” is grammatically correct, but doesn’t have any useful meaning. For this reason, the software agent also checks for semantic errors. In order words, it determines whether or not the meaning of the sentence agrees with the human knowledge of the world. To do this, the software agent checks the sentence against a knowledge base. The software agent also uses the knowledge base when attempting to correct a syntax or semantic error.

5.2 Scope and Limitations

The software applications developed as a result of our research were intended to conduct experiments in order to further investigate the concept that common sense knowledge can be effectively used to increase speech recognition accuracy. However, several limitations exist on the software prototypes developed, that limit their immediate use in many practical applications.

The system was tested with a vocabulary of 180 words. This isn’t an actual limitation of the software agent. We could have used a large vocabulary without making any changes to the software. However, every new word in the dictionary needs to be added to the knowledge base as well. Creating new entries in the knowledge base is a time consuming task, since the human knowledge associated with the new word and the existing words, have to be entered for the software agent to function correctly. Since it is possible to create over five million different three-word sentences with 180 words alone, it was decided that the vocabulary size was sufficient for the purposes of this research.

The syntax or grammar of the sentences that the system is allowed to recognize correctly is limited to two types of sentences, containing specific sequences based on the word type. The DNV sentence, as it shall be called in this document, consists of a determinant (such as a definite, or indefinite article, for example ’the’ or ’a’),

University of Cape Town

followed by a singular noun, followed by a verb in the present or past tense. Examples of DNV sentences are 'the cat eats' and 'the phone rang'. DNVDN sentences, as used in our system, contain the following sequence: determinant - singular noun - present or past tense verb - determinant - singular noun. The following are examples of DNVDN sentences: 'a boy ate the apple', 'the girl opened the present' and 'the music filled the hall'. If a sentence does not fall into one of the above categories of valid sequences, it will be deemed to have a syntax error by the software agent, even if it is a valid English sentence. The system was therefore, only tested with DNV and DNVDN sentences. Once again, the limited time and resources available for creating a more complex knowledge base and the associated algorithms, were behind the reasons for this limitation.

The software agent can detect and correct substitution and deletion speech recognition errors. The current version was not designed to correct insertion errors. It should be noted that the "corrected sentence" is not guaranteed to be the actual spoken utterance. The goal of the agent is to make an intelligent guess, based on the available knowledge, in a similar way that a human would.

University of Cape Town

5.3 Use Case Model

In UML, a use case model describes the behaviour of a system from the point of view of the user. One or more related scenarios of interaction between the user and the system are grouped into a use case. A use case model is composed of a use case diagram containing one or more actors, one or more use cases, and the associated use case text.

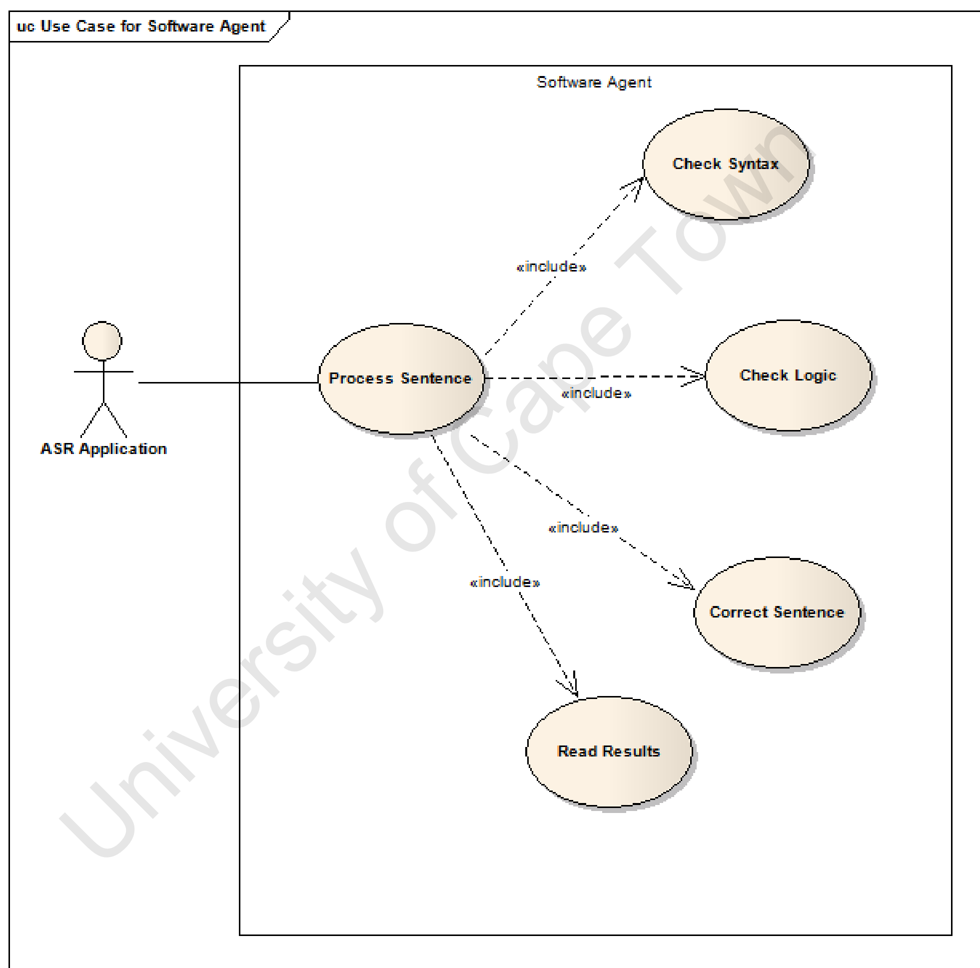
The use case text may contain a number of word repetitions, which is done on purpose, to avoid ambiguity. Following the principles of the ICONIX methodology, the use case text was written in the active voice, for the same reason.

Since the software agent is a software component not directly accessible by the user, the actor in our model is another computer program, more specifically, an automatic speech recognition application. The UML 'include' stereotype indicates that smaller use cases, such as 'check syntax' are part of, or 'included' in the larger

use case 'process sentence'.

The use case diagram and associated use case text for the software agent are shown in the following subsections. It should be noted again that the purpose of the use case model is to describe the 'what' not the 'how'. Therefore, only the essential details of the interaction between the actor and the system are made explicit.

5.3.1 Use Case Diagram



University of Cape Town

Use Case Diagram for Software Agent

5.3.2 Process Sentence

The ASR Application passes a String array containing the sequence of words to be processed, as an argument to the SpeechSentence.setOriginalSentence(String[]) method of the software agent. The ASR Application invokes the SpeechSentence.analy-

zeSentence() method of the software agent. This use case includes the Check Syntax, Check Logic, Correct Sentence and Read Results use cases.

5.3.3 Check Syntax

The Software Agent checks the syntax of the original_sentence. If the sequence of words is invalid, the Software Agent sets the syntax_error_detected flag to true. Otherwise, it sets the flag to false.

5.3.4 Check Logic

The Software Agent checks the logic of the original_sentence. If the meaning of the sentence is invalid, according to the knowledge base of the Software Agent, the Software Agent sets the logic_error_detected flag to true. Otherwise, it sets the flag to false.

5.3.5 Correct Sentence

If the Software Agent finds that both the syntax and logic of the original_sentence are valid, it sets the sentence_OK flag to true.

If the Software Agent finds a syntax and/or a logic error in the original_sentence, it sets the sentence_OK flag to false and attempts to correct the errors with the information on its common sense knowledge base. If it finds a unique solution, the Software Agent stores the corrected sentence on the SpeechSentence.suggested_sentence field as a String array.

5.3.6 Read Results

The ASR Application reads the SpeechSentence.sentence_OK flag to determine if the software Agent found any errors with the original_sentence. If the flag is set to true, the ASR Application uses the original_sentence as the final speech recognition result.

If the sentence_OK flag is set to false, indicating that the Software Agent found an error with the original_sentence, the ASR Application may access the

University of Cape Town

syntax_error_detected and logic_error_detected flags to inform the user of the nature of the error. The ASR Application checks the suggested_sentence_exists flag. If it is set to true, the ASR Application uses the suggested_sentence as the final speech recognition result. If the suggested_sentence_exists flag is set to false, the ASR Application takes an appropriate action, for example, requesting the user to repeat the sentence, in the case of an interactive application.

5.4 The Genese Package

All the classes that make up the software agent were placed in a Java package called Genese. The UML class diagram for the intelligent agent in the Genese package is shown in figure 5.1 on page 40. This provides a useful overview of the architecture of the software agent before delving into the details of its inner operations. The IntelligentAgent class is the core component of the software agent, containing all the methods that analyze and correct sentences. The DeletionErrorHandler is responsible solely for handling deletion errors. It will be implemented as a nested class in future versions of the software and is therefore shown as a member of the IntelligentAgent class. In the current release, the source code for the IntelligentAgent and the DeletionErrorHandler classes are both contained in the IntelligentAgent.Java file.

University of Cape Town

Knowing in advance that the software agent is supposed to process only DNV and DNVDN sentences, we developed algorithms designed to attempt correction from all recoverable deletion errors. We have grouped the deletion errors into three categories, each handled by a different method in the DeletionErrorHandler class.

The TextUtilities class is another helper class, containing general methods for manipulating text, which are frequently used by the software agent. For example, a single line of text containing a sentence to be processed, can be passed to the software agent as a String. In this case, the software agent uses the TextUtilities.splitSentence(String) method to create a String array containing each word in the sentence, ignoring spaces. This allows the software agent to identify and process words in the sentence.

The SpeechSentence class is the interface of the software agent with applications

that make use of its intelligent text processing capability. It contains fields that hold the sentence to be analyzed, the suggested (correct) sentence, as well as flags that indicate the result of the processing. The SpeechSentence.analyzeSentence() method creates an instance of the IntelligentAgent class and invokes its processSentence(SpeechSentence) method.

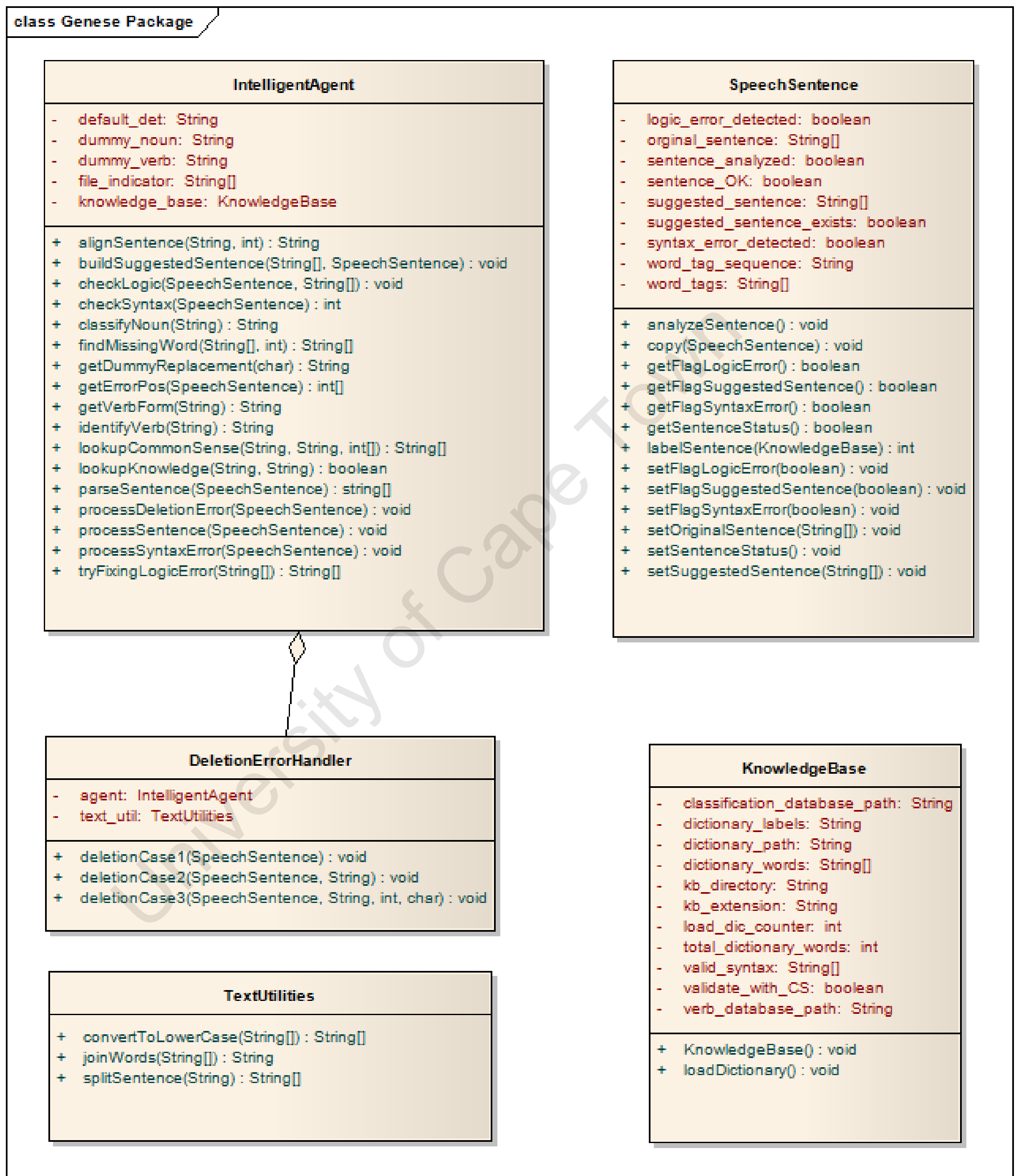
The software agent checks for semantic errors and corrects sentences, using the information contained in a KnowledgeBase object created with the KnowledgeBase class. The fields in the KnowledgeBase class hold elements of the knowledge base, such as all the words in the dictionary and the valid syntax, which is DNV and DNVDN in our current prototype. The fields also contain the path to the knowledge base files. The constructor in the KnowledgeBase automatically detects the operating system, and initializes the correct path to the knowledge base files, depending on whether the software is running on the Microsoft Windows or Linux platform. It also makes sure that the dictionary is loaded only once, via the loadDictionary() method.

5.5 Knowledge Base

The knowledge Base (KB) used by the IntelligentAgent class was designed specifically for processing DNV and DNVDN sentences. However, it could be used with a slightly wider syntax, or expanded to accommodate a more complex grammar. The Knowledge Base serves two purposes. One is to validate the meaning of the sentences. The other, is to correct a syntax or semantic error, by suggesting another sentence. The Knowledge Base therefore integrates a Validation Knowledge Base (VKB) and a Common Sense Knowledge Base (CSKB). An important element in the design of the Knowledge Base, was the use of abstraction and a format to represent meaning, to avoid repetition. This was done by using a classification database and a verb database.

Table 5.1 on page 41 lists all the files that make up the knowledge base. It can be seen that the VKB and the CSKB share an identical structure. The KB files are text files, which were easy to edit and fast enough for the vocabulary size used in our

University of Cape Town



University of Cape Town

Figure 5.1: Class Diagram for the Genese Package

File Name	KB Group	Example of Content
classification_database.txt	N/A	cat ANIMALS
verb_database.txt	N/A	answered ANSWER
SPEV.txt	Validation	BEARD BURN
GENV.txt	Validation	ANIMALS EAT
GENVGEN.txt	Validation	ANIMALS ANSWER ANIMALS
GENVSPE.txt	Validation	ANIMALS PAY BILL
SPEVGEN.txt	Validation	CAR DRIVE ANIMALS
SPEVSPE.txt	Validation	CLOCK SHOW TIME
CS_SPEV.txt	Common Sense	ANIMALS EAT
CS_GENV.txt	Common Sense	BEARD GROW
CS_GENVGEN.txt	Common Sense	PEOPLE EAT FRUIT
CS_GENVSPE.txt	Common Sense	ANIMALS DRINK WATER
CS_SPEVGEN.txt	Common Sense	HEATER WARM OBJECTS
CS_SPEVSPE.txt	Common Sense	COW EAT GRASS

Table 5.1: Knowledge Base Files

University of Cape Town

experiments, even with simple search algorithms. However, if a large vocabulary was to be used, a relational database management system (RDBMS), or another system with better performance would be preferable. The following subsections describe the Knowledge Base components in more detail.

5.5.1 Meaning Representation

When designing the knowledge base, we wanted to develop a simple method of representing meaning, in order to create the agent's knowledge of the world in a way that is independent of grammar or sentence structure. At the same time, we wanted each knowledge base statement to read in a way that coincides with the human knowledge of the world, since the KB had to be created manually. The result can be seen in table 5.1, which provides examples of various KB statements using the meaning representation adopted.

Let us take, for example, the statement "BEARD GROW". This statement means, quite obviously, that a beard can grow. In our implementation, this can be used to validate that the following sentences are semantically correct: "the beard grows", or "this beard grew". The statement "ANIMALS EAT" can be used to validate the sentences "the dog ate", or "the cow eats" as correct. The meaning of

the statement is, the first word, indicated by an object, which represents one or more nouns, can be associated with the action indicated by the second word, which represents a verb.

5.5.2 Dictionary

As it was mentioned in Section 5.1, the agent's world consists of all the words in the dictionary. The dictionary file in the knowledge base, provides the agent with the first level of information on its world, by mapping each word to a letter, which represents a noun (N), verb (V) or a determinant (D). Without this information, the software agent is unable to perform any further processing. The word type is usually not present in ASR systems based on statistical language models, which means that this categorization had to be entered in the KB dictionary manually. The syntax of the KB dictionary is a word followed by a single space, followed by the letter N, V or D. Each word must be placed in a separate line. It should be noted that this dictionary is not the same dictionary used by the ASR engine, which maps words to phonemes.

University of Cape Town

5.5.3 Classification Database

It has been explained in 5.5.1 that in our meaning representation syntax, one word or symbol can be used to represent more than one word or noun. This is accomplished by using the classification database. For example, instead of entering the following statements in the knowledge base: "CAT EAT", "COW EAT", "DOG EAT", we can create a generalization and say that the word "ANIMALS" represents "CAT", "COW" and "DOG". To do that we would enter the following lines on the classification database:

```
cat ANIMALS
cow ANIMALS
dog ANIMALS
```

Then we only need to enter a single statement in the knowledge base: "ANIMALS EAT". Any word could of course be used, but since the knowledge base has to be

created manually, it makes sense to choose a symbol that allows us to structure the knowledge base in a similar way that humans structure their knowledge about the world.

It is important to note that not every word in the knowledge base dictionary needs to be entered in the classification database. There may be cases when there are no several words in the dictionary with an equivalent meaning or classification, in which instance, they would only be used on their own.

5.5.4 Verb Database

The verb database serves a purpose similar to that of the classification database. It allows a single word or symbol to represent an action indicated by one or more verbs in various forms. In the current implementation of the knowledge base, the verb database was used to map a symbol to a verb in the present and past tense. For example, the word "EAT" is used to represent the present and past tense of the verb "to eat" as follows:

```
ate EAT
eats EAT
```

Once again, this information is usually not available in a statistical ASR system. As far as the speech recognition engine is concerned, the word "ate" and the word "eats" are two unrelated symbols, each with their own assigned probability of occurrence. The verb database also had to be created manually. Whereas entering a noun in the classification database is optional, in contrast, every verb must be present in the verb database.

5.5.5 Validation Knowledge Base

The Validation Knowledge Base, serves to check that sentences have a valid meaning. It therefore should contain all the human knowledge associated with the agent's world. In other words, it should contain all the valid meanings associated with the words in the knowledge base dictionary. This does not mean that the VKB stores every valid combination of words. By using the meaning representation described

University of Cape Town

in 5.5.1 with abstraction, the number of knowledge base statements is drastically reduced.

Looking at Table 5.1 on page 41 it can be seen that the validation knowledge base is organized into six files. Each file name gives an indication of the type of knowledge base statements that the file contains. The "GEN" stands for generalization, and means that the symbol in that position represents several words, as indicated by the classification database. The "SPE" stands for specific, and means that the symbol in that position represents a single, actual word in the dictionary. The "V" stands for verb, and represents words that relate to a single verb, as indicated by the verb database. GENV and SPEV statements are used to validate DNV sentences, whereas all the other statement types are used to validate DNVDN sentences.

An example of a GENVSPE statement is "ANIMALS DRINK WATER". In this statement, the word "ANIMALS" is a symbol from the classification database, as seen in 5.5.3. The word "DRINK" is a symbol from the verb database that represents the words "drinks" and "drank". Finally, "WATER" is a "SPE" word, which means, it is the actual word "water" from the dictionary. This statement could therefore be used to validate the meaning of sentences such as the following: "the dog drinks the water", "that cow drank this water" or "this cat drank the water".

As it can be seen from the sample statements in Table 5.1 on page 41, the different variations of knowledge base statement types, allow for greater flexibility in representing meaning. The SPEVSPE file is used for those cases where generalization could not be used.

Since the validation knowledge base is use to check the output of the speech recognition engine for semantic errors, it should accommodate a wide range of English sentences. The human world knowledge it contains, may sometimes contradict our common sense. If the KB statement is possible, even if unlikely, then it is not incorrect to include it in the VKB. For example, the statement seen in Table 5.1 on page 41, "ANIMALS PAY BILL" would validate a sentence like "the cat paid the bill". While uncommon, this sentence contains a valid meaning. It could be used, for instance, to refer to an action of a cartoon character, or even figuratively. As it will be seen in Chapter 6, the knowledge engineer can tune the validation knowledge

University of Cape Town

base to suit a particular speech recognition task.

5.5.6 Common Sense Knowledge Base

The common sense knowledge base is a subset of the validation knowledge base. As the name implies, it contains only those statements in the validation knowledge base that are common sense. For example, typically the CSKB would not include the statement “ANIMALS PAY BILL” discussed in 5.5.5. However, a statement such as “ANIMALS DRINK WATER” would normally be included in the common sense knowledge base.

When the software agent needs to correct a semantic or syntax error, including a deletion error, it attempts to make an intelligent guess based on the common sense knowledge base. For this reason, the CSKB should only contain statements that are obvious, or that are aligned to our particular speech recognition task.

With reference to Table 5.1 on page 41, the CS_GENV.txt and CS_SPEV.txt files, which contain GENV and SPEV common sense statements respectively, are used to correct DNV sentences, whereas all the other CSKB files are used to correct DNVDN sentences.

The choice of common sense knowledge base statements should be made by taking into account the algorithm used to find the correct word, which is described in more detail in 5.6.7. The agent looks for a “match” in the CSKB files. If more than one match is found in the same CSKB file, the agent discards any solution from that CSKB file, because there can only be one match, which will be used to create the “suggested sentence”. There needs to be a good balance between having enough CSKB statements to enable more sentences to be corrected, and not having too many statements in the same file, resulting in multiple solutions, which will lead to no corrections being made at all by the agent.

5.6 System Activity Diagram

In this section, we describe the entire operation of the software agent, which is the implementation of the use case model discussed in Section 5.3, using UML activity

University of Cape Town

diagrams. Because the diagrams are too large, they have been split into modules or behaviours, which do not necessarily map to methods in the source code. The name of the module appears in the beginning of the activity diagram, enclosed in "**". If a behaviour name appears anywhere else in the diagram, it means that a call, or transfer is made to that module from the current one. The activity diagrams shown here do not contain the same level of detail as the source code.

5.6.1 Initial Processing

When a line of text is passed to the software agent for analysis, the initial processing takes place as depicted in Figure 5.2 on page 47. The first step is to identify each word in the sentence as either a noun (N), verb (V) or a determinant (D). The agent performs word tagging using the dictionary described in 5.5.2, and creates a sentence label from the tag sequence. For example, if the *original_sentence* is "the girl loves the cow", the tag sequence or sentence label will be "DNVDN". If the *original_sentence* is "girl that dog loves" the sentence label or tag sequence would be "NDNV".

University of Cape Town

Next, the agent checks the tag sequence against the valid sequences stored in the KnowledgeBase.valid_syntax array, which in our current implementation are only "DNV" and "DNVDN". If only one syntax error is found, the software identifies the position of the incorrect word. If it's the position where a D is expected, the word is replaced by a default 'D', in our case the word 'the'. The agent then calls the 'Check and Fix Logic Error' module. If it is a N or a V, the agent calls the "Correct Sentence" module. The agent determines that there is only one syntax error if the sentence label is the same length as one of the valid tag sequences, and only one of the expected tags is wrong. For example, the sentence "dog girl laughs" has a NNV sentence label, which yields only one syntax error when compared to the DNV valid syntax. For all other cases we have more than one syntax error, and the agent calls the "Deletion Error" module.

If the agent finds no syntax error, that is, if the sentence label is valid, the agent calls the Check and Fix Logic Errors module, to check for semantic errors.

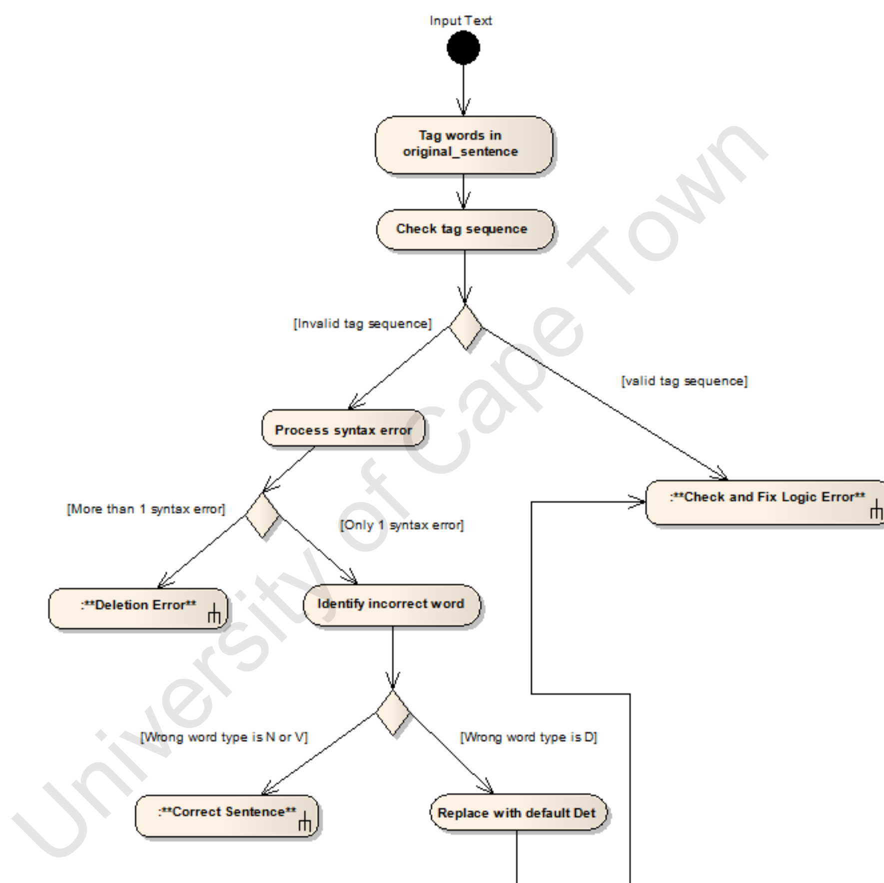


Figure 5.2: Activity Diagram for Initial Processing

University of Cape Town

5.6.2 Deletion Error

One the agent reaches the Deletion Error module, shown in Figure 5.3 on page 49, it doesn't necessarily mean that there was an actual speech recognition deletion error. In fact, it may be that there was an insertion error, or perhaps several substitution errors by the ASR engine. If there was an actual deletion error, there are 32 cases where a solution may be possible. This is based on the fact that we know the allowed syntax beforehand, and how much information is needed for the agent to be able to make an intelligent guess. The software agent has a list of all the tag sequences or sentence labels, that can be processed further. If a sentence label is not on the list, then the software agent cannot find a solution and sets the *syntax_ error_ detected* flag to true and *suggested_ sentence_ exists* to false.

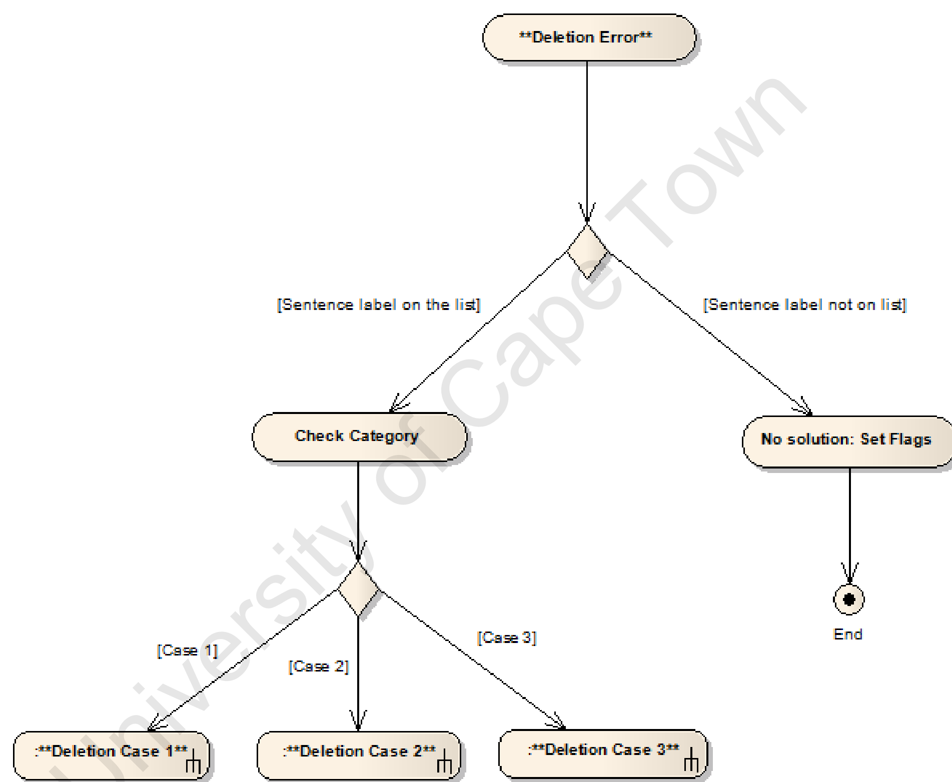
Each sentence on the list is categorized as either a Deletion Error 1, 2 or 3 and the agent calls the appropriate module according to the sentence label of the *original_ sentence*. This categorization was used, because a different algorithm was needed to handle each case.

University of Cape Town

5.6.3 Deletion Case 1

The first category of deletion error, which we call Deletion Case 1, shown in Figure 5.4 on page 50, is the simplest type of deletion error that the software agent can handle. This occurs when only determinants are missing from the sentence. There may be one or more determinants missing. Text with the following sentence labels can be corrected under this category: NV, NVN, NVDN and DNVN. Some examples are: “sun shines”, and “woman waters plant”.

Correcting this syntax error is simple because in linguistics, determinants are function words. Unlike content words, such as verbs and nouns, function words do not affect the core meaning of the sentence. Certainly the meaning of a sentence is affected depending, for example, on whether the determinant ’this’ or ’that’ is used, but the change in meaning is not as substantial as using the wrong noun or verb. Consequently, the agent makes a guess by merely using a default determinant, such as the word “the”. To correct the first sentence in our example, we would



University of Cape Town

Figure 5.3: Deletion Error Activity Diagram

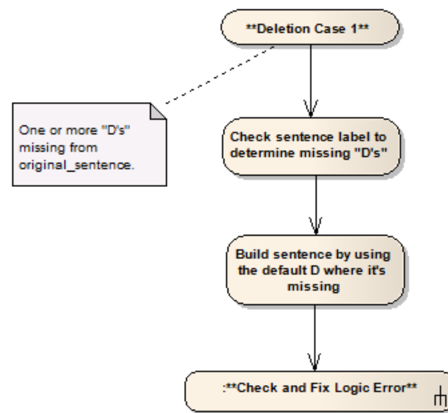


Figure 5.4: Deletion Case 1 Activity Diagram

build a sentence by filling in the missing determinant to produce “the sun shines”. There could still be a speech recognition substitution error, hence after correcting the syntax, the agent checks the sentence for semantic errors by calling the Check and Fix Logic Error module.

5.6.4 Deletion Case 2

In the Deletion Case 1, only function words, or more specifically, determinants were missing from the sentence. We call it a Deletion Case 2, when a single content word is missing from the sentence. This word could either be a noun or a verb. In addition, there could be one or more determinants missing. The activity diagram for this module can be seen in Figure 5.5 on page 52.

The list of sentence labels under this category was created taken into consideration that for a DNVDN sentence, at least two content words in the correct sequence are needed for the software agent to be able to make an intelligent guess. For a DNV sentence, the verb must be present. An example would be “the barks”. From the common sense knowledge base the agent would be able to guess that the missing noun is “dog”. There are probably few cases such as this one, but the software agent does have the algorithm for handling them. Whether the agent finds a solution or not, depends entirely on the data in the common sense knowledge base.

In order to preserve the *original_sentence*, its content is copied to a tempo-

University of Cape Town

rary array. The agent then fills in any missing determinants on the temporary array, and adds the word 'dummy_noun' in place of the missing noun, or the word 'dummy_verb' in place of the missing verb. The words dummy_noun and dummy_verb are placeholders for a noun or a verb respectively, and they are listed in the knowledge base dictionary for that purpose. This ensures that the syntax is correct for further processing. The position of the placeholder for the missing word is passed along with the temporary array to the Find Matching Word module. As an example, if the original_sentence was "woman waters the", the temporary array would have the sentence "the woman waters the dummy_noun".

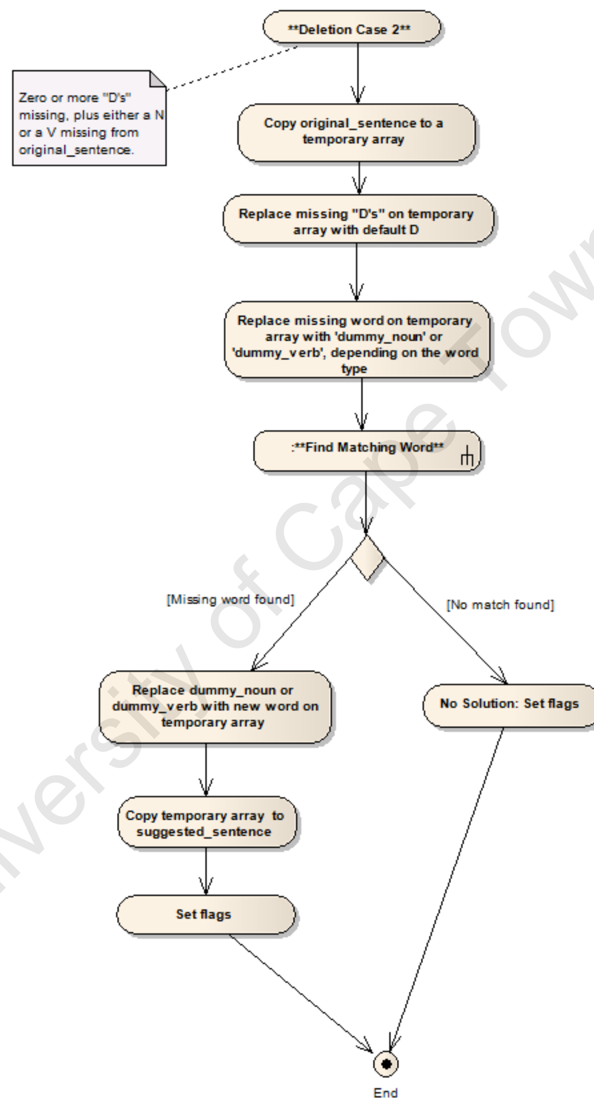
If the Find Matching Word module is able to find a matching word based on the common sense knowledge base, the agent replaces the dummy_noun or dummy_verb with the matching word and copies the array to suggested_sentence. The agent sets the *syntax_error_detected* flag to true, since a deletion error is a type of syntax error, and the *suggested_sentence_exists* to true. In our previous example, if the matching word was 'plant', the *suggested_sentence* would be "the woman waters the plant".

If no matching word could be found, the agent concludes that there is no solution and sets the *syntax_error_detected* flag to true, and the *suggested_sentence_exists* to false.

University of Cape Town

5.6.5 Deletion Case 3

It was discussed in 5.6.1 that one syntax error is present when the agent detects one word type that does not agree with the valid syntax, such as a noun or a determinant where a verb should be placed instead. If a syntax error such as this occurs and at the same time one or more determinants are missing, the error is classified as Deletion Case 3. The activity diagram is shown in Figure 5.6 on page 53. The actual deletion refers to the determinants, but the fact that a wrong word type is identified as well, makes it a special case. The following sentences are examples of this category: "man cat the apple", "dog man ate apple", or "man ate ate". The preceding examples are processed as errors on DNVDN sentences. An example of a deletion case 3 on a DNV sentence would be "eats barked".



University of Cape Town

Figure 5.5: Deletion Case 2 Activity Diagram

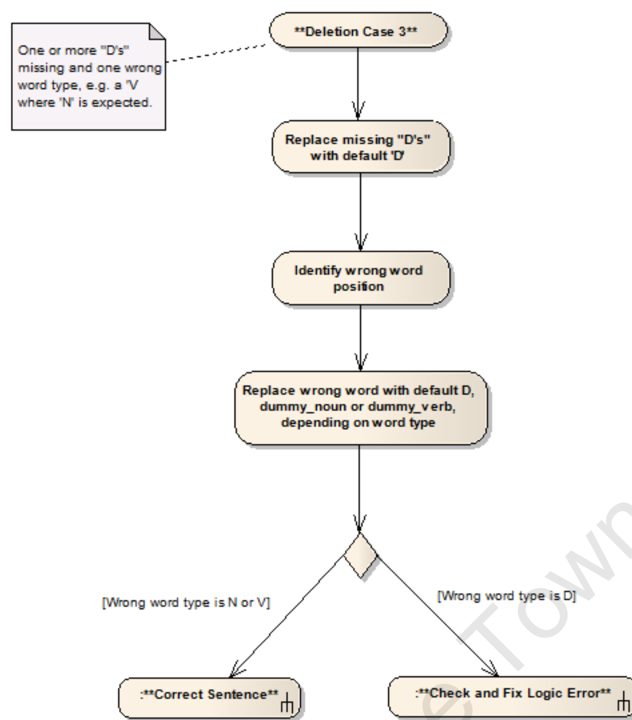


Figure 5.6: Deletion Case 3 Activity Diagram

University of Cape Town

Once the agent has filled in any missing determinants with the default, it identifies the position of the word that violates the syntax. If a noun or a verb is where a determinant should be, it is replaced by the default determinant as well. For example, "dog man ate apple" would become "the man ate the apple". The agent then calls the Check and Fix Logic Error module to ensure that the sentence just built is semantically correct.

If there is a noun where a verb should be, the agent replaces the noun with the word dummy_verb. Likewise, the agent replaces a verb in the place of a noun with dummy_noun. The agent then passes the sentence to the Correct Sentence module including the position of the placeholder, which is where the incorrect word was identified.

5.6.6 Correct Sentence

This module handles a syntax error, where the position of the error is already identified. The error position is passed along with the sentence to the Find Matching

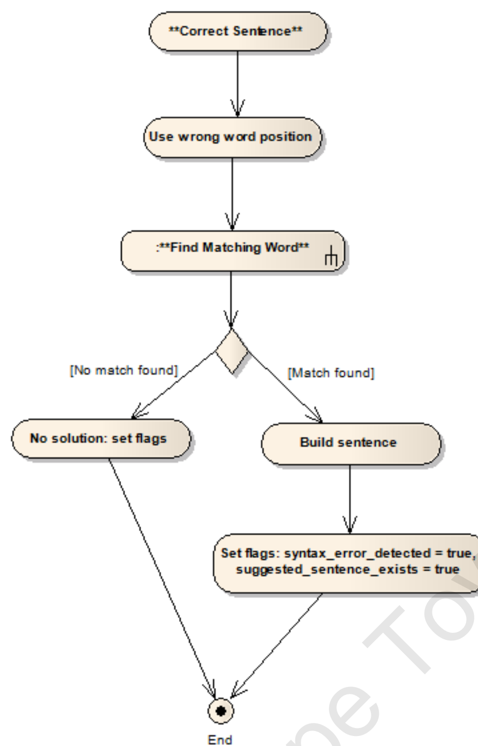


Figure 5.7: Correct Sentence Activity Diagram

University of Cape Town

Word module. A match means a word from the common sense knowledge base that can be used to replace the word that doesn't agree with the valid syntax. If this match is not found, the agent sets the appropriate flags. If is is found, the agent builds the suggested_sentence and sets the flags to indicate that an error was found but the agent was able to correct it.

5.6.7 Find Matching Word

This module is called to find a word in a specific position, which can be used to create a meaningful sentence. The process is summarized in the activity diagram shown in Figure 5.8 on page 57. The agent finds the solution from its common sense knowledge base, which has a syntax to represent meaning, as explained in 5.5.1. As a result, the agent first parses the sentence to extract and represent the meaning of the sentence in the same format. This is done by converting every verb into its representation specified in the verb database, and if a noun exists in the classification database its generic form is retrieved to form the GEN element of the

logic statement. For example, “eats” or “ate” would be converted to “EAT”, and television could be converted to OBJECTS to form the GEN element.

The terms “parsed sentences” and “logic statements” are used interchangeably to refer to the representation of the meaning extracted from the sentence. Considering the discussion in 5.5.5, it follows that one or two logic statement types, namely GENV and SPEV could be parsed from a DNV sentence, whereas up to four statement types, namely SPEVSPE, SPEVGEN, GENVSPE and GENVGEN could be generated from a DNVDN sentence.

All possible logic statements are thus created from the sentence, using ’X’ as a placeholder for the word the agent is trying to find. As an example, the sentence “the cat ate the dummy_noun”, would be represented by “CAT EAT X” and “ANIMALS EAT X”, since the word ’cat’ is classed as “ANIMALS” in the classification database. Each logic statement is used to perform a lookup in the common sense knowledge base file with a corresponding type. Since the agent needs to find an actual word, this lookup is done only for those CSKB types where the X matches with a SPE or a V element, which represent a specific word or a verb respectively. GEN elements represent a class of words, which would lead to multiple solutions. Consequently, considering the previous example, the agent would perform a lookup on the GENVSPE and the SPEVSPE types of common sense knowledge base files. However, it would exclude the GENVGEN and SPEVGEN CSKB files which could contain statements such as “ANIMALS EAT ANIMALS” or “CAT EAT CATFOOD”, which would produce matches that are generic.

When performing a lookup on a knowledge base file, the agent uses ’X’ instead of the actual element in the CSKB statement according to the error position, so that the string comparison can return a match. When a match is found, the word in the CSKB that is replaced by ’X’ is a potential solution. Continuing with our example, if the SPEVSPE CSKB file had the statements “CAT EAT RAT” and “COW EAT GRASS”, the agent would compare “CAT EAT X” with “CAT EAT X” and “COW EAT X”. “RAT” would then be a match, and therefore a potential solution.

If more than one match is found in a CSKB file, they are all discarded as potential solutions. In the previous example, if the SPEVSPE file had the statements “CAT

University of Cape Town

Common Sense Knowledge Base File	Assigned Rank
CS_GENV	0
CS_SPEV	1
CS_GENVGEN	0
CS_GENVSPE	1
CS_SPEVGEN	1
CS_SPEVSPE	2

Table 5.2: Ranking of Common Sense Knowledge Base Files

EAT RAT" and "CAT EAT FISH", both "RAT" and "FISH" would be matches in the same CSKB file, and thus discarded as potential solutions. If more than one CSKB file yields a potential solution, the agent compares the rank of the solutions. The highest ranking potential solution becomes the final answer. However, if there are several potential solutions with the same highest rank, they are discarded as well and the agent returns an empty string, which indicates that no solution was found. If there is only one match with the highest rank, the word is returned to build the suggested sentence.

The ranking of a match is done according to the rank of the CSKB file in which it is found. Each common sense knowledge base file is assigned a rank as seen in Table 5.2 on page 56. This rank is assigned based on the number of SPE elements in the type of knowledge base statements they contain. As a result, the software agent gives preference to more specific knowledge base statements.

When a match is found on a SPE element it corresponds to an actual word in the dictionary, and that word can subsequently be used to build the *suggested_sentence*. If a match is done on a 'V' element, which represents a verb, an actual word needs to retrieved from the verb database. In the current implementation, the agent retrieves the first entry, or verb form, that it finds. For example, if the match is 'CATCH' the actual word returned becomes 'catches', since it is the first entry in the verb database for 'CATCH'.

5.6.8 Check and Fix Logic Error

The Check and Fix Logic Error module checks if a sentence contains a semantic error and attempts to correct the error, if it finds one. This process is summarized

University of Cape Town

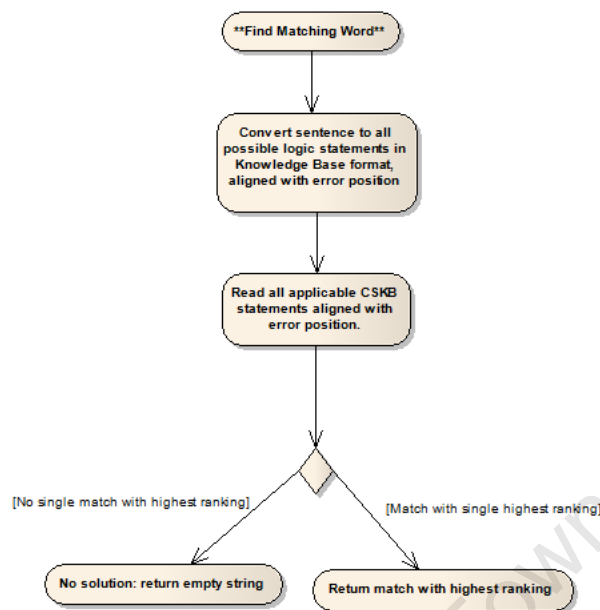


Figure 5.8: Find Matching Word

in Figure 5.9 on page 59.

To check for a semantic error, the sentence is parsed and converted into all possible logic statements, using the meaning representation syntax of the knowledge base. The agent performs a lookup of each statement in their corresponding validation knowledge base file. If at least one match is found, that is, if one of the statements exists in one of the validation knowledge base files, the sentence is evaluated as semantically correct, and no further processing is needed. The flags are set to indicate that the sentence has no syntax or semantic errors.

If none of the statements are found in their corresponding knowledge base files, the agent concludes that the sentence does not have a valid meaning, ergo, it has a semantic error. To correct the syntax error, the agent uses a procedure similar to that outlined in 5.6.7, to correct a syntax error. There is a significant difference, however. When correcting a syntax error, the position of the incorrect word is first identified. When there is a semantic error, it is known that the sentence doesn't make sense, but the agent cannot determine at the outset, which of the words should be replaced. Hence, the agent tries every possibility. To this end, the software agent performs a lookup of the common sense knowledge base, for each logic statement,

University of Cape Town

assuming each element of the logic statement in its turn, as incorrect.

The search for a match for each element of every possible logic statement, is identical to that described in the "Find Matching Word" module in 5.6.7, using ranking as an attempt to resolve multiple matches. Should there be more than one potential solution, from all these searches, the software agent takes the rank into account once again. If there is only one potential solution with the highest rank, it is used as the final match, otherwise the agent does not produce a solution and sets the flags accordingly.

If the agent finds the final match, it records the position in the sentence, in order to replace the word assumed to be invalid, and construct the *suggested_sentence*. The agent sets the flags to indicate that a logic error was present, but a solution is proposed.

5.7 Processing Examples

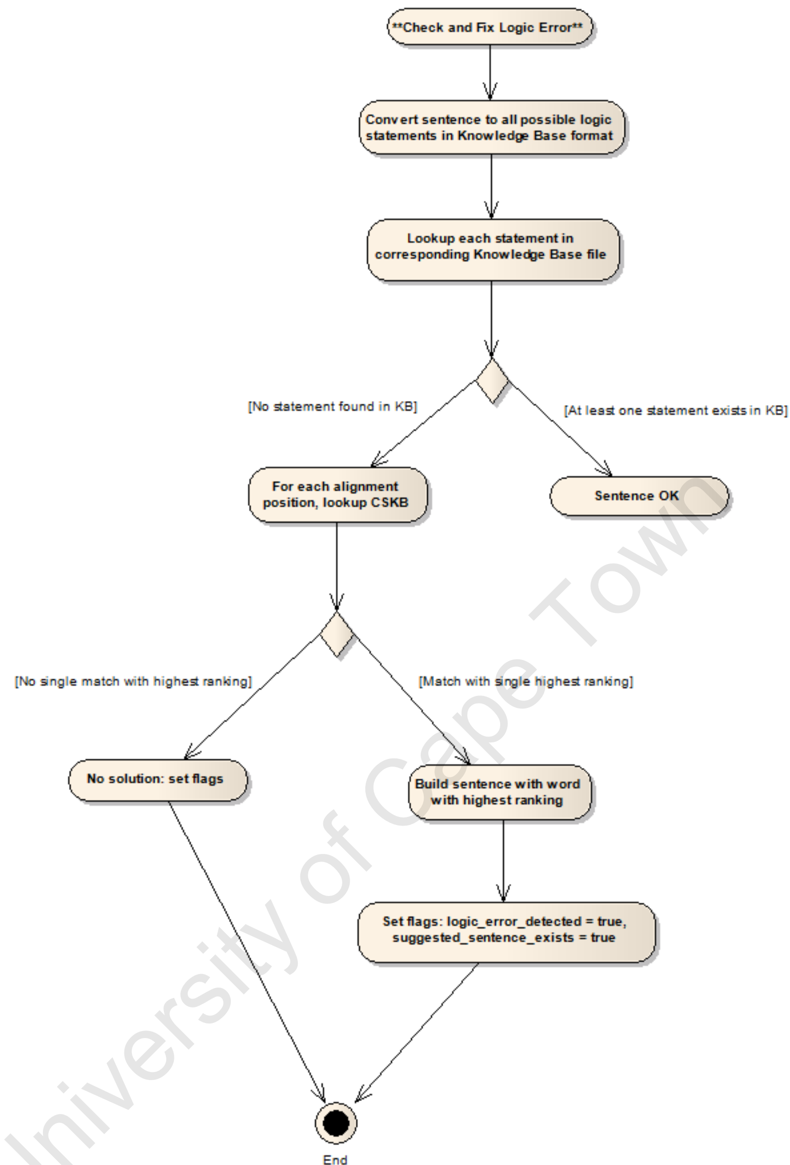
This section describes examples of how the software agent processes a line of text, using actual sentences produced by the Sphinx-4 ASR engine containing recognition errors. We do not provide the same level of detail as in Section 5.6, as we focus primarily on the main aspects of the process that could be clarified further with practical examples.

5.7.1 Syntax Error Example

Let us assume that the *original_sentence* is "the man catches the the". The software agent performs word tagging and detects that the fifth word violates the syntax, since a determinant is present instead of a noun. Figure 5.10 on page 59 illustrates this, showing the actual sentence label on top and the correct syntax at the bottom.

Using the classification database, the agent produces logic statements and performs a lookup in the corresponding knowledge base common sense files, as shown in Table 5.3 on page 60. The 'X' is used to mark the position of the incorrect word, in both the lookup statements and the CSKB statements, for the string comparison to be done correctly. The statement "PEOPLE CATCH BUS" is found in the

University of Cape Town



University of Cape Town

Figure 5.9: Check and Fix Logic Error

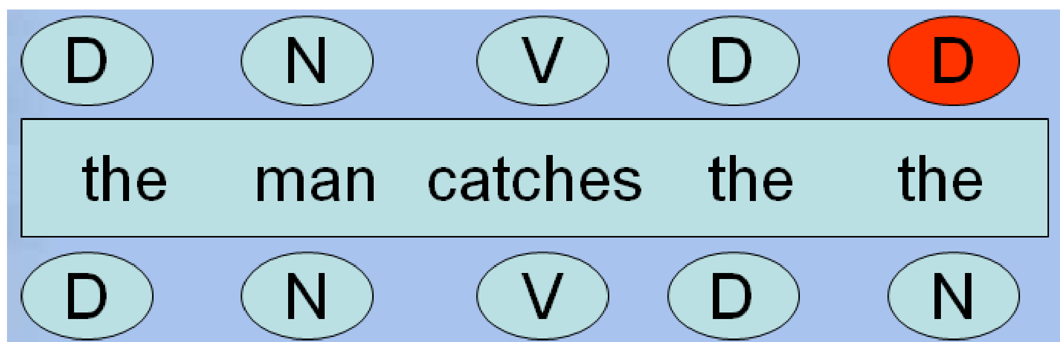


Figure 5.10: Example of Syntax Error Detection

Logic Statement to Search	Common Sense Knowledge Base File
MAN CATCH X	CS_SPEVSPE
PEOPLE CATCH X	CS_GENVSPE

Table 5.3: CSKB Lookup to Correct Syntax Error

Parsed Sentence	Corresponding Validation Knowledge Base File
WOMAN WATER MAN	SPEVSPE
WOMAN WATER PEOPLE	SPEVGEN
PEOPLE WATER MAN	GENVSPE
PEOPLE WATER PEOPLE	GENVGEN

Table 5.4: VKB Lookup for Deletion Error Case 1 Example

CS_GENVSPE file and "BUS" is therefore a single match. The agent uses this word to build the sentence "the man catches the bus".

5.7.2 Deletion Error Case 1 Example

Let us examine the following sentence from the ASR which is passed to the software agent for processing: "the woman waters man". The sentence label for this phrase is DNVN. The agent processes this as a Deletion Error Case 1 and fills in the missing determinant to produce the sentence "the woman waters the man". Having now generated a sentence that is syntactically correct, the agent proceeds to check if the sentence has a semantic error.

To check if the sentence has a valid meaning, the agent parses the sentence to produce all possible logic statements. Each statement is used to perform a lookup in the corresponding validation knowledge base file, as shown in Table 5.4 on page 60. None of the logic statements exist in the validation knowledge base and as a result, the agent concludes that the sentence has a semantic error.

Since the incorrect word isn't known a priori, the agent tries all valid combinations on each parsed sentence, considering that we only search for a match where there is a SPE or V element, as shown in Table 5.5 on page 61. This search would be equivalent to a human using common sense to ask themselves "what would water a man?" as in "X WATER MAN", or what "action is most common between people?" as in "PEOPLE X PEOPLE", taking into consideration the limited vocabulary and syntax.

University of Cape Town

Logic Statements to Search	CSKB File
X WATER MAN, WOMAN X MAN, WOMAN WATER X	CS_SPEVSPE
X WATER PEOPLE , WOMAN X PEOPLE	CS_SPEVGEN
PEOPLE X MAN, PEOPLE WATER X	CS_GENVSPE
PEOPLE X PEOPLE	CS_GENVGEN

Table 5.5: CSKB Lookup For Deletion Error Case 1 Example

Search	Match	CSKB File	Rank
PEOPLE X PEOPLE	PEOPLE LOVE PEOPLE	CS_GENVGEN	0
PEOPLE WATER X	PEOPLE WATER PLANT	CS_GENVSPE	1

Table 5.6: Matches for Deletion Error Case 1 Example

This search yields two matches, as shown in Table 5.6 on page 61. “LOVE” and “PLANT” in their respective positions, are thus potential solutions. However, since “PLANT” is found on a common sense knowledge base file with a higher rank, it is used as the final solution to build the *suggested_sentence*, which becomes “the woman waters the plant”.

University of Cape Town

5.7.3 Deletion Error Case 2 Example

In this example, the sentence from the speech recognition engine passed to the intelligent agent was “the mother the child”. The sentence label is DNDN and a syntax error is detected. The sentence label is classified as a Deletion Error Case 2 and processed accordingly. First, the agent fills in the missing verb, to create the temporary sentence “the mother dummy_verb the child” which is now syntactically correct. Since the position of the incorrect word is already identified, the agent performs a lookup on the common sense knowledge base as shown in Table 5.7 on page 62. There are two matches, ’FEED’ and ’LOVE’. The match with the highest rank is ’FEED’. Thus, the agent proceeds to lookup the first verb form entry in the verb database and finds the word ’feeds’. Replacing the ’dummy_verb’ with the correct word, the *suggested_sentence* becomes “the mother feeds the child”.

Search	Match	CSKB File	Rank
MOTHER X CHILD	MOTHER FEED CHILD	CS_SPEVSPE	2
MOTHER X PEOPLE	-	CS_SPEVGEN	1
PEOPLE X CHILD	-	CS_GENVSPE	1
PEOPLE X PEOPLE	PEOPLE LOVE PEOPLE	CS_GENVGEN	0

Table 5.7: Search and Matches for Deletion Error Case 2 Example

Search	Match	CSKB File	Rank
DOCTOR TREAT X	DOCTOR TREAT PATIENT	CS_SPEVSPE	2
PEOPLE TREAT X	-	CS_GENVSPE	1

Table 5.8: Search and Matches for Deletion Error Case 3 Example

5.7.4 Deletion Error Case 3 Example

The sentence to be analyzed in this example is "the doctor treated ate" which, like all the examples in this section, was actually produced by the speech recognition engine. The intelligent agent detects a syntax error from the sentence label, which is DNVV. Further processing indicates a deletion error. The sentence label is on the list and is categorized as a Deletion Error Case 3. The intelligent agent detects and replaces the incorrect word "ate" and fills in the missing determinant, to produce the temporary sentence "the doctor treated the dummy_noun". Knowing the error position, the software parses the sentence and searches the common sense knowledge base for a solution. This is shown in Table 5.8 on page 62. Only one match is found, making "PATIENT" the final solution. The software then replaces the dummy_noun with the match, to build the suggested_sentence as "the doctor treated the patient".

University of Cape Town

5.7.5 Semantic Error Example

The software agent was passed the following sentence for analysis: "the man melts the lawn". Word tagging reveals that this sentence has a DNVDN sentence label indicating a valid syntax. The agent then proceeds to check for semantic errors, by performing a lookup of the validation knowledge base, as seen in Table 5.9 on page 63. The software agent finds no matches in the validation knowledge base which indicates a semantic error.

In order to find a solution to the semantic error, the agent performs a lookup of

Parsed Sentence	Corresponding Validation Knowledge Base File
MAN MELT LAWN	SPEVSPE
MAN MELT OBJECTS	SPEVGEN
PEOPLE MELT LAWN	GENVSPE
PEOPLE MELT OBJECTS	GENVGEN

Table 5.9: VKB Lookup for Semantic Error Example

Logic Statements to Search	CSKB File
X MELT LAWN, MAN X LAWN, MAN MELT X	CS_SPEVSPE
X MELT OBJECTS, MAN X OBJECTS	CS_SPEVGEN
PEOPLE X LAWN, PEOPLE MELT X	CS_GENVSPE
PEOPLE X OBJECTS	CS_GENVGEN

Table 5.10: CSKB Lookup for Semantic Error Example

University of Cape Town

the common sense knowledge base as shown in Table 5.10 on page 63. It finds the matches shown inTable 5.11 on page 63, yielding two potential solutions, namely "LOVE" and "MOW". The latter was found in a common sense knowledge base file with a higher rank, and therefore becomes the final solution. The agent retrieves the first verb form in the verb database. Knowing the position of the correct word, the intelligent agent builds the suggested_sentence which becomes "the man mows the lawn".

5.8 Source Code

Approximately two thousand lines of Java code have been developed for the intelligent agent. Due to space constraints the entire source code is not included in this document, but can be found on the accompanying CD instead. Nevertheless, some important methods of the IntelligentAgent class are briefly discussed here. The source code for the tryFixingLogicError method is listed in this section. Comments from the original version have been reduced. This method implements the main algorithm used to correct semantic errors.

Search	Match	CSKB File	Rank
PEOPLE X OBJECTS	PEOPLE LOVE OBJECTS	CS_GENVGEN	0
PEOPLE X LAWN	PEOPLE MOW LAWN	CS_GENVSPE	1

Table 5.11: Matches for Semantic Error Example

```
String[] tryFixingLogicError(String[] parsed_sentence) {
      String[] match = {"", ""};
      String[] new_match = {"", ""};
      int number_of_matches = 0;
      int file_rank = 0;
      String cskb_file;
      int current_match_rank = -1;
      int[] x;
      x = new int[0];
      //Determine rank and establish x
      //for each statement that exists
      for (int i = 0; i < 4; i++) {
          if (parsed_sentence[i].equalsIgnoreCase("") == false) {
              if (i == 0) {
                  if (file_indicator[i].equalsIgnoreCase("GENV")) {
                      x = new int[1];
                      x[0] = 1;  //if GENV, search only GEN-X
                      file_rank = 0;
                  } else {
                //if SPEVSPE, search all combinations
                      x = new int[3];
                      x[0] = 0;
                      x[1] = 1;
                      x[2] = 2;
                      file_rank = 2;
                  }
              } else if (i == 1) {
               //if SPEV, or SPEVGEN, search X-V and SPE-X
               //or X-VGEN and SPE-X-GEN accordingly
                  x = new int[2];
                  x[0] = 0;
```

University of Cape Town

```
                x[1] = 1;
                file_rank = 1;
            } else if (i == 2) {
//if GENVSPE, search GEN-X-SPE, GENV-X
                x = new int[2];
                x[0] = 1;
                x[1] = 2;
                file_rank = 1;
            } else if (i == 3) {
//if GENVGEN, search GEN-X-GEN
                x = new int[1];
                x[0] = 1;
                file_rank = 0;
            }
            cskb_file = knowledge_base.kb_directory
             + "CS_" + file_indicator[i]
             + knowledge_base.kb_extension;
            new_match = lookupCommonSense(parsed_sentence[i],
                                                   cskb_file, x);
            if (new_match[0].equalsIgnoreCase("") == false) {
                if (file_rank >= current_match_rank) {
                    number_of_matches++;
                    if ((number_of_matches > 1) &&
                        (file_rank == current_match_rank)) {
                        match[0] = ""; //multiple solutions:
                        break;    //reset match and abort
                    } else {
                        match = new_match;
                        current_match_rank = file_rank;
                    }
                }
```

University of Cape Town

```
                }
            }
        }
        return match;
    }
```

The input to the tryFixingLogicError method is the parsed sentence, which is an array containing all the logic statements, as shown for example in Table 5.9 on page 63. The tryFixingLogicError method returns a String array containing the match and its position, or an array of empty strings, if it cannot find a solution.

As explained in the previous sections, if a word does not exist in the classification database, the 'GEN' element cannot be created, and therefore there will not be a parsed sentence with that element. For this reason, for each iteration of the loop, we first check that the parsed sentence actually exists. Each parsed sentence type in the array has a fixed index, which ranges from 0 to 3. The *file_indicator[]* array points to the corresponding knowledge base files and thus have the same range. If the parsed sentence was generated from a DNV sentence, then the first element of *file_indicator[]* will be 'GENV', otherwise it will be 'SPEVSPE'.

The *x[]* array, determines the positions in the logic statement, marked with X, where the agent will try to find a match. 0 indicates the first element, 1 the middle element which is always 'V', and 2 the third element. Since the 'X' can only be used in the place of a 'SPE' or a 'V', the length of *x[]* will vary according to the type of logic statement. For example, if the parsed sentence is of the type SPEVGEN, we want to search for "X V GEN" and "SPE X GEN" only. Hence we will have x[0] = 0, and x[1] = 1. It should be noted that the index of *x[]* does not mean anything. It is the content of the array that matters.

Using *file_indicator[]* we generate the name of the common sense knowledge base file for the current parsed sentence. This is passed to the lookupCommonSense method, along with the current parsed sentence and *x[]*. The lookupCommonSense method will search the common sense knowledge base file for matches, on all positions specified by *x[]*. If it finds more than one match in the CSKB file it returns an array of empty Strings, otherwise it returns the match and its position. Every new

University of Cape Town

match replaces the previous one if it ranks higher. However, more than one match with the same rank are immediately discarded.

5.9 The Genese Applications

During development and after completion of the intelligent agent, it was necessary to test the behaviour of the software using various sentences. Furthermore, it was useful to test the intelligent agent after making changes to the knowledge base. To this end, an application was developed to interface with the IntelligentAgent class. The ConsoleInteraction class, which is part of the genese package, is a simple application that takes a line of text, uses the software agent to analyze it and displays the result. It is a console based application. Figure 5.11 on page 69 shows a snapshot of the ConsoleInteraction application running under Microsoft Windows Vista, although it runs under Linux as well. The source code for ConsoleInteraction.java shown below, is an example of how to interface with the IntelligentAgent class. In addition, it shows how the TextUtilities class is used.

University of Cape Town

```
package genese;
import java.io.Console;
import java.io.IOException;
public class ConsoleInteraction {
    SpeechSentence myPhrase = new SpeechSentence();
    void interactWithUser()throws IOException{
        String[] split_sentence;
        //initialize console----------------
        Console c = System.console();
        if (c == null) {
            System.err.println("No console.");
            System.exit(1);
        }
        TextUtilities userInput = new TextUtilities();
```

```
        System.out.println("Type *exit to quit");
        String text_input = c.readLine("Enter a line of text: ");
        while (text_input.equalsIgnoreCase("*exit")==false){
            split_sentence = userInput.splitSentence(text_input);
            myPhrase.setOriginalSentence(split_sentence);
            myPhrase.analyzeSentence();
            printStatus();
            System.out.println("");
            text_input = c.readLine("Enter a line of text: ");
        }
    }
    void printStatus(){
        if (!myPhrase.syntax_error_detected){
            System.out.println("The syntax is correct...");
            if (myPhrase.logic_error_detected){
                System.out.println("But this sentence
                                      doesn't sound right.");
            } else
                System.out.println("And sentence
                                     appears to be normal.");
        } else
            System.out.println("Syntax error detected...");
        System.out.println("");

        if (myPhrase.suggested_sentence_exists){
            System.out.println("Did you mean?... ");
            printSentence(myPhrase.suggested_sentence);
        }else if (!myPhrase.sentence_OK)
            System.out.println("Sorry, I have no suggestions!");
    }
```

University of Cape Town

```
C:\Windows\system32\cmd.exe - java -jar genese.jar
C:\MSc_UCT\Development\Projects\genese\dist>java -jar genese.jar
Type *exit to quit
Enter a line of text: the mother the child
Syntax error detected...

Did you mean?...
the mother feeds the child

Enter a line of text: the woman waters man
Syntax error detected...

Did you mean?...
the woman waters the plant

Enter a line of text: the doctor treated ate
Syntax error detected...

Did you mean?...
the doctor treated the patient

Enter a line of text:
```

Figure 5.11: Snapshot of ConsoleInteraction

University of Cape Town

```
    void printSentence(String[] sentence){
        String sentence_line ="";
        for (int i=0; i < sentence.length; i++)
            sentence_line = sentence_line + " " + sentence[i];
        System.out.println(sentence_line.trim());
    }



    public static void main(String[] args)throws IOException{
        ConsoleInteraction user_interface = new ConsoleInteraction();
        user_interface.interactWithUser();
    }


}
```

To improve the user experience, a graphical user interface was developed to implement the same functionality of ConsoleInteraction. The VisualInteraction application under the genese package, interfaces with the IntelligentAgent class in the same way as ConsoleInteraction does. The application running under Windows Vista can be seen in Figure 5.12 on page 70.

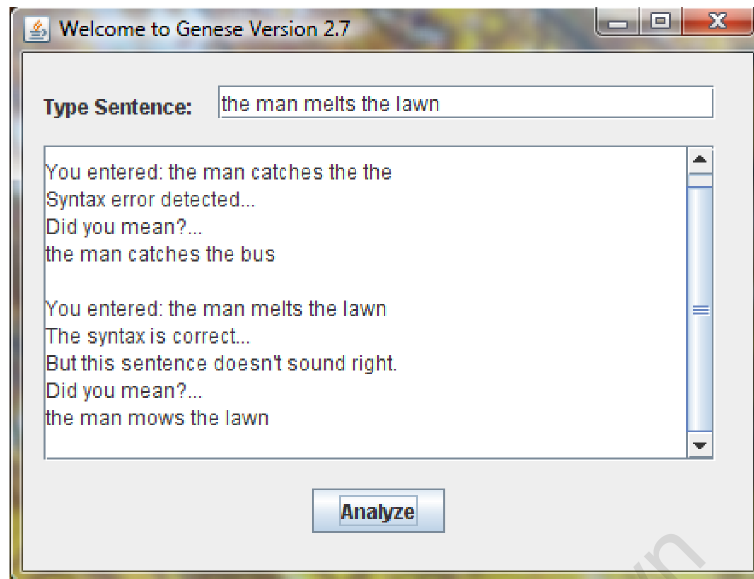


Figure 5.12: Snapshot of VisualInteraction

University of Cape Town

Another interesting application under the genese package is the SentenceGenerator class, first mentioned in Section 4.4 on page 28. This application was used to generate over 5000 sentences from about 200 statements in the common sense knowledge base. These sentences were used to generate a statistical language model for Sphinx-4. Some of the operations in the SentenceGenerator class are in a way, the reverse of some of the operations done by the parseSentence method, in the IntelligentAgent class.

Whereas the parseSentence method converts a sentence into logical statements, methods in the SentenceGenerator class take statements from the CSKB and convert them into actual sentences. This is done by retrieving all words corresponding to a 'GEN' element and all verb forms corresponding to a 'V' element, from the classification database and the verb database respectively. Using 'the' as a default determinant, the SentenceGenerator class then builds DNV and DNVDN sentences from all valid combinations that agree with the logic statements in the common sense knowledge base.

An extended version of the common sense knowledge base, identical to the one used to create the sentence corpus, can be found in Appendix C. After integrating the software agent with Sphinx-4, we later experimented with a more restricted version, for reasons discussed in Chapter 7.

Chapter 6

Integrating the Intelligent Agent with Sphinx-4

University of Cape Town

Although the software agent could be simply used to process text, it was developed specifically for speech recognition applications. In this chapter we describe how the intelligent agent was integrated into Sphinx-4, and we discuss the two main applications designed to test the augmented system.

6.1 Integration with Sphinx-4

The set of classes that represent the result of a recognition in Sphinx-4 are placed in the 'result' package. Since the software agent was designed to post process the recognition results, we incorporated the classes from the genese package shown in Figure 5.1 on page 40, in the result package as well. The files IntelligentAgent.java, SpeechSentence.java, KnowledgeBase.java and TextUtilities.java were copied to the "sphinx4\edu\cmu\sphinx\ result" folder. The source code for these files were modified to reflect their new package, as follows:

```
package edu.cmu.sphinx.result;
```

The speech applications described in the following sections, were created as demonstration programs and placed in the "sphinx4\demo\sphinx" folder under their own sub-folders. The 'demo.xml' ant file, which is used to build the Sphinx-4 demonstration programs was modified to include them. Next, Sphinx-4 and all the demos,

including our applications, were re-built, and new class and jar files were thus created. The configuration of the Sphinx-4 applications have already been discussed in Section 4.5.

6.2 The PlainSpeech Application

The PlainSpeech application was based on the Sphinx-4 hellodigits demo. However, instead of recognizing digits, it recognizes approximately 180 words. The application prompts the user to speak a sentence into the microphone. When it detects the end of the sentence, it performs recognition and prints the result on the screen. The recognition result is also appended to a text file. This application can therefore be used to create a text file from speech, by dictating one line at a time. When the user has finished dictating the sentence, they should speak the word 'goodbye'. The application prompts them to confirm by saying 'yes', or 'no' to cancel and return to the dictation session.

The PlainSpeech application does not make use of the intelligent agent. It was used to demonstrate the results and performance of a standard Sphinx-4 speech recognition application.

University of Cape Town

6.3 The SmartSpeech Application

The SmartSpeech Application is another console application, very similar to the PlainSpeech application, except that it makes use of the intelligent agent. The software agent analyzes each sentence produced by Sphinx-4 before printing the results to the screen or the text file. If it finds no errors, it prints the recognized sentence to the screen, and appends the line of text to the 'dictation.txt' file. If it finds a syntax or semantic error, it first corrects the sentence before printing the result to the screen and to the file. The error correction is therefore done transparently.

If the intelligent agent cannot correct the error detected, it prints the result to the screen with a message requesting the user to repeat the sentence, but does not write

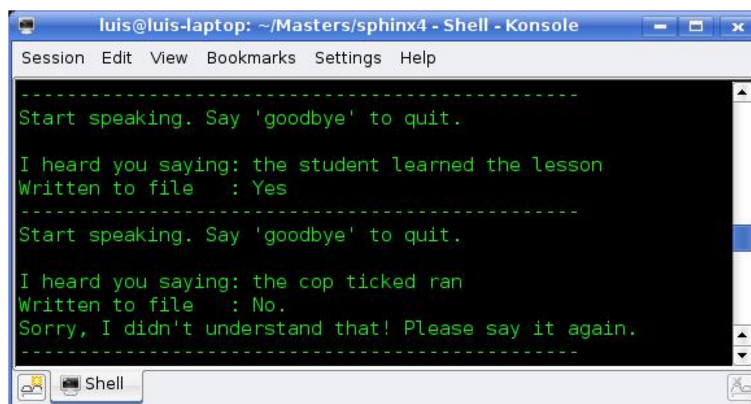


Figure 6.1: Snapshot of the SmartSpeech Application

the sentence to the file. Consequently, the sentences written to the dictation file will always be syntactically and semantically correct, in accordance with the knowledge base. This does not mean, of course, that they will always be the correct recognition of the actual spoken utterance. Sphinx-4 may produce a sentence that contains a recognition error but is syntactically and semantically correct. Or the software agent may incorrectly guess the spoken utterance when correcting recognition errors.

In addition, the SmartSpeech Application keeps a log of the entire dictation session, so that an analysis can be made. From the log, it is be possible to see what the results from the baseline recognizer were, and what corrections, if any, were made. The instances when the user is requested to repeat the sentence are also recorded in the log. An excerpt of the log from an actual dictation session is shown below.

University of Cape Town

```
Recognized Sentence: the drove eats the book
Sentence Status: Error detected
Written to file: Nothing! Requested user to repeat sentence.
----------------------------------------------------------------------
Recognized Sentence: fed no reads the book
Sentence Status: Error detected
Written to file: the girl reads the book
----------------------------------------------------------------------
Recognized Sentence: the caught lifted the bus
Sentence Status: Error detected
```

```
Written to file: the plane lifted the bus
----------------------------------------------------------------------
Recognized Sentence: the man catches the bus
Sentence Status: OK
Written to file: Original recognized sentence.
----------------------------------------------------------------------
```

6.4 The WavToText Application

The WavToText application was based on the Sphinx-4 transcriber demo. It reads an audio file containing speech, and writes the transcribed text to a file. For each sentence, the WavToText application appends two lines of text to the transcription file. The first line is the original sentence recognized by Sphinx-4, whereas the second line contains the corrected sentence, after being processed by the software agent. If the software agent doesn't find any error with the original sentence, or if it cannot correct the error found, then the second line is the same as the first one.

The audio file can be of any format readable by Java Sound, such as '.wav' or '.au'. However, the audio format must be as specified in the config.xml file, which is 16KHz, little endian, 16-bit signed PCM linear.

The code for the WavToText application is shown below. It can be seen how the application interfaces with the intelligent agent through the SpeechSentence class.

```
/*
 * Copyright 1999-2004 Carnegie Mellon University.
 * Portions Copyright 2004 Sun Microsystems, Inc.
 * Portions Copyright 2004 Mitsubishi Electric Research Laboratories.
 * All Rights Reserved.  Use is subject to license terms.
 *
 * See the file "license.terms" for information on usage and
 * redistribution of this file, and for a DISCLAIMER OF ALL
 * WARRANTIES.
 *
 */
/***Modified by Luis R. Lopes, University of Cape Town, 2008****/
package demo.sphinx.wavtotext;
import edu.cmu.sphinx.frontend.util.StreamDataSource;
import edu.cmu.sphinx.recognizer.Recognizer;
```

University of Cape Town

```
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
import edu.cmu.sphinx.util.props.PropertyException;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.io.PrintWriter;
import java.io.FileWriter;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;
import edu.cmu.sphinx.result.SpeechSentence;
import edu.cmu.sphinx.result.TextUtilities;
public class WavToText {
        public static void main(String[] args) {
        String transcribedtext;    //path to transcription file
        PrintWriter output_stream = null;
        SpeechSentence phrase = new SpeechSentence();
        TextUtilities text_util = new TextUtilities();


        String operating_system = System.getProperty("os.name");
        //Default OS is Windows. If we are running on Linux, update file path
        if (operating_system.substring(0, 5).equalsIgnoreCase("Linux")) {
            transcribedtext = "/home/luis/Masters/Temp/test/transcribedtext.txt";
        } else { //Use Windows Paths
            transcribedtext = "C:/MSc_UCT/Temp/test/transcribedtext.txt";
        }


        try {
            output_stream = new PrintWriter(new FileWriter(transcribedtext));

            URL audioURL;
            if (args.length > 0) {
                audioURL = new File(args[0]).toURI().toURL();
            } else {
                audioURL =
                    WavToText.class.getResource("default.wav");
            }
            URL configURL = WavToText.class.getResource("config.xml");
            ConfigurationManager cm = new ConfigurationManager(configURL);
            Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
            /* allocate the resource necessary for the recognizer */
            recognizer.allocate();
            AudioInputStream ais = AudioSystem.getAudioInputStream(audioURL);
            StreamDataSource reader =
```

University of Cape Town

```
        (StreamDataSource) cm.lookup("streamDataSource");
    reader.setInputStream(ais, audioURL.getFile());

    boolean done = false;
    int count = 0;
    System.out.println("Processing wave file...");
    while (!done) {
        /*
         * This while loop will terminate after the last utterance
         * in the audio file has been decoded, in which case the
         * recognizer will return null.
         */
        count++;
        if (count > 60) done = true;
        Result result = recognizer.recognize();
        if (result != null) {
            String resultText = result.getBestResultNoFiller();
            if (!(resultText.equalsIgnoreCase(""))){
                     System.out.println(resultText);
                     phrase.setOriginalSentence(
                     text_util.splitSentence(resultText));
                     //use intelligent agent to analyze sentence
                     phrase.analyzeSentence();
                     output_stream.println(resultText);
                     if (phrase.suggested_sentence_exists)
                        output_stream.println(
                        text_util.joinWords(phrase.suggested_sentence));
                     else
                        output_stream.println(resultText);
            }
        } else {
            done = true;
        }
    }
} catch (IOException e) {
    System.err.println("Problem when loading WavToText: " + e);
    e.printStackTrace();
} catch (PropertyException e) {
    System.err.println("Problem configuring WavToText: " + e);
    e.printStackTrace();
} catch (InstantiationException e) {
    System.err.println("Problem creating WavToText: " + e);
    e.printStackTrace();
} catch (UnsupportedAudioFileException e) {
    System.err.println("Audio file format not supported.");
    e.printStackTrace();
} finally {
```

University of Cape Town

```
                if (output_stream != null)
                    output_stream.close();
                    System.out.println("Done!");
            }
        }
    }
```

University of Cape Town

Chapter 7

Experimental Results

In this chapter we describe some of the experiments carried out with the speech recognition applications developed to test the effectiveness of the intelligent agent in improving recognition accuracy.

7.1 User Survey

One of the early experiments carried out, was a small survey designed to compare the two live speech recognition applications, namely PlainSpeech and SmartSpeech, described in previous chapters. We wanted to evaluate the user experience for both systems in a simple dictation task. In particular, we were interested in verifying if the ability of SmartSpeech to detect recognition errors and requesting the user to repeat the sentence when it was unable to perform a correction, would help produce better results in a dictation task. The subjective measure of performance would be judged by the amount of time required by the user to manually correct the automatically produced document, in order to create the error free final text document. The dictation task consisted of reading a list of 50 sentences to each application in turn.

Two users have completed this survey, with divergent opinions. One of the users has found that SmartSpeech was a better application because it wouldn't write nonsensical sentences to the text file, but would give the user the opportunity to repeat the sentence. Even though repeating the sentence didn't produce the desired result

University of Cape Town

Spoken Utterance	Sphinx-4 Output	SmartSpeech Output
the car overtook the bus	ticked ice overtook the bus	the car overtook the bus
the man paints the wall	the man paints the warmed	the man paints the wall
the girl wears the dress	the cat wins the dress	the cat wears the dress
the man mows the lawn	the man mows the girl	the man mows the lawn
the fire burned the house	the fire answered the house	the fire burned the house

Table 7.1: Examples of Corrections Made by SmartSpeech

in many instances, this user found that the overall results were better. The other user found PlainSpeech to be a better application. This user who happened to be a Swedish female, had an accent that was clear to humans. Nevertheless, her accent led to a considerable degradation in recognition accuracy by the ASR engine. As a result, SmartSpeech was unable to correct a great percentage of the incorrectly recognized sentences and frequently requested the user to repeat the sentence. Repeating the sentence several times often generated equal or worse recognition results, which led to frustration. Some examples, extracted from the dictation log, are shown below.

```
----------------------------------------------------------------------
Recognized Sentence: the woman watched dust the cow
Sentence Status: Error detected
Written to file: Nothing! Requested user to repeat sentence.
----------------------------------------------------------------------
Recognized Sentence: gentleman watches ticked mountain
Sentence Status: Error detected
Written to file: Nothing! Requested user to repeat sentence.
----------------------------------------------------------------------
Recognized Sentence: gentleman lock dust ticked mountain
Sentence Status: Error detected
Written to file: Nothing! Requested user to repeat sentence.
----------------------------------------------------------------------
```

Even though it was not visible to users, the overall word accuracy was improved with SmartSpeech in both cases. Examples for the latter user extracted from the dictation log, can be seen on Table 7.1 on page 79. Nevertheless, there is clearly room for improving the user interface for SmartSpeech, particularly with regards to the error detection feature. This is however, outside the scope of this research.

University of Cape Town

	Test 1	Test 2	Test 3	Test 4	Test 5
Original WER	11.59%	11.34%	7.56%	22.75%	13.87%
SmartSpeech WER	0.43%	0.84%	0.84%	8.15%	0.00%
WER Reduction	96.30%	92.59%	88.89%	64.15%	100.00%

Table 7.2: Word Error Rate Reduction for Several Tests

7.2 Experiments with SmartSpeech

One of the first points we wanted to test in SmartSpeech was speed performance. Unlike PlainSpeech, which immediately prints out the recognized sentence to the screen, SmartSpeech has to do a considerable amount of processing first. SmartSpeech checks the syntax of the sentence, then checks the meaning of the sentence by looking up the validation knowledge base, and if it finds any errors it attempts to correct them using the common sense knowledge base. The final result is then presented to the user. In all experiments we conducted, we have found no noticeable additional delays, despite the extra processing. We therefore concluded that SmartSpeech is, for practical purposes, as fast as PlainSpeech, with the size of our vocabulary and knowledge base.

The most important aspect was to verify increases in speech recognition accuracy. For this purpose, we used the sentence corpus shown in Appendix D. We have tried to create a list of sentences using words that are commonly associated with one another. The purpose of the software agent is to increase recognition accuracy by using common sense knowledge to guess the missing or incorrect words. This sentence list is therefore suited to test this ability.

Table 7.2 on page 80 shows the results that were recorded from five tests. The original word error rate (WER) was calculated from the sentences produced by the baseline recognizer, Sphinx-4. A comparison is made with the respective word error rate calculated from the output of SmartSpeech. The reduction in the word error rate ranged from 64% to 100% in these tests, with an average of 88%.

We also measured the sentence error rate for the five tests as shown in Table 7.3 on page 81. The reduction in sentence error rate ranged from 67% to 100%, with an average of 88%.

As one would expect, there is a relationship between the original WER and the

University of Cape Town

	Test 1	Test 2	Test 3	Test 4	Test 5
Original SER	40.82%	40.00%	32.00%	63.27%	58.00%
SmartSpeech SER	2.04%	4.00%	4.00%	20.41%	0.00%
SER Reduction	95.00%	90.00%	87.50%	67.74%	100.00%

Table 7.3: Sentence Error Rate Reduction for Several Tests

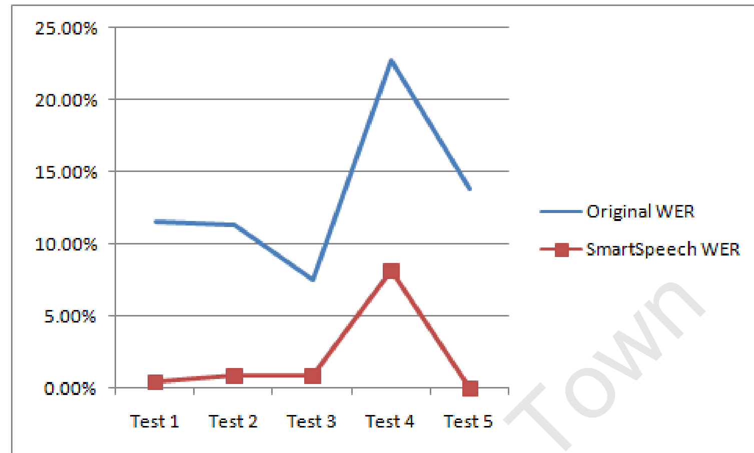


Figure 7.1: Comparison of Original WER and SmartSpeech WER

University of Cape Town

SmartSpeech WER, which can be seen clearly on Figure 7.1 on page 81.

Likewise, a higher original WER results in a lower reduction of the WER by the SmartSpeech, as displayed in the chart on Figure 7.2 on page 81. This is understandable if we take into account the fact that SmartSpeech needs a minimum of correctly recognized words, to have a better chance of accurately guessing the missing or incorrect word.

From these experiments it was possible to see what happens when there are two

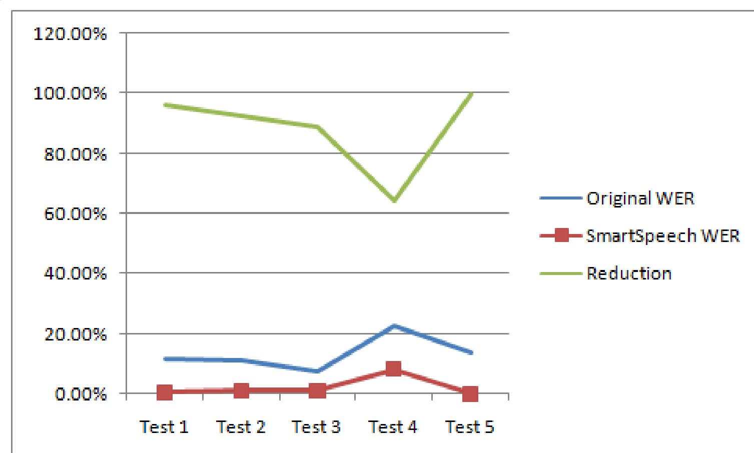


Figure 7.2: Original WER and SmartSpeech Reduction

Original WER	WER After Correction	Reduction
40%	26%	33%
17%	3%	80%

Table 7.4: Original WER and WavToText Reduction

potential solutions with the same rank. When the sentence "the man paints the lawn" was recognized, SmartSpeech requested the user to repeat the sentence. That is because there are in fact two possible solutions, "the man paints the wall" or "the man mows the lawn", and the agent could not resolve between the two, because they have the same rank.

7.3 Experiments with WavToText

University of Cape Town

The same sentences used for SmartSpeech were recorded in sequence to create the 'wav' files transcribed by the WavToText application. The results of two such tests can be seen in Table 7.4 on page 82. Once more it can be observed that there is a sharp decline in the WER reduction for a smaller increase in the original WER.

The reductions in word error rate are generally smaller compared to SmartSpeech because this is not an interactive application. If an error is detected but the intelligent agent is unable to correct it, then WavToText uses the original sentence as the output. In other words, it makes no improvement in accuracy. SmartSpeech rejects these nonsensical sentences altogether and therefore they do not count towards the word error rate. Nonetheless, WavToText shows a significant reduction in the word error rate. We have also tried recording with natural background noise, such as someone working in the kitchen. The text below is an excerpt of the transcription of the recording done with background noise from a television set. It should be recalled that WavToText writes the original sentence on the first line, and the processed text on the second line.

```
the man ice the bicycle
the man rides the bicycle
the time eats the caught
the time eats the caught
```

```
the woman watches the lawn
the woman watches the movie
dog watches drives
the dog watches the house
the girl played the cake
the girl bakes the cake
the sun shines
the sun shines
girl reads company
the girl reads the book
teacher started fitted dress
the teacher started the lecture
reads the plays shone
reads the plays shone
the maid feeds the bus
the maid catches the bus
```

The results have shown that background noise does decrease the accuracy as expected, but improvements were achieved with WavToText, due to the ability of the intelligent agent to guess some of the incorrectly recognized or missing words.

University of Cape Town

7.4 Adjusting the Knowledge Base

Having appreciated how the software agent works, along with several examples, it is now possible to give further consideration to some points related to knowledge engineering.

As discussed in Chapter 5, the intelligent agent does not look for a match if it would lead to a generic statement. As an example, let us consider the case when the recognized sentence is “the drove reads the book”. The word ’drove’ is detected as incorrect and the agent searches for a replacement word. The agent knows that “PEOPLE READ BOOK” but it will not use that statement because it would lead to ’PEOPLE’ as a possible solution. “PEOPLE” represents several words, such as

'girl'. 'man', 'woman', 'doctor' and many others.

One simple solution using the current algorithm of the software agent is to add a statement in the 'SPEVSPE' common sense knowledge base file, such as "GIRL READ BOOK". This will in effect provide a default word in case any of the other two need correction. Although this does bias the software agent to a certain extent, it is not the same as restricting it to a set of sentences. For instance, the sentence "patient read man tv" is still corrected to "the patient read the book".

Careful knowledge engineering is necessary in order to adjust or fine tune the knowledge base to suit a particular task, by choosing the classification adequately, and selecting the right logic statements for the validation knowledge base and the common sense knowledge base. Let us consider the example of the sentence "the man melts the lawn" which is corrected to "the man mows the lawn". This is only possible because we did not include the statement "PEOPLE MELT OBJECTS" in the validation knowledge base. Since words such as 'door', 'plant' and 'lawn' are classed as 'OBJECTS' for the purposes of the knowledge base, the statement "PEOPLE MELT OBJECTS" wouldn't suit all cases and is thus better left out. Of course, it would not be incorrect to include such a statement in the validation knowledge base since the meaning of the sentences would be valid, albeit unusual. However, doing so would allow more recognition errors to go undetected. There is therefore a trade-off that needs to be made according to each recognition task.

University of Cape Town

Chapter 8

Conclusions

8.1 Conclusions

Earlier versions of the software agent were able to correct substitution errors as intended. This was tested using the VisualInteraction application in the genese package as described in Section 5.9. However, the early versions were unable to correct deletion errors and therefore didn't produce significant improvements in recognition accuracy, when tested with a very small vocabulary set [50]. The final version of the intelligent agent as described in this document, is capable of handling deletion errors, which made a considerable difference when integrated with Sphinx-4. The two main applications developed, SmartSpeech and WavToText, have proved that simulating common sense reasoning can be used effectively to considerably reduce speech recognition errors.

The findings from this research have therefore satisfied the initial objectives, as set out in Chapter 1. Although the scope of this research was limited and the constrained syntax of the software agent presents an obstacle for many practical applications, there were benefits in focusing on a smaller subset of the larger problem, as evidenced by the results.

University of Cape Town

8.2 Comparison with Other Research

It is not our intention to directly compare our results with that of other researchers, since the systems were developed with different specifications and objectives. For reference purposes, however, the improvements in recognition accuracy for many other systems are mentioned in Chapter 3.

Some differences in approach are worth mentioning. Several methods rely only on the semantics of the sentence to post process the recognition results without checking the syntax, as seen in 3.2.5 and 3.2.6. This may affect the detection and correction of deletion errors, which are common in speech recognition. In addition, we have decided to split our knowledge base into two modules. One is used for validation and the other for correction. This allows the knowledge engineer to specify more clear boundaries between what is valid, and what is common sense, so that the intelligent agent can make more informed decisions. Furthermore, ranking logic statements in the knowledge base proved to be a useful technique, which allows the software agent to make more intelligent guesses.

We also used the ability of the software agent to detect errors, to prevent non-sensical sentences from being accepted and to allow a more intelligent interaction between the speech recognition system and the user, as demonstrated in the Smart-Speech application.

University of Cape Town

8.3 Recommendations

One obvious recommendation for future work is expanding the syntax allowed by the software agent, to eventually correctly process any valid English Sentence. This would also require developing a suitable knowledge base.

Transferring common sense knowledge to computers and machines has been a long standing challenge in the artificial intelligence community. In this regard, the work done by the Open Mind Common Sense (OMCS) project, described in 3.2.6 is highly commendable and could be very useful to speech recognition researchers, who might carry on the work done by Lieberman *et al.* [4].

Our general recommendation would be based on the motivation already set out

in Chapter 1, that speech recognition technology and artificial intelligence should be developed in conjunction, instead of being researched as two separate disciplines. Arguably, advances in artificial intelligence, coupled with additional progress in digital signal processing and statistical techniques, that address and resolve problems such as the "cocktail party effect", could give birth to the ideal speech recognition system, which would perform as well as any human.

University of Cape Town

Bibliography

[1] R. Rosenfeld *"Two decades of statistical language modeling: where do we go from here?",* Proceedings of the IEEE, vol. 88, August 2000.

[2] A. Sarma and D.D. Palmer *"Context-based speech recognition error detection and correction"* in Proc. Of the HLT-NAACL Conference: Short Papers, Boston, MA, 2004.

[3] S.R. Young *"The MINDS system: using context and dialog to enhance speech recognition"* in Proc. Of the workshop on Speech and Natural Language, Philadelphia, Pennsylvania, 1989.

[4] H. Lieberman, A. Faaborg, W. Daher, and J. Espinosa *"How to wreck a nice beach you sing calm incense"* in Proc. Of the 10th International Conference on Intelligent User Interfaces, New York, NY, 2005.

[5] B. Stefan and B. Denisa *"Advances in automatic speech recognition by imitating spreading activation"* in Text, speech and dialogue International conference No 6, Ceske Budejovice, Czech Republic, 2003, vol. 2807.

[6] R. Moore, J. Dowding, J. M. Gawron and D. Moran *"Combining Linguistic and Statistical Knowledge Sources in natural-language processing for ATIS"* in ARPA Spoken Language Technology Workshop, 1995.

[7] E. K. Ringger, J.F. Allen *"Error correction via a post-processor for continuous speech recognition"* in Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996, vol. 1.

University of Cape Town

[8] Shaojun Wang, Shaomin Wang, Russell Greiner, Dale Schuurmans, Li Cheng *"Exploiting Syntactic, Semantic and Lexical Regularities in Language Modeling via Directed Markov Random Fields"* in Proceedings of the 22nd international conference on Machine learning, Bonn, Germany, 2005.

[9] Dowding, John and Gawron, Jean Mark and Bear, John *"GEMINI: A natural language system for spoken-language understanding"* in Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics, 1993.

[10] Grau, Sergio and Segarra, Encarna and Sanchis, Emilio and Garcia , Fernando and Hurtado, Lluis F. *"Incorporating semantic knowledge to the language model in a speech understanding system"* in IV Jornadas en Tecnologia del Habla, 2006.

[11] W. Ward and S. Issar *"Integrating semantic constraints into the SPHINX-II recognition search"* in Proc. ICASSP'94, Adelaide, Australia, 1994.

[12] Dowding, John and Moore, Robert and Andry, Franqois and Moran, Douglas *"Interleaving syntax and semantics in an efficient bottom-up parser"* in Proc. of ACL-94, 1994.

[13] Jeong, Minwoo and Lee, Gary Geunbae *"Improving Speech Recognition and Understanding using Error-Corrective Reranking"*, ACM Transactions on Asian Language Information Processing (TALIP), vol. 7, no. 1, 2008.

[14] Jeong, Minwoo and Kim, Byeongchang *"Semantic-Oriented Error Correction for Spoken Query Processing"*, IEEE workshop on Automatic Speech Recognition and Understanding, 2003.

[15] Inkpen, Diana and Désilets, Alain *"Semantic Similarity for Detecting Recognition Errors in Automatic Speech Transcripts"*, 2005.

[16] Jeong, Minwoo and Jung, Sangkeun and Lee, Gary Geunbae *"Speech Recognition Error Correction Using Maximum Entropy Language Model"* in INTERSPEECH, 2004.

University of Cape Town

[17] Gurevych, Iryna and Porzel, Robert *"Using knowledge-based scores for identifying best speech recognition hypothesis"* in Proc. of ISCA Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems. Chateau-d'Oex-Vaud, 2003.

[18] Bühler, Dirk and Minker, Wolfgang and Elciyanti, Artha *"Using Language Modelling to Integrate Speech Recognition with a Flat Semantic Analysis"* in 6th SIGdial Workshop on Discourse and Dialogue, Lisbon, Portugal, 2005.

[19] Moore, Robert C. *"Using natural-language knowledge sources in speech recognition"* in Computational Models of Speech Pattern Processing, 1999.

[20] A, Hakan Erdogan and B, Ruhi Sarikaya and B, Stanley F. Chen and B, Yuqing Gao and B, Michael Picheny *"Using semantic analysis to improve speech recognition performance"* in Computer Speech and Language, vol.19, 2005.

[21] M. A. A. Tatham *"An Integrated knowledge base for speech synthesis and automatic speech recognition"*, Journal of Phonetics , vol. 13,1985.

[22] E. Trentin, M. Gori *"A survey of hybrid ANN/HMM models for automatic speech recognition"*, Neurocomputing, vol.37, 2001.

[23] L. Rabiner *"A tutorial on hidden markov models and selected applications in speech recognition"*, Proceedings of the IEEE, vol. 77, 1989.

[24] T. Hazen *"Automatic Alignment and Error Correction of Human Generated Transcripts for Long Speech Recordings"*, International Conference on Spoken Language Processing, Pennsylvania, 2006.

[25] Nanjo, Hiroaki and Lee, Akinobu and Kawahara, Tatsuya *"Automatic Diagnosis of Recognition Errors in Large Vocabulary Continuous Speech Recognition Systems"*, Joho Shori Gakkai Kenkyu Hokoku, 1999.

[26] Geoffrey Zweig *"Bayesian network structures and inference techniques for automatic speech recognition"*, New Computational Paradigms for Acoustic Modeling in Speech Recognition, 2003.

University of Cape Town

[27] J. Savage, E. Hernandez, G. Vazquez, A. Hernandez and A. Ronzhin *"Control of a Mobile Robot Using Spoken Commands"*, 9th Conference Speech and Computer, St. Petersburg, Russia, September 20-22, 2004.

[28] L. Deng, A. Acero, M. Plumpe and X. Huang *"Large-Vocabulary Speech Recognition under Adverse Acoustic Environments"* in Proc. Int. Conf. on Spoken Language Processing. Beijing, China, Oct, 2000.

[29] Mark Huckvale *"Learning from the experience of building automatic speech recognition systems"*, UCL Working Papers, Speech, Hearing and Language, 1996.

[30] C. Havasi, R. Speer, and J. Alonso *"ConceptNet 3: a Flexible, Multilingual Semantic Network for Common Sense Knowledge"* in Proceedings of Recent Advances in Natural Languages Processing, 2007.

[31] Christiane Fellbaum, editor. *"WordNet: An Electronic Lexical Database."* MIT Press, Cambridge, MA, 1998.

[32] Shen, Edward and Lieberman, Henry and Lam, Francis *"What am I gonna wear?: scenario-oriented recommendation"*, in Proceedings of the 12th International Conference on Intelligent User Interfaces, 2007.

[33] Frederick Jelinek *"Statistical Methods for Speech Recognition."* MIT Press, Cambridge, Massachusetts, 1997.

[34] Jerome R. Bellegarda *"Large vocabulary speech recognition with multi-span statistical language models"* IEEE Transactions on Speech and Audio Processing, 2000.

[35] Acero, A., Wang, Y., and Wang, K. *"A Semantically Structured Language Model"*, in Special Workshop in Maui (SWIM), Jan. 2004.

[36] H. Liu and P. Singh *"ConceptNet: a practical commonsense reasoning toolkit"*, BT Technology Journal, 2004.

University of Cape Town

[37] Y. Liu, N. V. Chawla, M. P. Harper, E. Shriberg, and A. Stolcke *"A study in machine learning from imbalanced data for sentence boundary detection in speech"*, Computer Speech and Language, vol. 20, October 2006.

[38] A. Stolcke, B. Chen, H. Franco, V. R. R. Gadde, M. Graciarena, M.Y. Hwang, K. Kirchhoff, N. Morgan, X. Lin, T. Ng, M. Ostendorf, K. Sönmez, A. Venkataraman, D. Vergyri, W. Wang, J. Zheng, and Q. Zhu *"Recent innovations in speech-to-text transcription at SRI-ICSI-UW"*, IEEE Transactions on Audio, Speech, and Language Processing, vol. 14, September 2006.

[39] V. R. R. Gadde, A. Stolcke, D. Vergyri, J. Zheng, K. Sonmez, and A. Venkataraman *"Building an ASR system for noisy environments: SRI's 2001 SPINE evaluation system"* in Proc. International Conference on Spoken Language Processing, vol. 3, (Denver, CO), September 2002.

[40] A. Stolcke, K. Boakye, Ö. Cetin, A. Janin, M. Magimai-Doss, C. Wooters, and J. Zheng *"The SRI-ICSI Spring 2007 meeting and lecture recognition system"* in Proc. NIST 2007 Rich Transcription Workshop, 2007.

[41] E. Shriberg, L. Ferrer, S. Kajarekar, N. Scheffer, A. Stolcke, and M. Akbacak *"Detecting nonnative speech using speaker recognition approaches"* in Proc. Odyssey Speaker and Language Recognition Workshop, (Stellenbosch, South Africa), January 2008.

[42] E. Segarra and L. Hurtado *"Construction of Language Models using Morfic Generator Grammatical Inference MGGI Methodology"* in Proc. of Eurospeech, 1997.

[43] Lawrence Rabiner and Bing-Hwang Juang. *"Fundamentals of Speech Recognition."* Prentice-Hall PTR, New Jersey, 1993.

[44] Dénes Paczolay and András Kocsor and László Tóth *"Real-time vocal tract length normalization in a phonological awareness teaching system"* in Text Speech and Dialogue, Springer, 2003.

University of Cape Town

[45] Victoria Fromkim, Robert Rodman and Nina Hyams *"An Introduction to Language."* Thomson Wadsworth, Boston, MA, 2007.

[46] Frederik Jelinek "Statistical Methods for Speech Recognition." The MIT Press, Cambridge, Massachusetts, 1997.

[47] Carnegie Melon University "Sphinx-4: A speech recognizer written entirely in the Java programming language" in http://cmusphinx.sourceforge.net/sphinx4, 1999-2004.

[48] Commonsense Computing Group "Open Mind Common Sense Project" in http://openmind.media.mit.edu, 2009.

[49] Stuart Russell and Peter Norvig *"Artificial Intelligence - A Modern Approach."* Pearson Education, New Jersey, 2003.

[50] Luis Lopes *"A Software Agent for Detecting and Correcting Speech Recognition Errors Using a Knowledge Base."* in Southern African Telecommunications Networks and Applications Conference, South Africa, September 2008.

University of Cape Town

Appendix A

Sample of Language Model

University of Cape Town

Since the language model is too large to be printed, only a small fraction is shown here. The 1-grams are shown entirely so that all the words used by the application can be found here. The 2-grams and 3-grams (trigrams) have been truncated. All three applications, namely SmartSpeech, PlainSpeech and WavToText use the same language model. Each application has its own copy of the language model in its respective JAR file.

```
Language model created by QuickLM on Wed Jul 23 14:12:28 EDT 2008
Copyright (c) 1996-2000
Carnegie Mellon University and Alexander I. Rudnicky
This model based on a corpus of 5742 sentences and 179 words
The (fixed) discount mass is 0.5
\data\
ngram 1=179
ngram 2=1686
ngram 3=4013
\1-grams:
-1.1423 </s> -0.3010
-1.1423 <s> -0.2345
-3.5589 ANSWERED -0.2346
-3.5589 ANSWERS -0.2346
-3.1090 APPLE -0.2685
-3.5396 ARRESTED -0.2346
-3.5589 ARRESTS -0.2346
-2.4983 ARTIST -0.2402
-2.9720 ATE -0.1965
-3.5589 BAKED -0.2346
-3.5396 BAKES -0.2346
-3.1090 BANANA -0.2685
-3.2482 BAPTIZED -0.2346
```

```
-4.9014 BARKED -0.2686
-4.6003 BARKS -0.2686
-3.2293 BEARD -0.2685
-2.6583 BICYCLE -0.2659
-3.2482 BILL -0.2686
-2.8295 BOOK -0.2676
-2.9771 BURNED -0.1965
-2.9823 BURNS -0.1965
-2.6559 BUS -0.2659
-2.8295 CAKE -0.2676
-2.6510 CAR -0.2659
-2.7370 CARRIED -0.2346
-2.7341 CARRIES -0.2346
-2.5854 CAT -0.2471
-3.2293 CATCHES -0.2346
-3.2482 CAUGHT -0.2346
-2.4931 CHILD -0.2402
-3.2112 CIRCUIT -0.2686
-2.9771 CLEANED -0.1965
-2.9720 CLEANS -0.1965
-3.4390 CLIMBED -0.2346
-3.4242 CLIMBS -0.2346
-2.8045 CLOCK -0.2668
-2.6839 COMPANY -0.2654
-2.4315 COP -0.2399
-2.5854 COW -0.2471
-2.6813 CRANE -0.2666
-3.9471 CRASHED -0.1965
-3.9983 CRASHES -0.1965
-2.5017 CUSTOMER -0.2402
-3.4100 DELIVERED -0.2346
-3.3962 DELIVERS -0.2346
-3.5589 DESIGNED -0.2346
-3.5396 DESIGNS -0.2346
-2.4983 DOCTOR -0.2402
-2.5710 DOG -0.2466
-2.7942 DOOR -0.2670
-3.4864 DRANK -0.2346
-2.6684 DRESS -0.2673
-3.4700 DRINKS -0.2346
-3.5396 DRIVES -0.2346
-3.5589 DROVE -0.2346
-3.1610 DUST -0.2676
-2.9720 EATS -0.1965
-2.4983 EMPLOYEE -0.2402
-2.5017 ENGINEER -0.2402
-3.1771 EXAM -0.2686
```

University of Cape Town

```
-4.9014 FED -0.2346
-4.6003 FEEDS -0.2346
-2.6583 FIRE -0.2676
-4.6003 FIRED -0.2346
-4.9014 FIRES -0.2346
-3.4864 FITS -0.1965
-3.5212 FITTED -0.1965
-2.5017 GENTLEMAN -0.2402
-2.4948 GIRL -0.2402
-4.9014 GOODBYE -0.2686
-2.9771 GRASS -0.2675
-4.1232 GREW -0.2686
-4.1232 GROWS -0.2686
-2.7491 HEATER -0.2674
-4.9014 HELD -0.2346
-4.6003 HOLDS -0.2346
-3.0093 HOUSE -0.2676
-4.1232 ICE -0.2685
-3.0093 KEY -0.2673
-3.5589 LAUGHED -0.2686
-3.5589 LAUGHS -0.2686
-2.8010 LAWN -0.2676
-4.1232 LEARNED -0.2346
-4.2024 LEARNS -0.2346
-4.2024 LECTURE -0.2678
-3.7874 LESSON -0.2678
-2.9876 LIFTED -0.1965
-2.9823 LIFTS -0.1965
-3.0321 LOAD -0.2676
-2.8222 LOCK -0.2676
-1.7928 LOVED -0.2346
-1.7928 LOVES -0.2346
-2.3391 MAID -0.2406
-2.8222 MAIL -0.2676
-2.4864 MAN -0.2402
-4.6003 MELTED -0.1965
-4.2993 MELTS -0.1965
-3.0206 MONEY -0.2676
-2.4983 MOTHER -0.2402
-2.8079 MOUNTAIN -0.2676
-2.6735 MOVIE -0.2671
-3.5589 MOWED -0.2346
-3.5396 MOWS -0.2346
-4.9014 NO -0.2686
-3.0206 OIL -0.2676
-3.2024 OPENED -0.1965
-3.2112 OPENS -0.1965
```

University of Cape Town

```
-3.6226 OVERTOOK -0.2346
-3.0818 PAID -0.2346
-3.5396 PAINTED -0.2346
-3.5034 PAINTS -0.2346
-3.1690 PASSED -0.1965
-3.1771 PASSES -0.1965
-2.4983 PATIENT -0.2402
-3.0753 PAYS -0.2346
-3.0149 PHONE -0.2668
-2.8295 PIANO -0.2676
-3.0206 PICTURE -0.2676
-2.8222 PIPE -0.2658
-3.2293 PIPELINE -0.2668
-2.5322 PLANE -0.2665
-2.7942 PLANT -0.2675
-3.5396 PLAYED -0.1965
-3.5212 PLAYS -0.1965
-2.5017 PLUMBER -0.2402
-2.4983 POSTMAN -0.2398
-2.4315 PRIEST -0.2399
-3.1610 RACE -0.2678
-4.6003 RAN -0.2686
-4.9014 RANG -0.2686
-2.5854 RAT -0.2471
-3.5589 READ -0.2346
-3.5396 READS -0.2346
-2.6865 REPAIRED -0.2346
-2.6919 REPAIRS -0.2346
-3.5396 RIDES -0.2346
-4.6003 RINGS -0.2686
-3.5589 RODE -0.2346
-4.6003 RUNS -0.2686
-3.5396 SHAVED -0.2346
-3.5589 SHAVES -0.2346
-4.6003 SHINES -0.2686
-4.9014 SHONE -0.2686
-3.1305 SHOWED -0.1965
-3.1232 SHOWS -0.1965
-2.8222 STAND -0.2676
-3.0952 STARTED -0.1965
-3.1020 STARTS -0.1965
-4.6003 STEALS -0.1965
-4.4242 STOLE -0.1965
-2.4931 STUDENT -0.2402
-2.9983 SUN -0.2674
-2.5017 SURFER -0.2402
-2.4983 TEACHER -0.2402
```

University of Cape Town

```
-2.4948 TECHNICIAN -0.2402
-4.6003 TELEVISION -0.2686
-0.8480 THE -0.2346
-2.4898 THIEF -0.2402
-4.9014 TICKED -0.2686
-4.6003 TICKS -0.2686
-3.1455 TIME -0.2679
-4.9014 TREATED -0.2346
-4.6003 TREATS -0.2346
-3.0563 TV -0.2668
-3.1771 WALL -0.2686
-3.5034 WARMED -0.2346
-4.9014 WARMS -0.2346
-3.1160 WATCHED -0.2346
-3.1020 WATCHES -0.2346
-3.1379 WATER -0.2685
-3.5589 WATERED -0.2346
-3.5396 WATERS -0.2346
-3.2482 WAVE -0.2686
-3.4700 WEARS -0.2346
-3.5589 WINS -0.2346
-2.4983 WOMAN -0.2402
-3.5396 WON -0.2346
-3.4864 WORE -0.2346
-4.9014 YES -0.2686
\2-grams:
-4.0601 <s> GOODBYE 0.0000
-4.0601 <s> NO 0.0000
-0.3013 <s> THE -0.0094
-4.0601 <s> YES 0.0000
-0.3010 ANSWERED THE -0.2995
-0.3010 ANSWERS THE -0.2995
-0.3153 APPLE </s> -0.3010
-2.0934 APPLE GREW 0.0000
-2.0934 APPLE GROWS 0.0000
-0.3010 ARRESTED THE -0.1730
-0.3010 ARRESTS THE -0.1730
-0.8349 ARTIST </s> -0.3010
-2.7042 ARTIST ANSWERED 0.0000
-2.7042 ARTIST ANSWERS 0.0000
-2.4031 ARTIST ATE -0.0200
-2.4031 ARTIST CAUGHT 0.0000
-2.7042 ARTIST CLEANED -0.0051
-2.7042 ARTIST CLEANS -0.0051
-2.7042 ARTIST CLIMBED 0.0000
-2.7042 ARTIST CLIMBS 0.0000
-2.7042 ARTIST DESIGNED 0.0000
```

University of Cape Town

```
-0.3010 WORE THE -0.2977
-0.3010 YES </s> -0.3010
\3-grams:
-0.3010 <s> GOODBYE </s>
-0.3010 <s> NO </s>
-3.7588 <s> THE APPLE
-1.8070 <s> THE ARTIST
-0.3010 WOMAN STARTS THE
-0.3010 WOMAN WATCHED THE
-0.3010 WOMAN WATCHES THE
-0.3010 WOMAN WEARS THE
-0.3010 WOMAN WINS THE
-0.3010 WOMAN WON THE
-0.3010 WOMAN WORE THE
-0.3010 WON THE RACE
-0.3010 WORE THE DRESS
\end\
```

University of Cape Town

Appendix B

Sphinx-4 Configuration

The XML configuration file for the SmartSpeech application is shown below. The file name used was smartspeech.config.xml located in the same directory as the .class file and packaged in the Jar file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
   Sphinx-4 Configuration file
-->
<!-- ******************************************************** -->
<!--  biship  configuration file                              -->
<!-- ******************************************************** -->
<config>
    <!-- ******************************************************** -->
    <!-- frequently tuned properties                              -->
    <!-- ******************************************************** -->
    <property name="absoluteBeamWidth"  value="500"/>
    <property name="relativeBeamWidth"  value="1E-80"/>
    <property name="absoluteWordBeamWidth" value="20"/>
    <property name="relativeWordBeamWidth" value="1E-60"/>
    <property name="wordInsertionProbability" value="1E-16"/>
    <property name="languageWeight" value="7.0"/>
    <property name="silenceInsertionProbability" value=".1"/>
    <property name="frontend" value="epFrontEnd"/>
    <property name="recognizer" value="recognizer"/>
    <property name="showCreations" value="false"/>

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

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

University of Cape Town

```
<!-- 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}"/>
</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">
```

University of Cape Town

```
    <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="wsj"/>
    <property name="languageModel" value="trigramModel"/>
    <property name="dictionary" value="dictionary"/>
    <property name="addFillerWords" value="false"/>
    <property name="fillerInsertionProbability" value="1E-10"/>
    <property name="generateUnitStates" value="false"/>
    <property name="wantUnigramSmear" value="true"/>
    <property name="unigramSmearWeight" value="1"/>
    <property name="wordInsertionProbability" value="${wordInsertionProbability}"/>
    <property name="silenceInsertionProbability" value="${silenceInsertionProbability}"/>
    <property name="languageWeight" value="${languageWeight}"/>
    <property name="unitManager" value="unitManager"/>
</component>


<!-- ******************************************************* -->
<!-- The Dictionary configuration                            -->
<!-- ******************************************************* -->
<component name="dictionary" ype="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
    <property name="dictionaryPath"  value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.
     Model!/edu/cmu/sphinx/model/acoustic/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/cmudict.0.6d"/>
    <property name="fillerPath" value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.
     Model!/edu/cmu/sphinx/model/acoustic/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/fillerdict"/>
    <property name="addSilEndingPronunciation" value="false"/>
    <property name="wordReplacement" value="&lt;sil&gt;"/>
    <property name="unitManager" value="unitManager"/>
</component>

<!-- ******************************************************* -->
<!-- The Language Model configuration                         -->
<!-- ******************************************************* -->
<component name="trigramModel" type="edu.cmu.sphinx.linguist.language.ngram.SimpleNGramModel">
    <property name="location" value="resource:/demo.sphinx.smartspeech.SmartSpeech!/demo/sphinx/smartspeech/smartspeech.lm"/>
    <property name="logMath" value="logMath"/>
    <property name="dictionary" value="dictionary"/>
    <property name="maxDepth" value="3"/>
    <property name="unigramWeight" value=".7"/>
</component>


<!-- ******************************************************* -->
<!-- The acoustic model configuration                         -->
<!-- ******************************************************* -->
<component name="wsj"
           type="edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model">
    <property name="loader" value="wsjLoader"/>
    <property name="unitManager" value="unitManager"/>
</component>

<component name="wsjLoader" type="edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.ModelLoader">
    <property name="logMath" value="logMath"/>
```

University of Cape Town

```
    <property name="unitManager" value="unitManager"/>
</component>
<!-- ******************************************************** -->
<!-- The unit manager configuration                           -->
<!-- ******************************************************** -->
<component name="unitManager" type="edu.cmu.sphinx.linguist.acoustic.UnitManager"/>

<!-- ******************************************************** -->
<!-- The frontend configuration                               -->
<!-- ******************************************************** -->

<component name="mfcFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
    <propertylist name="pipeline">
        <item>microphone </item>
        <item>premphasizer </item>
        <item>windower </item>
        <item>fft </item>
        <item>melFilterBank </item>
        <item>dct </item>
        <item>liveCMN </item>
        <item>featureExtraction </item>
    </propertylist>
</component>
<!-- ******************************************************** -->
<!-- The live frontend configuration                          -->
<!-- ******************************************************** -->
<component name="epFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
    <propertylist name="pipeline">
        <item>microphone </item>
        <item>speechClassifier </item>
        <item>speechMarker </item>
        <item>nonSpeechDataFilter </item>
        <item>premphasizer </item>
        <item>windower </item>
        <item>fft </item>
        <item>melFilterBank </item>
        <item>dct </item>
        <item>liveCMN </item>
        <item>featureExtraction </item>
    </propertylist>
</component>

<component name="microphone" type="edu.cmu.sphinx.frontend.util.Microphone">
    <property name="closeBetweenUtterances" value="false"/>
</component>

<component name="speechClassifier" type="edu.cmu.sphinx.frontend.endpoint.SpeechClassifier">
    <property name="threshold" value="13"/>
</component>

<component name="nonSpeechDataFilter" type="edu.cmu.sphinx.frontend.endpoint.NonSpeechDataFilter"/>

<component name="speechMarker"
            type="edu.cmu.sphinx.frontend.endpoint.SpeechMarker">
    <property name="speechTrailer" value="50"/>
</component>

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

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

<component name="fft"
```

University of Cape Town

```
        type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform"/>

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

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

    <component name="liveCMN"
                type="edu.cmu.sphinx.frontend.feature.LiveCMN"/>

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

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

    <component name="accuracyTracker"
                type="edu.cmu.sphinx.instrumentation.AccuracyTracker">
        <property name="recognizer" value="${recognizer}"/>
        <property name="showRawResults" value="false"/>
        <property name="showAlignedResults" value="false"/>
    </component>

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

University of Cape Town

Appendix C

Common Sense Knowledge Base

The contents of an extended version of the common sense knowledge base files is shown below.

```
CS_GENV
ANIMALS EAT
FRUIT GROW
PHONESET RING
TVSET BURN
TRANSPORT CRASH
PEOPLE LAUGH
OBJECTS BURN
CS_SPEV
BEARD GROW
CAR START
CHILD PLAY
CLOCK TICK
COMPANY GROW
DOG BARK
DOOR OPEN
DUST LIFT
FIRE BURN
GRASS GROW
ICE MELT
KEY FIT
LECTURE START
LESSON START
MAID CLEAN
MAN RUN
MOVIE START
PHONE RING
PLANE CRASH
PLANT GROW
```

University of Cape Town

```
RACE START
STUDENT PASS
SUN SHINE
THIEF STEAL
TIME SHOW
WATER RUN
CS_GENVGEN
ANIMALS EAT ANIMALS
ANIMALS EAT FRUIT
ANIMALS LOVE OBJECTS
ANIMALS LOVE PEOPLE
ANIMALS CRASH PHONESET
ANIMALS WATCH TRANSPORT
PEOPLE LOVE ANIMALS
PEOPLE EAT FRUIT
PEOPLE LOVE PEOPLE
PEOPLE ANSWER PHONESET
PEOPLE WATCH TVSET
TRANSPORT CARRY OBJECTS
TRANSPORT CARRY PEOPLE
TRANSPORT CARRY PHONESET
TRANSPORT CARRY TVSET
TRANSPORT OVERTAKE TRANSPORT
CS_GENVSPE
ANIMALS EAT CAKE
ANIMALS WEAR DRESS
ANIMALS OPEN DOOR
ANIMALS CLEAN DUST
ANIMALS PASS EXAM
ANIMALS START FIRE
ANIMALS EAT GRASS
ANIMALS LEARN LESSON
ANIMALS CLIMB MOUNTAIN
ANIMALS WATCH MOVIE
ANIMALS EAT PLANT
ANIMALS START RACE
ANIMALS PASS TIME
ANIMALS CLIMB WALL
ANIMALS DRINK WATER
PEOPLE BAKE CAKE
PEOPLE DESIGN CIRCUIT
PEOPLE SHAVE BEARD
PEOPLE RIDE BICYCLE
PEOPLE PAY BILL
PEOPLE READ BOOK
PEOPLE CATCH BUS
PEOPLE DRIVE CAR
```

University of Cape Town

```
PEOPLE REPAIR CLOCK
PEOPLE PAY COMPANY
PEOPLE REPAIR CRANE
PEOPLE OPEN DOOR
PEOPLE CLEAN DUST
PEOPLE WEAR DRESS
PEOPLE PASS EXAM
PEOPLE START FIRE
PEOPLE REPAIR HEATER
PEOPLE MOW LAWN
PEOPLE REPAIR LOCK
PEOPLE OPEN MAIL
PEOPLE CLIMB MOUNTAIN
PEOPLE WATCH MOVIE
PEOPLE PLAY PIANO
PEOPLE REPAIR PIPE
PEOPLE REPAIR PIPELINE
PEOPLE START PLANE
PEOPLE WATER PLANT
PEOPLE WIN RACE
PEOPLE REPAIR STAND
PEOPLE PASS TIME
PEOPLE PAINT WALL
PEOPLE DRINK WATER
PEOPLE CATCH WAVE
PHONESET BURN CIRCUIT
PHONESET START FIRE
TVSET BURN CIRCUIT
TVSET START FIRE
TRANSPORT START FIRE
CS_SPEVGEN
COP ARREST PEOPLE
COMPANY DELIVER ANIMALS
COMPANY DELIVER FRUIT
COMPANY DELIVER OBJECTS
COMPANY REPAIR PHONESET
COMPANY REPAIR TVSET
COMPANY REPAIR TRANSPORT
COMPANY PAY PEOPLE
CRANE LIFT OBJECTS
DRESS FIT PEOPLE
FIRE BURN ANIMALS
FIRE BURN OBJECTS
FIRE BURN PEOPLE
FIRE BURN PHONESET
FIRE BURN TVSET
FIRE BURN TRANSPORT
```

University of Cape Town

```
HEATER WARM OBJECTS
MOVIE SHOW ANIMALS
MOVIE SHOW FRUIT
MOVIE SHOW OBJECTS
MOVIE SHOW PEOPLE
MOVIE SHOW PHONESET
MOVIE SHOW TVSET
MOVIE SHOW TRANSPORT
PLANE LIFT ANIMALS
PLANE LIFT FRUIT
PLANE LIFT OBJECTS
PLANE LIFT PEOPLE
PLANE LIFT PHONESET
PLANE LIFT TVSET
PLANE LIFT TRANSPORT
PRIEST BAPTIZE PEOPLE
TECHNICIAN REPAIR TVSET
TECHNICIAN REPAIR PHONESET
CS_SPEVSPE
ARTIST PAINT PICTURE
CAR OVERTAKE BUS
CAT EAT RAT
CLOCK SHOW TIME
COW EAT GRASS
COMPANY REPAIR CLOCK
COMPANY FIRE EMPLOYEE
COMPANY PAY ENGINEER
COP ARREST THIEF
CRANE LIFT LOAD
CUSTOMER PAY BILL
DOCTOR TREAT PATIENT
DOG WATCH HOUSE
DOG CATCH THIEF
DRESS FIT GIRL
ENGINEER DESIGN CIRCUIT
FIRE BURN HOUSE
GENTLEMAN OPEN DOOR
GIRL READ BOOK
GIRL BAKE CAKE
GIRL LOVE MAN
GIRL WEAR DRESS
GIRL DRINK WATER
HEATER WARM HOUSE
KEY FIT LOCK
MAID CLEAN DUST
MAN RIDE BICYCLE
MAN CATCH BUS
```

University of Cape Town

```
MAN DRIVE CAR
MAN CLIMB MOUNTAIN
MAN SHAVE BEARD
MAN MOW LAWN
MAN PLAY PIANO
MAN PAINT WALL
MOTHER FEED CHILD
PIPE CARRY WATER
PIPELINE CARRY OIL
PLUMBER REPAIR PIPE
POSTMAN DELIVER MAIL
PRIEST BAPTIZE CHILD
STAND HOLD TV
STUDENT LEARN LESSON
STUDENT PASS EXAM
SUN MELT ICE
SUN WARM WATER
SURFER CATCH WAVE
TEACHER START LECTURE
TECHNICIAN REPAIR TV
THIEF STEAL MONEY
WALL HOLD PICTURE
WOMAN LOVE MAN
WOMAN WATCH MOVIE
WOMAN WATER PLANT
WOMAN WIN RACE
```

University of Cape Town

Appendix D

Test Sentences

The following sentences were used to test the SmartSpeech and WavToText applications.

```
the girl reads the book
the car overtook the bus
the man catches the bus
the woman watches the movie
the man paints the wall
the man rides the bicycle
the cow eats the grass
the woman waters the plant
the dog watches the house
the girl bakes the cake
the sun shines
the girl wears the dress
the teacher started the lecture
the doctor treats the patient
the maid cleans the dust
the man mows the lawn
the ice melted
the fire burned the house
the man drives the car
the man climbs the mountain
the phone rang
the cat ate the rat
the thief stole the money
the gentleman opened the door
the sun melts the ice
the student passed the exam
the man shaved the beard
the dog barked
the dress fits the girl
```

University of Cape Town

```
the artist paints the picture
the technician repaired the tv
the girl drinks the water
the pipeline carries the oil
the plumber repaired the pipe
the company fired the employee
the priest baptized the child
the mother feeds the child
the crane lifts the load
the key fits the lock
the surfer catches the wave
the man plays the piano
the plane crashed
the customer pays the bill
the postman delivers the mail
the engineer designs the circuit
the student learned the lesson
the clock ticked
the clock shows the time
the heater warms the house
the stand holds the tv
```

University of Cape Town

Appendix E

Contents of CD-ROM

The following items can be found in the accompanying CD-ROM:

- This entire document in PDF format
- The source code for the genese package, which contains the applications described in this document
- The JAR files for the PlainSpeech, SmartSpeech and WavToText applications
- The Sphinx-4 speech recognition system, with the integrated software agent
- Documents and research papers used in this research

University of Cape Town