AUTOMATED REGRESSION TESTING APPROACH TO EXPANSION AND REFINEMENT OF SPEECH RECOGNITION GRAMMARS

by

RAUL AVINASH DOOKHOO
B.S. University of Guyana, 2004

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2008

ABSTRACT

This thesis describes an approach to automated regression testing for speech recognition grammars. A prototype Audio Regression Tester called ART has been developed using Microsoft's Speech API and C#. ART allows a user to perform any of three tasks: automatically generate a new XML-based grammar file from standardized SQL database entries, record and cross-reference audio files for use by an underlying speech recognition engine, and perform regression tests with the aid of an oracle grammar. ART takes as input a wave sound file containing speech and a newly created XML grammar file. It then simultaneously executes two tests: one with the wave file and the new grammar file and the other with the wave file and the oracle grammar. The comparison result of the tests is used to determine whether the test was successful or not. This allows rapid exhaustive evaluations of additions to grammar files to guarantee forward process as the complexity of the voice domain grows.

The data used in this research to derive results were taken from the LifeLike project. However, the capabilities of ART extend beyond LifeLike. The results gathered have shown that using a person's recorded voice to do regression testing is as effective as having the person do live testing. A cost-benefit analysis, using two published equations, one for *Cost* and the other for *Benefit*, was also performed to determine if automated regression testing is really more effective than manual testing. *Cost* captures the salaries

of the engineers who perform regression testing tasks and *Benefit* captures revenue gains or losses related to changes in product release time. ART had a higher benefit of $21461.08 when compared to manual regression testing which had a benefit of $21393.99. Coupled with its excellent error detection rates, ART has proven to be very efficient and cost-effective in speech grammar creation and refinement.

To my Mom and Dad for their unlimited love and support

ACKNOWLEDGMENTS

I would like to acknowledge and extend my sincere gratitude to Dr. Ronald DeMara, my advisor, for his constant guidance, support and encouragement. Appreciation is also extended to my committee members, Dr. Avelino Gonzalez and Dr. Damla Turgut for their comments and suggestions. A show of thanks goes out to the National Science Foundation for their continued support of the research conducted at the UCF Intelligent Systems Laboratory. I would also like to thank my colleagues for their words of wisdom and encouragement throughout the entire writing process. Above all, I would like to thank my family for their never-ending support of all of my endeavors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

AI	Artificial Intelligence
ART	Audio Regression Tester
CMU	Carnegie Mellon University
COTS	Commercial-Off-The-Shelf
DARPA	Defense Advanced Research Projects Agency
DLL	Dynamic-Link Library
HMM	Hidden Markov model
LPC	Linear predictive Coding
NSF	National Science Foundation
SAPI	Speech Application Programming Interface
SDK	Software Development Kit
SR	Speech Recognizer
SRC	Speech Recognizer Control
SRE	Speech Recognition Engine
SRGS	Speech Recognition Grammar Specification
TCP/IP	Transmission Control Protocol/Internet Protocol
XML	Extensible Markup Language

CHAPTER 1**: INTRODUCTION**

1.1 Research Objective

The objective of this thesis was to formulate a development and testing approach for domain specific speech vocabularies and grammar rules. The benefits of this approach are realized in knowledge creation and entry phases of the lifecycle, as well as during maintenance. The result is a novel development environment that allows speech grammars to be refined with reduced labor cost and minimal side effects, while supporting a rich interaction-information abstraction in the native domain. As changes are made to grammars, ideally the interaction and rule-firing complexities are minimized for the developer. This yields a new *regression testing approach* to speech recognition grammars described herein.

In speech recognition problems, as the amount of domain knowledge increases then the likelihood of recognition errors also increases. This research proposes a method to reduce errors in speech recognition system development and aid in context-specific words being more accurately recognized. In order to reduce the amount of errors every time the knowledge base is expanded, we need to continually test the system to ensure the newly entered knowledge does not have a negative net effect on the previously entered knowledge.

Testing a system continuously becomes a tedious task, if for every iteration; the grammar set increases in size. Regression testing ensures a previous problem is not re-introduced into a subsequent iteration of the program. Manual regression testing in the speech domain is laborious and requires hours of painstaking testing, potentially with multiple users who must recite many phrases to assess adequate coverage. However, a practical development environment implies limited labor resources and thus we seek to optimize the testing process.

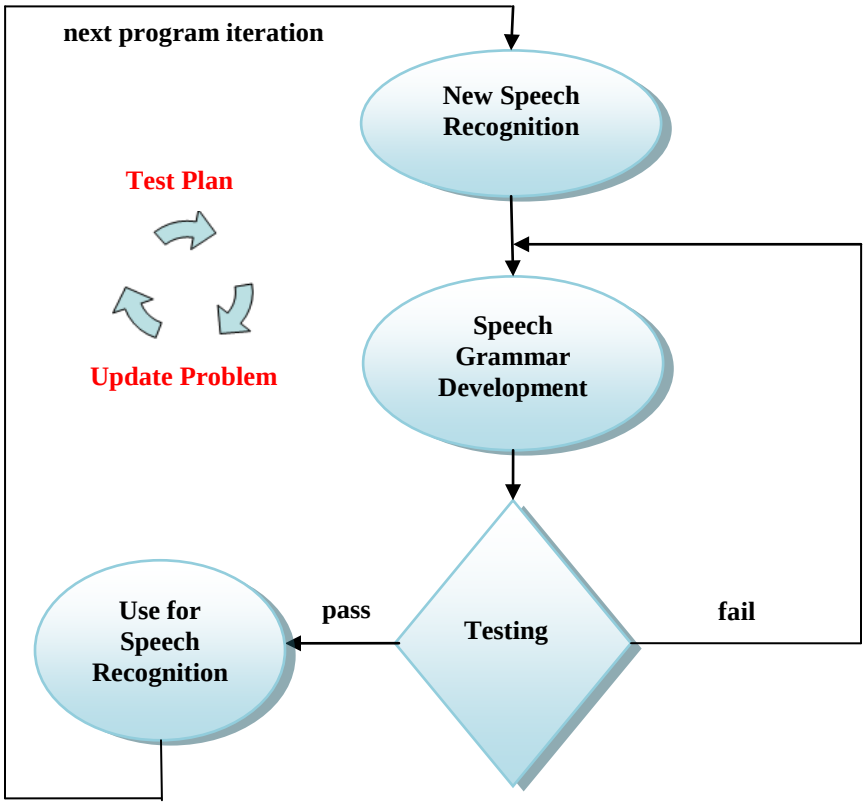


Figure 1: Speech Development Grammar Development and Testing Process

As shown in Figure 1, speech recognition systems require extensive iterative testing during development. The process begins with a behavioral description of the Speech Recognition Specification. This represents the desired recognition capability for phrases in a certain domain. The desired phrases are then realized as grammars which encode the time sequence of the constituent words. Grammars correspond to recognition rules. Each grammar is a form of template with fixed words which are required for the template to fire and also empty slots that can be filled with variable content. These rules are then added to the previous grammar list, thus increasing the speech recognition knowledge base. However, before deployment in an actual system, the complexity of grammar rules and its interaction with the underlying phoneme recognition process requires that testing be performed to assess coverage and correctness of the grammar set as a whole. This cycle corresponds to an instance of the *Test Plan Update Problem* (Leung & White, 1989).

Finding a solution to the Test Plan Update Problem is motivated by the fact that testing time and financial resources to do manual testing are limited. Automated regression testing on the other hand can assist greatly with testing in the speech domain by reducing time and money needed to do functional testing.

As shown in Figure 1 when a set of speech grammars are developed, a series of tests are needed to ensure these grammars function according to specification. If they pass the

series of tests applied then they become available for use in the speech recognition application; if not then modifications need to be made and more testing is required to ensure realization of the behavioral specifications.

With manual regression testing, for the next iteration of the program, the user needs to repeat the first series of tests and then do the second series of test. This can lead to low levels of output since development is bogged down in the testing phase. Automated regression testing allows the user to quickly rerun the initial tests with ease. This thesis represents the first known attempt to make the benefits of automated regression testing applicable to the domain of speech recognition problems.

1.2 Background

Speech recognition is the process of converting an acoustic signal, captured by a microphone, into written text. A *Speech Recognition Engine (SRE)* is a software program that converts audio data to recognized speech. Significant strides have been made in the field of speech recognition over the past few decades (Chen, Rosebblum, & Vo, 1994) (Gupta, Harrold, & Soffa, 1992) (Memon, Banerjee, Hashmi, & Nagarajan, 2003) (Leung & White, 1992) (DeMara, et al., 2008). It has been a topic of focus in the medical, military, telecommunications and educational fields. For many persons, the ability to have a spoken conversation with a computer represents one of the ultimate challenges to the understanding of the production and perception processes involved in human

communication (Baker, 1975) (Chow, et al., 1987) (Dupont & Luettin, 2000) (Chow, et al., 1987) (Lee, Hon, & Reddy, 1990) (DeMara, et al., 2008).

Many speech recognition systems are based on *Hidden Markov Models (HMM)* which were first introduced to speech recognition research in 1975 (Baker, 1975). A HMM is a statistical model that outputs a sequence of symbols or quantities. More precisely, the HMM is a probabilistic pattern matching technique in which the observations are probabilistic functions of the state. The widespread popularity of HMMs can be attributed to its simple algorithmic structure which is straight-forward to implement and is better suited to phoneme recognition than alternative recognition structures. Over the past 30 years, although speech recognition has come so far that commercially-available products can support isolated word recognition rates for continuous speech, the problem of completely fluent speaker-independent speech recognition still requires specification of domain knowledge on valid word orderings.

One approach to narrow the scope of the problem is to use speech recognition grammars. This is a faster and more accurate approach than an unconstrained search and yields better results as grammars provide a coding language to specify phrase contents. Instead of having endless possibilities to match within an open domain, the scope has been reduced to just a few words corresponding to the slots in each grammar template.

1.2.1 Speech Recognition Grammar

Speech Recognition Grammar Specification (SRGS) is a W3C standard for how speech recognition grammars are specified (W3C, 2004). A speech recognition grammar is a set of word patterns that are used primarily to indicate to the speech recognizer what to expect from the user; specifically, words that may be spoken, patterns in which those words occur and the spoken language surrounding each word. The syntax of the grammar format can be specified in two forms:

- ABNF or Augmented Backus-Naur Form – this is a non-XML plain text representation similar to traditional BNF grammar (W3C, 2004).
- XML or Extensible Markup Language – this syntax uses XML elements to represent the grammar constructs (W3C, 2004).

Both the ABNF and XML forms and have the expressive power of a Context-Free Grammar (CGF) and are specified to ensure the two representations are semantically mappable. It is be possible to convert from one form to the other and achieve identical semantic performance of the grammars. Semantic equivalence implies that both grammars accept the same language as input and reject the same language as input and, both grammars parse any input string identically.

This research focuses primarily on the use of the XML based grammar format which was influenced by the use of Microsoft's *Speech API (SAPI)* version 5 SRE. Thus, Microsoft SAPI 5 specifies a CFG structure and grammar rule format using XML. A grammar compiler transforms the XML grammar into a SAPI 5 binary format for the SAPI 5 compliant SRE. A SAPI 5 grammar text file is comprised of XML grammar elements and attributes that express one or more rules i.e. recognizable utterances. An example of a SAPI 5 compatible file of rules is show in Figure 2.

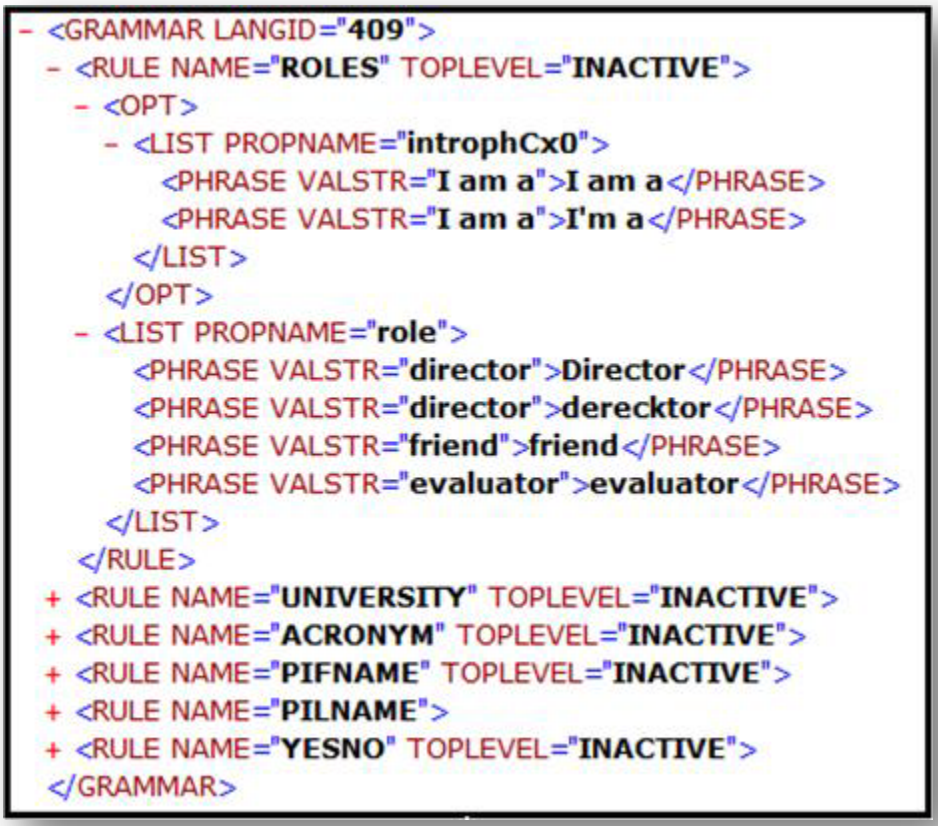


Figure 2: Condensed XML Grammar File (DeMara, et al., 2008)

In Figure 2, the outermost tags, <GRAMMAR>, define the bounds of the grammar. Within the grammar, rules are defined using the <RULE> tags. Each rule contains one or more phrases, specified by <PHRASE> tags. These phrases are the actual words being recognized in a specific rule. Rules can either be active or inactive. When a rule is active, it indicates to the Speech Recognizer (SR) that it should listen for those words.Figure 2 represents the XML grammar file used in LifeLike, the project on which this research is based (DeMara, et al., 2008). Each rule name represents a different *context* in the LifeLike domain. An overview of LifeLike will be presented later in this chapter.

1.2.2 Motivating Example

The task of manually building and testing grammars is most straightforward when there are a limited number of word interactions that need to be recognized. However, the grammar specification task rapidly encounters conflicting and cumbersome rule interactions as the word count increases.

For example, the rule “UNIVERSITY” in Figure 2 has been contracted in the view-space, i.e. the entire rule is not shown in figure, but actually contains over one hundred University names that need to be recognized in Project LifeLike. These Universities are part of the National Science Foundation (NSF) project lexicon to be recognized. To realize the application-level goal that LifeLike supports speech recognition from the directors of any of the Centers at the Universities, it is impossible to predict the

recognition behavior without testing the utterances of each University name. Thus, the goal of this grammar rule is to facilitate better speech recognition performance when the director identifies his University to the system. The system then uses "stored knowledge" to access this University name to properly identify the user. This creates a large test plan update problem as the set of grammars is developed to support recognition of a meaningful conversation.

1.3 LifeLike System Overview

LifeLike is a NSF-funded project that involves the development of an interactive avatar prototype of a NSF program manager (DeMara, et al., 2008). The objective of the LifeLike project is to enable domain-specific conversation with a realistic avatar. The system architecture of LifeLike composes the *LifeLike Speech Recognizer*, the *LifeLike Dialog Manager* and the *LifeLike Responsive Avatar Framework (RAF)* and a SQL Database that stores domain-specific information. The LifeLike Speech Recognizer uses speech input and translates it to words which are then processed by the Dialog Manager where it is refined and finally passed to the Responsive Avatar Framework (DeMara, et al., 2008).

1.3.1 LifeLike Dialog Manager

The Dialog Manager (DM) is the central module in the LifeLike system. It directs conversation flow and maintains the current context of the conversation. This context is

relayed to the Recognizer and enables it to focus the recognition task. Processed speech input from the recognizer is sent to the DM module which uses it ontology to disambiguate the data. The disambiguated data is then channeled to the RAF.

1.3.2 LifeLike Responsive Avatar Framework (RAF)

Creating a realistic active digital representation of a particular human being is a challenging and multifaceted task. Creating a realistic active digital representation of particular human being is a challenging and multifaceted task. Initially, investigations were conducted to identify and evaluate the interoperability of COTS packages for facial modeling, rendering of real-time graphics, motion-capture, and text-to-speech synthesis. The result was a customized Graphical Asset Production Pipeline which encapsulates the tasks needed to create a visual representation of a human character (DeMara, et al., 2008).

1.3.3 LifeLike Speech Recognizer

Speaker-independent audio input from a microphone headset is passed to the Speech Recognizer (SR) where it is processed by an SRE. There are two forms of recognized speech data. The first form utilizes the grammar XML file, where a context-specific rule is made active and the speech utterance is matched against the phrases in that rule. This narrow scope of words allows for a more precise match and better recognition results. If the Recognizer did not find a match of high confidence within the grammars, the second

form is used. In this form, the SR uses a generic non-customized grammar-free lexicon against which the utterance is matched.

Since the grammar XML file allows better recognition rates, research in this area was intensified. Initially, a small prototype grammar file was built and tested with much success. However, as the knowledge base grew, building and testing the file manually proved futile and lacked ingenuity. The need for an autonomous audio regression testing system resulted.

1.4 Regression Testing Process

As software is developed, reemergence of faults is quite common. These faults could be a result of fixes being lost due to poor revision control or human error in revision control. In the case of grammar testing, the cause is due to nuances in the interaction between rules. This can be viewed as an instance where changes to impact a specific problem have side effects on more general cases which become evident as the knowledge base grows. In mainstream software development, it is considered good practice when a bug is located and fixed; a test that reveals the bug is recorded and regularly retested after subsequent changes to the program.

According to Leung and White (Leung & White, 1989) *regression testing* is the testing process which is applied after a properly working program has been modified. It involves

testing the modified program with test cases in order to re-establish confidence that the program will perform according to its specification. These test cases form what is known as the *test bucket*. Regression testing is a major part of software maintenance where the software system may be corrected, adapted to its new environment, or enhanced to improve its performance.

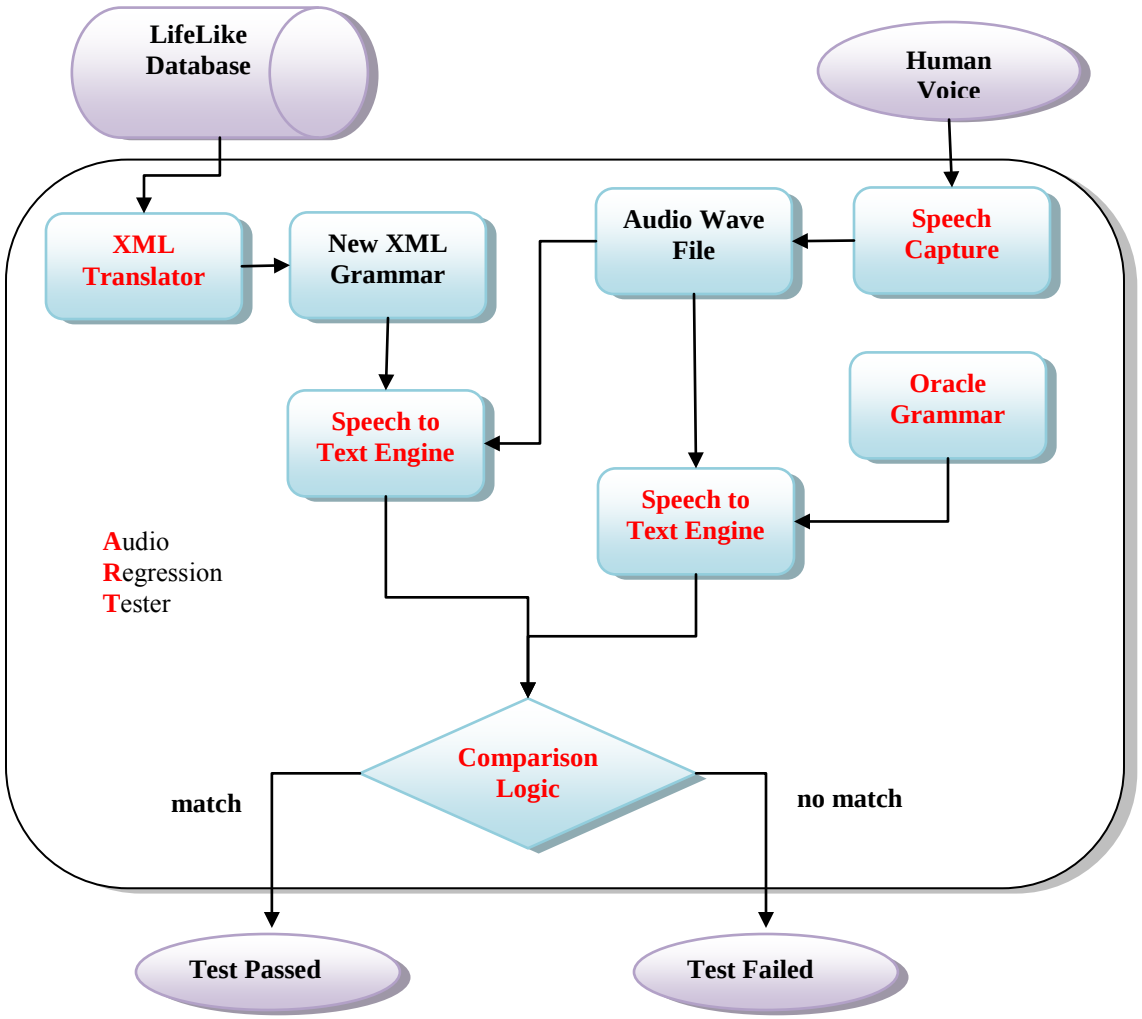


Figure 3: Audio Regression Tester (ART) System Diagram

The concept of *Audio Regression Tester (ART)* originated after elaborate and time-consuming manual testing and retesting during the early developmental phases of the speech recognition grammars in the LifeLike project. ART is proposed as a solution to automatically build and test the XML grammar file each time the knowledgebase changes. When data is added to the SQL knowledgebase, ART will be invoked to build a new XML grammar file and to run tests on it. Currently, the LikeLike system is capable of recording users' voices as wave files. Each response a user speaks is stored in a different sound file under a unique filename. This allows greater flexibility when playing the sound file against the grammar file. At any time, a user's recorded voice can be used to test the grammar file. The voice file will be played against the oracle grammar file with select context-specific rules being active in the grammar file. The LifeLike system already knows which context the response is contained in and therefore we can make that specific rule in the grammar file active while playing the corresponding sound file. As seen in Figure 3, the SRE will output a text string to display the result of the test i.e. whether there was a match between the output of the sound file and new grammar file and the sound file and the oracle grammar file or whether there was no match i.e. if there was proper recognition or not.

1.5 Challenges in Regression Testing with respect to Speech Recognition Grammars

The problem of regression testing can be broken down into the *test selection problem* and the *test plan update problem*. The *test selection problem* is concerned with the design and

selection of test cases to fully test a modified program (Leung & White, 1989). Some test cases can be reused from those in the existing test plan while new test cases might need to be created based on modifications in the program. This selective strategy can reduce the cost of retesting compared to the retest-all policy, which is designed to run all tests in the test bucket on the software system. However, a selective regression testing policy might not be cost effective if the effort made in test selection exceeds the cost of executing the extra test cases used by the re-test all policy. The *test plan update problem* is concerned with the management of a test plan as the software system is undergoing modification (Leung & White, 1989). Some tests will become obsolete and new tests cases will need to be created to test the modifications and new features of the software. This has also been referred to as the *coverage identification* problem (Rothermel & Harrold, 1997).

Since regression testing can account for as much as one-half the cost of software maintenance (Bezier, 1990) (Leung & White, 1989), it is essential to choose the correct tests for the speech recognition grammars while at the same time maintaining a comprehensive test plan. If a modification is made only to part of the XML grammar file e.g. a single or multiple phrases being added to a single rule, it is momentous that we design a test that will activate only that rule and run the relevant sound file(s) against the XML grammar file. If done manually, it would require someone to manually activate a specific rule in the grammar file, and then using a range of human speakers to exercise a

voice recognition program and microphone headset, to test each word in the rule to ascertain whether changes in the file have caused the quality of recognition of other phrases in the domain to be degraded. Since we have used only a single rule in this example, finding the changes in the file and undoing them might not be as difficult as compared to when there are multiple rules being modified in the XML grammar file. Perusing the file to find the additions or modifications becomes a tedious and perpetual task which ART aims to avoid.

Another challenge in regression testing with speech recognition grammars is the *test case selection policy*. Sound files that contain the recorded voices could be used by ART to test the XML grammar file. Yet, it is unclear which to select. The main consideration of voice recognition in LifeLike is the fact that the phoneme recognition system is speaker-independent. This means that SAPI strives to recognize any person who speaks the English language. While many persons speak what would qualitatively be referred to as proper English in terms of pronunciation, LifeLike operates in the more realistic environment which must consider persons with different accents who do not necessarily speak fluent English. This is an important consideration in building the test cases for ART since we will need to establish benchmarks will allow comparative testing. ART utilizes an ideal set is this respect since it has access to any of the pre-recorded sound files that contain different users' voice. This will allow it to readily compare persons with different accents against the grammar file to see how well their voice was deciphered by

the recognizer. These tests can possibly be used in the future to provide data which will allow LifeLike, to be more precise in interpreting non-traditional English speakers.

1.6 Contribution of Thesis

This thesis presents the development and analysis of a novel speech regression testing system called ART. ART is employed as part of the LifeLike project and the results presented confirms the hypothesis that manual testing of grammars costs an enormous amount of labor which inadvertently inhibits the amount of grammars that can be practically built and tested within a given development period and under practical budget constraints.

CHAPTER 2 introduces speech recognition and regression testing fields. This chapter gives an overview different speech recognition technology in use. It also shows the use of regression testing in software development.

CHAPTER 3 provides an in-depth view of the LifeLike system. It gives a comprehensive understanding of the different components that comprise LifeLike and the underlying technologies used in each as they relate to ART.

CHAPTER 4 describes in detail the software that was developed. This section investigates the design goals of ART along with other software that assist in the regression testing process.

CHAPTER 5 provides results obtained from ART and also shows the relationship between recorded voice and naturally spoken voice. A cost-benefit analysis is also presented to show the benefits of ART as compared to manual regression testing.

CHAPTER 6 discusses overall conclusions, and outlines topics for possible future work.

CHAPTER 2**: PREVIOUS WORK**

2.1 Speech Recognition

Speech recognition technology has been around for quite some time, but significant strides made by Baker in the 1970's sparked new interest in the field. Graduate students, James and Janet Baker became interested in speech technology while observing waveforms of speech on an oscilloscope at Rockefeller University in 1970. Technology at the time was only able to recognize a few hundred words of discrete speech, provided the system was trained on the speaker and the speaker paused between words. James Baker saw the waveforms and the problem of natural speech recognition as an interesting pattern-recognition problem. The Bakers moved to Carnegie Mellon University (CMU) and began working on natural speech recognition capabilities. While most speech researchers at that time were using contextual information to recognize spoken language, the Bakers took a different approach; their method was based purely on statistical relationships, such as the probability any two or three words would appear one after another in spoken English. This was where James Baker had introduced the Markov process in speech recognition. A Hidden Markov Model can be interpreted as a generator of vector sequences. It is a finite-state machine that changes state once every time unit, and each time, t, that a state, k, is entered, an acoustic speech vector, y_t, is generated with probability density $b_k(y_t)$ (Young, 1996).

Not too long after, in 1982, the Bakers formed their own company, Dragon Systems which led to today's popular product *Dragon Naturally Speaking* (Nuance Communications, 2008). Lernout & Hauspie (L&H), a Belgium-based speech recognition company formed in 1987, acquired Dragon Systems in 2000. In 2001, L&H went bankrupt and ScanSoft purchased the rights to Dragon products. ScanSoft merged with Nuance Communications in September, 2005 (Nuance Communications, 2005) with the combined entity being called Nuance Communications. Nuance continues to sell Dragon Systems under the name Dragon Naturally Speaking. Other companies such as IBM, with its ViaVoice (Embedded ViaVoice, 2008) product and Microsoft, with its SAPI (Microsoft Corporation, 2008) product, offer similar solutions as consumer products and system development kits for Speech Recognition applications.

2.1.1 Hidden Markov Models in Speech Recognition

The use of HMMs in speech recognition was initiated by James Baker with research done on the Dragon System (Baker, 1975). Since then several systems have employed the use of HMMs to aid in speech recognition. One notable system that has been under development for quite some time and claims to be able to do continuous speaker-independent speech recognition, is the SPHINX system developed at CMU (Lee, Hon, & Reddy, 1990). SPHINX is a system based on HMM with Linear Predictive Coding (LPC)-derived parameters that aims to tackle three major problems in speech recognition: speaker dependence, isolated words and small vocabulary. Research has shown that error

rates increased by 300-500 percent when a speaker-dependent systems is trained and allows to be used in the speaker-independent domain (Levinson, Rosenberg, & Flanagan, 1977) (Lowerre, 1977). This is the reason most speech recognition systems require a speaker to train the system before reasonable performance can be achieved. Based on results published from the SPHINX project, it was shown that large-vocabulary speaker-independent continuous speech recognition is feasible. This implies that there were drawbacks to developing such a system. Detailed models allowed the HMMs to perform better but needed considerable training to be successful (Lee, Hon, & Reddy, 1990). Some of the sophisticated modeling techniques used in this system helped to reduce the error rate from the baseline system by as much as 85% (Lee, Hon, & Reddy, 1990). SPHINX continues to be developed by the CMU Sphinx Group and is also an open source product.

Research done in BBN Laboratories produced a continuous speech recognition system called BYBLOS (Chow, et al., 1987). This system makes use of robust context-dependent phonetic models using HMMs. BYBLOS is composed of a signal processing frontend, dictionary, phonetic model training system, word model generator, grammar and decoder (Chow, et al., 1987). The parameters of the HMMs are automatically extracted from spoken speech and corresponding text transcription by the Baum-Welch (also known as the Forward-Backward) algorithm (Chow, et al., 1987). For the training of an utterance, the training system uses speech and text and builds a network of phonemes using the

dictionary. According to (Chow, et al., 1987) BYBLOS gives a word accuracy in the 98.5% range for speaker-dependent mode after 15 minutes of training; and, in speaker-adaptive mode, recognition rates of 97% is achieved after the HMM parameters are adapted to the new speaker.

Chung, DeMara and Moldovan (Chung, DeMara, & Moldovan, 1993) present a parallel computational model for the integration of speech and natural language processing. The model adopts a hierarchically-structured knowledge base and memory-based parsing techniques. Processing is carried out by passing multiple markers in parallel through the knowledge base. Speech-specific problems such as insertion, deletion, substitution, and word boundaries have been analyzed and their parallel solutions are provided. The complete system has been implemented on a parallel machine and is operational. Results show an 80% sentence recognition rate for the air traffic control domain. A 10-fold speed-up can be obtained over an identical sequential implementation with an increasing speed advantage as the size of the knowledge base grows (Chung, DeMara, & Moldovan, 1993).

Semantic Network Array Processor (SNAP) is a parallel architecture for knowledge representation and reasoning that uses the marker-propagation paradigm (DeMara & Moldovan, 1993). The primary application areas of SNAP are natural language understanding and speech processing. A first-generation SNAP-1 system has been

designed and constructed using an array of 144 digital signal processors organized as 32 multiprocessing clusters with dedicated communication units, a tiered synchronization scheme, and multiported memory network (DeMara & Moldovan, 1993) .

2.1.2 Augmenting Hidden Markov Models

Speech recognition technology does not understand the meaning of a sentence but merely converts utterances and matches them with words that together form a phoneme. According to (Lieberman, Faaborg, Daher, & Espinosa, 2005) acoustic analysis alone is not enough for accurate speech recognition. For example the two phrases "recognize speech using common sense" and "wreck a nice beach you sing calm incense," sound nearly identical but have completely different meanings. This is one of the underlying challenges facing any speech recognition technology of needing to distinguish between words and sentences that are phonetically the same yet contextually different. Most previous approaches used statistical language models based on techniques as Hidden Markov Models and *n-grams*. These models calculate the probability of each word in a vocabulary appearing next, based on the previous sequence of words. Research done by (Lieberman, Faaborg, Daher, & Espinosa, 2005) suggested the use of Commonsense Knowledge to solve the context problem with semantics, in addition to the statistical model. Their idea was to build a large semantic network of concepts, similar to WordNet (Fellbaum, 1998), that allows the understanding of relationships between concepts in

thousands of domains. This domain knowledge can be used by SREs to disambiguate phonetically similar phrases (Lieberman, Faaborg, Daher, & Espinosa, 2005).

There has been the use of visual speech cues to help improve speech recognition. Work done by (Dupont & Luettin, 2000) uses a visual module to track the lip movement of a user and extract relevant speech features. This is done with an appearance-based lip model that is learned from example images. Data is extracted from the curves in the lips and grey-level information of the mouth area. The visual speech information is represented in the form of shape and intensity parameters. It is argued that in noisy environments, phonemes that are hard to understand are easier to distinguish visually and vice versa (Dupont & Luettin, 2000). In this research, both the visual and acoustic modules are modeled using HMMs. This research produced results that attest to the fact that adding the visual speech components to acoustic-only systems provides better speech recognition and reduces the error rate in the presence of noise.

One of the latest pieces of research in this area is that done on the DARPA Global Autonomous Language Exploitation (GALE) program (Qin, et al., 2006). Some of the major components of the system are speech recognition, machine translation and question answering. The speech system is modeled by a five-state, left-to-right HMM with no skip states and is used partially to automatically transcribe Mandarin broadcast conversation to text (Qin, et al., 2006). The GALE project builds upon an earlier DARPA Effective

Affordable Reusable Speech-to-text (EARS) project which was used primarily to transcribe English telephone conversations (Chen, et al., 2006). The techniques used in EARS extend the general framework of HMMs and use Gaussian mixture models (GMMs) as output distribution (Chen, et al., 2006).

After years of research in speech recognition, little has changed since the introduction of HMMs. Most systems either use these statistical models alone or augment them to attain better speech recognition. Since this research isn't based on solving the problem of speech recognition, but augmenting available tools to attain better speech recognition, the next section will provide some insight into the commercial tools available, giving more emphasis to the ones used in this research.

2.1.3 Commercial-Off-The-Shelf (COTS) Tools for Speech Recognition

Some of the more popular COTS speech recognition tools available today include Nuance's Dragon Naturally Speaking (Nuance Communications, 2008), IBM's ViaVoice (Embedded ViaVoice, 2008), Nuance's VoCon 3200 (Nuance Communications, 2008) and Microsoft's Speech Application Programming Interface (SAPI) (Microsoft Corporation, 2008). Dragon Naturally Speaking version 10 boasts recognition rates as high as 99% and also claims to never make a spelling mistake. However, the Software Development Kit (SDK) for Nuance's speech products, Dragon Naturally Speaking and VoCon 3200, are not free and can be quite costly to acquire. ViaVoice while claiming to

provide superior speech recognition in multiple languages, multiple grammar formats, and text-to-speech (TTS) capabilities, is also commercially licensed and hence not free. Microsoft on the other hand, provides its Speech API free of charge. Based on the cost limitations posed by the other products, SAPI version 5.3 (Microsoft Corporation, 2008) was chosen as the most suitable SRE for this research. SAPI 5.3 supports the expression of SRGS as XML as well as it enables SRGS grammars to be annotated with semantic information. SAPI 5.3 is a derivative of the SAPI 5.0 family, in which applications and engines do not communicate directly, but rather through a special runtime Dynamic-Link Library (DLL). In the design phase of the LifeLike Recognizer, modularity was important and this prompted the addition of the Speech Recognizer Control (SRC) layer to the architecture. The SRC was implemented using Chant SpeechKit 5 (Chant Inc., 2008) and is shown below in Figure 4.

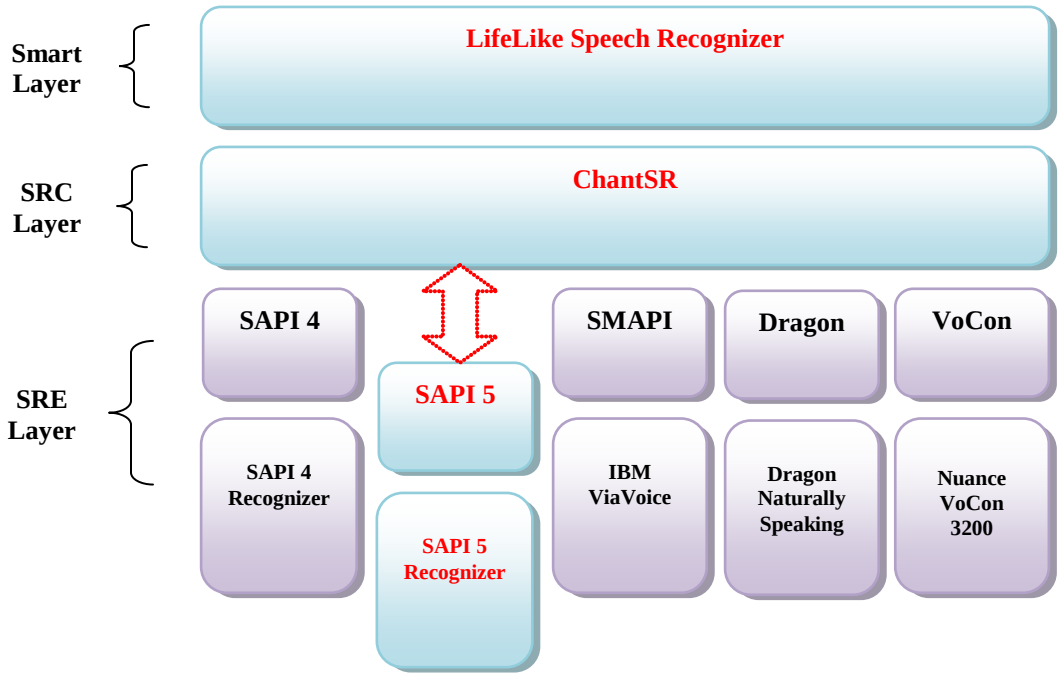


Figure 4: Layered Speech Recognition Architecture (DeMara, et al., 2008)

Figure 4 gives a high level understanding of the modularity encapsulated in the layered architecture model. Using the SRC layer gives flexibility to use any other recognizer should it become necessary.

2.2 Regression Testing

Regression testing is an important activity in software maintenance. It is the process of validating the modified parts of the software and ensuring that no new errors are introduced into previously tested code. Reference to software testing dates back to as early as 1950. According to (Hartmann & Robson, 1988), Miller's paper presented a citation to Turing indicating that 'testing is the empirical form of software quality

assurance, while proving is the theoretical way' (Miller, 1979). Systems involved in complex tasks such as speech recognition are mainly amenable to empirical analysis making this form of testing vitally important during product development.

2.2.1 Regression Testing Strategies

Leung and White describe two types of regression testing: *progressive regression testing* and *corrective regression testing* (Leung & White, 1989). Progressive regression testing involves a modified specification. When new data requirements are incorporated in a system, the specification will be modified to reflect these additions. It is usually employed at regular, fixed intervals. On the other hand, in corrective regression testing, the specification does not change. Some program instructions and design decisions are modified and test cases can be reused. Corrective regression testing is usually undertaken after corrective maintenance activities that can occur at any time and may be invoked at irregular intervals.

If there is a test bucket or test suite available at the time of regression testing, a decision has to be made whether to use all or some of the tests. This dilemma gives rise to two different testing strategies. The *retest-all* strategy reuses all tests in the suite but could waste computational resources and time if only minor changes were made to the system. The *selective* strategy uses some of the test cases and avoids wasteful overheads. The selective strategy is more economical than the retest-all strategy if the cost of selecting a

reduced subset of tests to run is less than the cost of running the tests that the selective policy allowed us to omit (Leung & White, A Cost Model to Compare Regression Test Strategies, 1991). The selective strategy was implemented in TestTube (Chen, Rosebblum, & Vo, 1994). In TestTube, Chen et al. made an analogy between selective recompilation, in the *make* and *nmake* tools, and selective regression testing. These tools employ a strategy whereby recompilation is done only on source files that have been modified or files that depend on modified files. In regression testing, a test unit must be rerun if and only if any of the program entities it covers has changed. The real challenge is to identify the dependency between a test unit and the program entities it covers.

The selective retest technique has been summarized by (Rothermel & Harrold, Analyzing Regression Test Selection Techniques, 1996) in the following steps:

- After modification, program P has become P'
- Select a subset of test cases T' from an existing test suite T to execute on P'
- Test P' with T' to establish the correctness of P' with respect to T'
- Examine the test results to identify failures
- Correct the failure by identifying the faults
- Create a new test suite for P' from test results

The steps mentioned above address four problems in regression testing: regression test selection, coverage identification problem, test suite execution problem, and test suite maintenance problem (Rothermel & Harrold, Analyzing Regression Test Selection Techniques, 1996).

According to (Leung & White, 1992) it is important to identify different levels of abstraction that regression testing should be applied. These include *unit testing*, *integration testing* and *system testing*.

2.2.1.1 Unit-Level Regression Testing

The selective regression testing strategy has also been used in unit testing. In particular, unit testing involves verifying that each individual module of a program is working properly. Automated unit regression testing using *make* was explained by (McCarthy, 1997). The method he proposed involves creating a test case for each module and writing targets in the makefile, indicating the test case's dependency on the module it tests. If the unit tests have passed the first time, *make* is run with the accepted target. This run produces reference copies or canon files of the test results (McCarthy, 1997). If changes are made in the future, a different target called the regress target is run to compare the new test outputs with the canon files and list the differences in what (McCarthy, 1997) refers to as a regression report. The report immediately shows what has changed. If the

changes are correct, they are accepted and new reference copies are made; otherwise, the problem is corrected and the regression tests are rerun.

Korel and Al-Yami (Korel & Al-Yami, 1998) proposed a method to do *corrective regression testing* at the unit level. In corrective regression testing, the specification is unchanged for a module; their idea is to find input data that generate different results when tested on the original and modified modules. If such data is found, it indicates an error because the input data is supposed to produce the same results for both modules. According to (Korel & Al-Yami, 1998) the likelihood of the error being in the changed module is very high since the original module was tested and previously used without problems.

One approach to *data flow regression testing* using slicing type algorithms was explained in (Gupta, Harrold, & Soffa, 1992). This approach explicitly detects definition-use pairs that are affected by a program change without the use of data flow history or the need to recompute the data flow for the entire program. Two slicing algorithms were used in this approach to directly ascertain the affected definition-use pairs; *BackwardWalk* and *ForwardWalk* (Gupta, Harrold, & Soffa, 1992). The *BackwardWalk* algorithm identifies the definitions of a set of variables *V* that reach a program point *P*. It then starts from that program point and does a backward traversal through the program for definitions of all variables in *U* (Gupta, Harrold, & Soffa, 1992). It ends when all definitions have been

countered along each path in the program. The *FowardWalk* algorithm starts from the same point *P* and works forward to find the uses and subsequent definition-uses which are affected by the change (Gupta, Harrold, & Soffa, 1992).

2.2.1.2 Integration-Level Regression Testing

Integration testing is the phase of software testing in which individual modules are combined and tested as a group. Integration testing follows unit testing and helps to detect failures that weren't discovered during unit testing.

The firewall concept developed by (Leung & White, 1992) (Leung & White, 1990) attempts to separate the modules that were affected by program changes from the rest of the code. The unchanged modules that interact with the modified ones are their direct ancestors and descendents and could also be part of the firewall. According to (Leung & White, 1990) all modules in four basis boundary cases they have defined must be included as modules within the firewall. The basis boundary cases for the firewall need to encompass program specification changes and code changes. Two of the boundary cases correspond to an unchanged module calling a modified module and the other two correspond to a modified module calling an unchanged module (Leung & White, 1992). Leung and White showed that the firewall concept reduced the amount of integration testing needed.

Regression test selection by the use of control flow graphs was introduced by (Rothermel & Harrold, A Safe, Efficient Regression Test Selection Technique, 1997). Control flow graphs (CFGs) are used to select tests from the test bucket. The algorithms handle either single modules or groups of modules and do not require prior knowledge of where code changes have been made. *SelectInterTests* is the algorithm used to create CFGs for both the original program P and its modified version P'. A test history table is created to keep track of which test cases were related to each traversed edge in the original program. *SelectTests2* is invoked on the entry procedures, P_E and P'_E of the two programs and if there are differences between the two CFGs, the corresponding test cases in the history table are selected (Li & Wahl, 1999).

2.2.1.3 System-level Regression Testing

System testing is testing applied to complete, integrated systems to evaluate the system's compliance to specification. System testing does not require knowledge of the inner design of the software components.

TestTube is a system that performs *selective* retesting of software written in the C language (Chen, Rosebblum, & Vo, 1994). It is used in system testing and identifies which subset of a test suite need to be used for retesting a new version of a system. TestTube works by splitting a software system into basic code entities, then monitoring the execution of each test unit and analyze its relationship with the system under test.

This allows TestTube to determine which subset of code entities the test unit covers. If there is a change in the system, the test unit that covers the entity that contains the change needs to be rerun. The system source code is instrumented by the Annotation Preprocessor (app) for C (Chen, Rosebblum, & Vo, 1994). A C program database is then built for each version of the system under test using the C information abstractor (CIA) (Chen, Rosebblum, & Vo, 1994). This database contains information about the system entities and entity dependency. If there are two versions of the program, TestTube analyzes the two corresponding databases and produces an entity difference list (Chen, Rosebblum, & Vo, 1994).

2.2.2 Regression Testing in the Object-Oriented Domain

The object-oriented (OO) paradigm for software development introduces new concepts such as encapsulation, inheritance and polymorphism all of which present unique problems in regression testing.

One approach to regression testing of object-oriented programs was presented by (Kung, Gao, Hsia, Toyoshima, & Chen, 1996). The regression test model used in this approach was developed to capture and represent complex relationships and interdependence between various parts of a C++ program at the class level. Three types of diagrams were used in this approach: object relation diagrams (ORD), block branch diagrams (BBD) and object state diagrams (OSD) (Kung, Gao, Hsia, Toyoshima, & Chen, 1996). The ORD

shows the inheritance, aggregation and association relationships between classes; the BBD allows understanding of member functions in a class and their relationship to other member functions and data items; and, the OSD is used to collect dynamic behavior of class objects (Kung, Gao, Hsia, Toyoshima, & Chen, 1996). When a change is made in an OO program, this change can propagate to different levels. This means that regression testing should be done at all the levels the change has affected. Class changes can be classified into class interface changes, class relation changes, object behavior changes and class member changes. A class firewall is used to identify the effects of a class change at the class level while the concept of *test order* was proposed as a test strategy for class unit retesting and class reintegration testing. (Kung, Gao, Hsia, Toyoshima, & Chen, 1996). This approach has showed promising results in realistic applications such as the InterViews class library (Kung, Gao, Hsia, Toyoshima, & Chen, 1996).

Rothermel and Harrold (Rothermel & Harrold, 1994) introduces a new selective retest policy for object-oriented software. Their approach builds on the concept of program dependence graphs (PDGs). PDGs encompass both control and data dependence (Rothermel & Harrold, 1994). Classes, unlike a program can have multiple entry points. This is as a result of classes having multiple public methods. To perform class testing, a driver is used to call different methods in the class in varying order. The PDG links all the driver programs together by selecting a root driver and adding edges to it from the public methods in the class (Rothermel & Harrold, 1994).

2.2.3 Regression Testing of Graphical User Interfaces

Graphical User Interface (GUI) testing is difficult since it involves many inputs, events and states. One other serious problem with testing GUIs is that the output can be graphical or may be an event. This means that if the maintenance engineer doesn't have sufficient knowledge of the GUI, and performs testing with the expectation of observing a fault and no visible change is seen, bugs can still be overlooked.

GUI interaction testing is one approach to this problem (White, 1996). This method seeks to test the pairwise interactions between all GUI objects and selections in an automatic and effective way. Two ways these interaction problems can arise are statically and/or dynamically (White, 1996). Static GUI interaction uses a single graphical screen whereas in dynamic GUI interaction a single action on one screen causes another screen to be brought up and the process can be repeated. Three algorithms were investigated in this approach: enumerate the elements of the interacting GUI objects (factors) and duplicate elements when necessary; generate the elements of each factor randomly, duplicating elements when needed; and, generate elements of each factor by using Mutually Orthogonal Latin Squares (White, 1996). If certain conditions are met, the algorithm based on the concept of Latin Squares results in the minimum number of tests generated (White, 1996).

More recently, a framework was developed to do frequent and automated retesting of GUIs. DART (Daily Automated Regression Tester) automates the entire testing process from structural GUI analysis, test case generation, test oracle creation, and code instrumentation to test execution, coverage evaluation, regeneration of test cases, and their re-execution (Memon, Banerjee, Hashmi, & Nagarajan, 2003). In the initial stages, DART performs *GUI ripping* by analyzing the Application Under Test (AUT) GUI structure, traversing each of the GUI's windows and identifies objects and their properties (Memon, Banerjee, Hashmi, & Nagarajan, 2003). These are then extracted and placed in an XML file. The GUI structure is used to create *event-flow graphs* and an *integration tree* which are used to create test cases and evaluate test coverage. The *event-flow graph* represents a top level action and the subsequent actions that can follow whereas the *integration tree* is constructed using the *event-flow graph* and shows the interaction among components (Memon, Banerjee, Hashmi, & Nagarajan, 2003). According to the authors, the effectiveness of DART will be studied by analyzing the number of faults detected.

CHAPTER 3**: LIFELIKE**

LifeLike is a system geared at developing an interactive avatar prototype of Dr. Alex Schwarzkopf who is a program director at NSF. It comprises various modules that seamlessly communicate visually, aurally and orally with a user. The three main modules are the Speech Recognizer (SR), the Dialog Manager (DM) and the Responsive Avatar Framework (RAF). The SR module is responsible for doing speech-to-text translation of a response provided by the user. The DM module uses the output from the SR along with its knowledge base to make sense if what was said. The RAF provides a life-like image of Dr. Schwarzkopf along with text-to-speech capabilities. The entire system is bound by a message passing communication protocol implemented using sockets. It helps to keep the modules of the system in synch by providing a channel for acknowledgement to and from each module. The remained of this chapter will be dedicated to giving an in-depth view of LifeLike as it formed the development system in which ART is tested.

3.1 Speech Recognizer Module

This module was designed using a layered model to allow greater flexibility so as to support various COTS engines. Figure 4 provides a visual understanding of the layers that make up SR architecture i.e. the SRC, SRE and Smart Layer. The SRC provides functionality to allow compatible COTS recognition engines to be used. The SRE is currently implemented using SAPI version 5.3 but can be replaced by other SREs made

by Dragon (Nuance Communications, 2008), Nuance (Nuance Communications, 2008) and IBM (Embedded ViaVoice, 2008).

The SRE uses the incoming audio signal captured by the audio capture device and compares it with its standard dictionary using the stochastic process of Markov Chains (Juang & Rabiner, 1991). This process matches each phoneme to the most probably text match for a particular language. Markov Chains finds the future states probabilistically. Therefore, the SRE can predict which word to match to a phoneme based on the probability of the word being in a particular sentence given certain rules.

The SRC layer is implemented using Chant SpeechKit 5 (Chant Inc., 2008). This allows the use of SREs from different vendors enabling code portability and reduced developmental time. Although Chant supports a wide array of programming languages, C# was chosen as the language of choice for the development of the SR due to its simple, modern, general-purpose, object-oriented nature.

The Dialog Manager also acts as a communication server to which the SR is connected via TCP/IP (Transmission Control Protocol/Internet Protocol). The DM sends a "microphone on" signal to the SR indicating that the SR should listen for a response from the user. After the user has made one (or more) utterance(s), the underlying SR automatically shuts the microphone off and sends the recognized event to the DM. The

microphone is turned off at this point to avoid spurious noise from interfering with recognition events.

Activation of the SR invokes the primary recognition strategy which is grammar-based. The grammars are grouped in the XML file by specific top-level rules which are synonymous with a specific context in this domain. The DM indicates through the socket which rule to make active for a particular instance of recognition. This provides the recognizer with only a small subset of active grammars which reduces the likelihood of confusion encountered in the recognizer if all rules were simultaneously made active in the XML file. A secondary backup strategy employed in the SR is dictation or free-speech mode. This has been employed as a secondary approach to parsing which is performed in parallel. The dictation results attempts to provide a failsafe backup during recognition that can be used if there was no recognition achieved the primary grammar-based approach. Even though it might not be as accurate as grammars, it still helps in somewhat identifying what the user was trying to say. One scenario of this is having an active grammar rule containing the phrase “director”; and, when the user speaks into the microphone the recognizer does not pick up “director” but rather “directing”. This is where the free-speech mode is automatically activated and outputs “directing”. Although it is not an exact match to “director”, it helps to provide some data to the DM which can ultimately be used to understand what the user was trying to convey. Table 1 gives a few more examples of these cases.

Table 1: Examples of Utterances and their corresponding output

Utterance	Grammar Result	Dictation Result
director	director	directing
evaluator	evaluator	evaluate
University of Central Florida	University of Central Florida	University of Central flower
University of Texas at Austin	University of Texas at Austin	University of Texas at dustin
Micheal Lovell	Micheal Lovell	Micheal Powell
Anjan Bose	Anjan Bose	Anjan hose

The grammar rules and phrases used in the grammar mode are automatically generated from a relational database that facilitates dialog development, maintenance and portability. New speech information for any domain can be added to the database and functions can be invoked to create a new grammar file. This automatic file creation and the use of an XML file allow the recognition engine to quickly switch between different recognition domains. Along with the creation of grammars, the SR posses the ability to record and store the user's vocal output in a wave file. These wave files can then be used to do numerous tasks such as regression testing and reconstruction of the conversation.

3.2 Dialog Manager Module

The dialog manager is the primary controller module of the LifeLike system. This module receives text streams from the Speech Recognizer module which helps it to determine how to react to contextual shifts. The DM also coordinates communication between the modules. The DM is tasked with making sense of what was decoded by the SR. It does this by using data stored in the database along with the context-specific text string(s) passed from the SR.

3.2.1 Natural Language Processing (NLP)

Natural Language Processing refers to a branch of Artificial Intelligence (AI) where a human agent interfaces a machine in his own native tongue. This interaction can be in the form of text based entry or spoken word speech input. There are four major issues associated with NLP; two of which will be explained below: linguistic systems and knowledge representation structures (Wilks, 2005).

3.2.1.1 Linguistic Systems

Linguistic Systems are those systems which interpret user input at the grammatical level. These systems have what is known as a parser, used to interpret a human's intent. Lieberman et al (2005) mention how difficult it is for a system to understand and disambiguate two *phonetically* similar sentences with different *semantic* print. They exemplified this claim with the use of "wreck a nice beach" as a homonym for "recognize

speech." Although these phrases sound alike, they have a completely different meaning. Speech recognition systems utilize HMMs to correctly decode each correctly pronounced word, but do nothing to interpret the semantic meaning of the sentence. Interpreting the series of words is challenging and resolving these ambiguities often require the use of contextual cues to constrain the number of possible matching words for the user's utterance.

Syntactic confusion occurs when parts of sentences can be interpreted in an array of permutations. For example, "the man drank coffee with a straw" can be interpreted as a man drinking a beverage with the aid of a straw, or it could be understood as a man drinking a particular cup of coffee that contained a plastic straw. Nevertheless, the sentence is confusing and requires additional information to interpret the semantic meaning. This kind of ambiguity often causes confusion in human minds, and understandably presents difficulty to automated systems. Once again, contextual recognition remains paramount in maintaining conversational cohesion. Combining knowledge of the current state of the environment with the current conversation goes a long way in resolving these syntactic ambiguities.

Semantic ambiguity refers to situations where sentence parts may be understood in multiple ways. These types of ambiguities can as a results of homophones; i.e. words that are pronounced the same but are different in meaning regardless of if they are spelled the

same or not. For example, “pitcher” can create bewilderment for any system. An instance of this type of confusion occurs in the following: “The cycle has stopped.” In this sentence, the semantic intent of ‘cycle’ is in question since it could mean either a bicycle or a recurring sequence. Since there is no real justification in choosing either meaning, it is once again necessary to be equipped with the context related to the current state of affairs.

Creating linguistic systems based on NLP systems is at best inaccurate and ambiguities can be resolved with a good grasp of the situational context associated with the linguistic utterances.

3.2.1.2 Knowledge Representation

Knowledge representation is crucial to resolve ambiguities. Wilks (2005) mentions that language has been viewed as a trivial issue once knowledge is established in a proper representation. Traditionally, this knowledge representation is expressed in logic-based systems. Knowledge modeling is concerned about the storing and processing of information so that computer programs can interpret this knowledge to aid, in this case, with speech recognition. Wilks (2005) mentions three viewpoints on the relationship of language and logic statement. The first dictates that logic inferences must be derived from conversation. Instead of parsing a sentence for its face value, the meaning of the utterance may have logical attachments that must be inferred from a back-end knowledge

model. The second viewpoint maintains meaning can exist outside logic. This essentially assumes some sort of association between words that is not established using logic alone. Lastly, the third viewpoint says that both logic and language suffer from the same problem of ambiguity. Since knowledge representation is usually expressed in logic-based syntax, creating a predicate logic rule-base allows easy sentence formation by reading off each individual rule. This concept aligns itself well with LifeLike which strives to have a natural conversation with the user.

3.2.2 Dialog Manager Architecture

Conversational goal management is achieved using a context-based approach (DeMara, et al., 2008). A context refers to a situation refers to a particular situation that is dictated by the configuration of internal and external circumstances such as the internal state of the conversation agent and the perceived state of the human trainee. A goal condition is associated with ever context and a group of relevant actions that can be executed to achieve this condition. A *goal condition* is defined as an end state that an agent desires to reach to impart specific knowledge to the trainee (DeMara, et al., 2008).

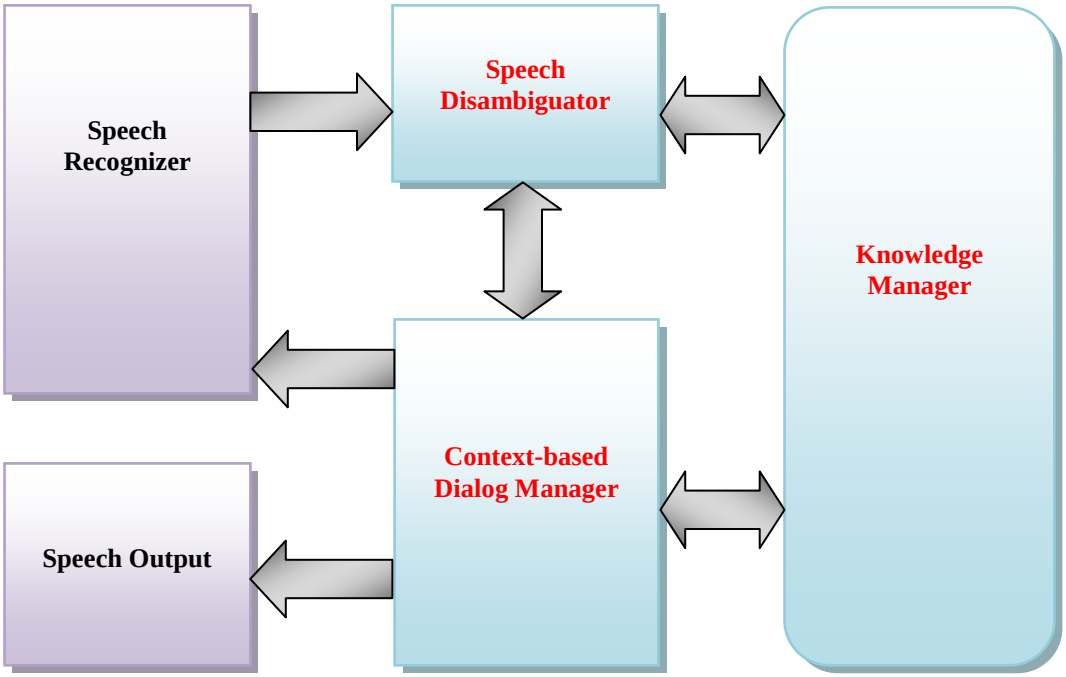


Figure 5: LifeLike Dialog Manager Architecture (DeMara, et al., 2008)

It is critical that conversational goals are properly handled by the dialog system since the user can have multiple goals or introduce new goals at any time. This means that the system must be able to service multiple goals simultaneously while at the same time be able to take on new goals, unannounced. This ability to alternate between goals in real-time lends itself to the *Context-Based Reasoning (CxBR)* used by Stensrud, Barrett, Trinh and Gonzalez (Stensrud, Barrett, Trinh, & Gonzalez, 2004). CxBR agents provide responses that are directly related to active content. The fact that contexts correspond to accomplishing particular goals combined with the idea that conversational goals take on a very fluid nature, yields the assertion that goal management can be facilitated with CxBR methods.

Figure 5 shows the architecture of the Dialog Manager, which is made up of three components: the Speech Disambiguator, the Knowledge Manager and the CxBR Dialog Manager. The Semantic Disambiguator serves as a listening comprehension filter. It uses the input from the SR and converts it to conversationally-relevant content to be processed by the person, known as the Disambiguated Input (DeMara, et al., 2008). The Knowledge Manager acts as a person's rote memory. The Speech Disambiguator along with the CxBR Dialog Manager send keyword-based requests to it and the Knowledge Manager outputs relevant information in the form of a contextualized data base (DeMara, et al., 2008). The Dialog Manager facilities the output of comprehensive responses to the Speech Output system. These responses are formulated by input from the Speech Disambiguator along with its own internal context-based mechanisms.

Goal management in the LifeLike DM comprises *goal recognition*, *goal bookkeeping* and *context topology* (DeMara, et al., 2008). Goal recognition refers to the process of analyzing user input utterances to determine the proper conversational goal that is to be addressed. This is somewhat similar to the context activation process in CxBR methods. Goal bookkeeping incorporates keeping track and servicing identified goals in an ordered fashion. Immediately after a goal is recognized, it is placed on a goal stack. Context topology refers to the entire set of speech acts of the conversation agent (DeMara, et al., 2008). This structure also includes the transitional actions when moving between contexts

when a goal shift is detected. The context topology, upon receiving the activated goal to be addressed from the goal stack, operates on this signal to provide the proper agent response. This in effects helps to clear out the goal bookkeeping stack. Goal recognition is accomplished using linguistic analysis of each user utterance. This is similar to the context activation process in CxBR methods where conditioned predicate logic rules determine the active context according to the state of the environment. The difference with the goal recognizer, however, is that the context is resolved using keywords and phrases that are extracted from the parts-of-speech parsing of input responses. By using a contextually-organized knowledge base, the user utterance is interpreted, and the context associated with this understanding is activated.

In order to facilitate the communication between modules, a customized protocol based on a message passing algorithm using sockets was created. A series of messages from the DM control the synchronization and operation of the different modules. Below, in Figure 6, is an example of a generic message that the DM would send to the SR indicating the activation of Context 1 and Context 2. These contexts are synonymous with rules in the grammar XML files.

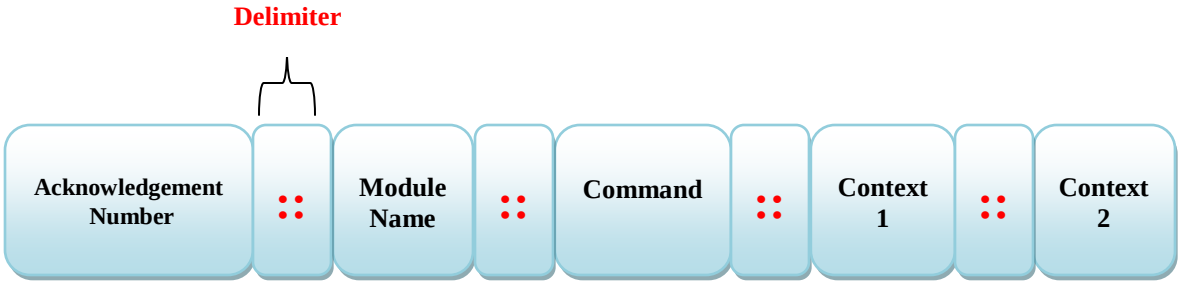


Figure 6: LifeLike Procotol Frame sent from DM to SR

It is possible to have one or more double-semicolon-delimited contexts in the frame depending on which stage of the dialog the system is in. The acknowledgement number is used by the DM to ensure a module had received the message sent to it and also to coordinate the synchronization of the entire system. The "module name" field contains the name of the module for which the frame was intended. Different modules require different operation-dependent frames but every frame used in the LifeLike domain contains a common header consisting of the acknowledge number, module name and command.

3.3 Responsive Avatar Framework

Creating a realistic active digital representation of particular human being is a challenging and multifaceted task (DeMara, et al., 2008). Investigations were conducted to identify and evaluate the interoperability of COTS packages for facial modeling, rendering of real-time graphics, motion-capture, and text-to-speech synthesis. The result was a customized Graphical Asset Production Pipeline which encapsulates the tasks

needed to create a visual representation of a human character (DeMara, et al., 2008). Furthermore, the options and best practices for recording vocal mannerisms and non-verbal mannerisms were evaluated and identified.

FaceGen, used by Heinrichs, Müller, Tewes and Würtz (2006), was incorporated into this framework. FaceGen is a tool used to generate three-dimensional (3D) head and face models using frontal and side photographic images. It provides a neutral face model that can be controlled parametrically to emulate almost any facial expression (DeMara, et al., 2008). FaceGen also enables a wide range of control over features of the model including age, race and gender. While this is initially sufficient, more advanced techniques such as modeling the sub-surface light scattering properties of the skin tissue can be done to improve realism (Donner & Jensen, 2005).

To enable motion capture, a new motion capture system equipped with eight high resolution infrared tracking cameras was used. Motion capture is the widely used in the film and video game industries for acquiring realistic human figure animation. A series of simple motions are recorded and the avatar is used allowed to “re-enact” them. This motion capture data can also be manipulated in real-time to allow more naturalistic behaviors of the avatar.

The Object-oriented Graphics Rendering Engine (OGRE) was used to render the real-time graphics of the avatar. OGRE provides a high-level interface for working with graphical objects as well as provides low-level shader control functions to create specialized visual effects to aid in building more realistic avatars. Text-to-speech synthesis was afforded by Microsoft's SAPI version 5.1 (Microsoft Corporation, 2008). SAPI 5.1 provides an event generation mechanism that reports the status of a phoneme or word change during the synthesis of voice in real-time. These events are used to provide real-time lip synchronization.

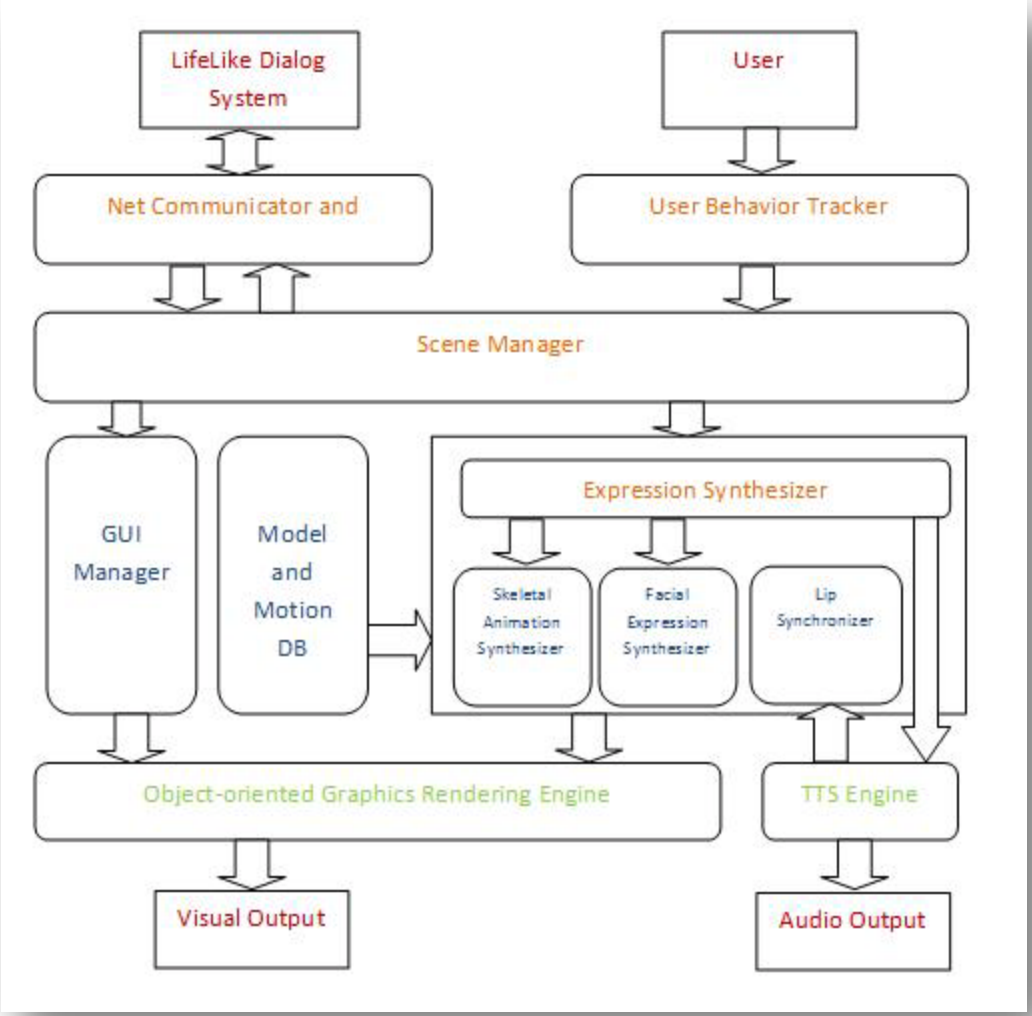


Figure 7: LifeLike Responsive Avatar Framework (DeMara, et al., 2008)

Numerous commercial speech systems provide an interface to SAPI 5.1 which allows applications to transparently leverage a multitude of speech systems. Figure 7 depicts the LifeLike Responsive Avatar Framework (RAF) which controls the avatar and provides connectivity to the SR and DM. The RAF is responsible for the avatar's operation to create a realistic representation that is capable of speech input, provides locomotion and a

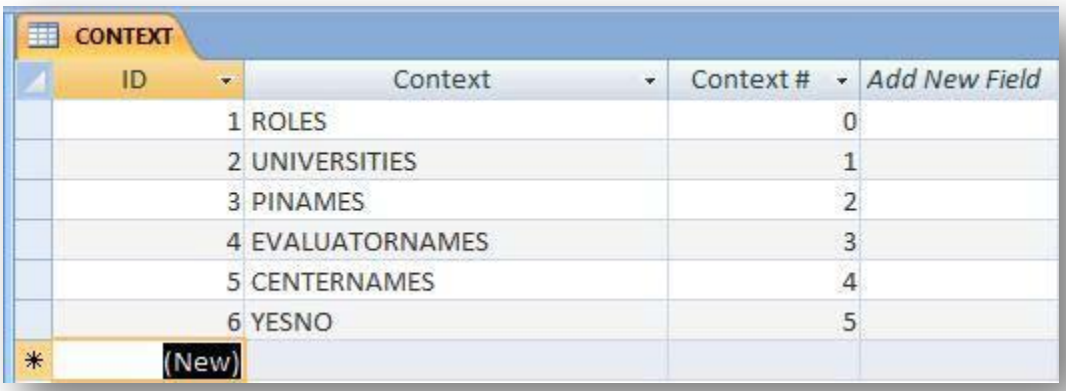
vocal response. The RAF has two separate sources of input: the Dialog Manager and user's behavioral information, such as eye-gaze. The DM provides whole sentences or phrases which are intended to be spoken by the avatar. These sentences will eventually contain tagged information which can be displayed on a whiteboard in the avatar space and also which relate behavioral information to the avatar. Eye-gaze tracking was done by the use of retro-reflective markers on a headband and an infrared camera.

The most significant component of the RAF is the Expression Synthesizer. It uses the 3D models and applies motion-capture data to produce a sequence of facial and body animations that fit the context of what is being spoken. Three major components of the Expression Synthesizer are: the Skeletal Animation Synthesizer, the Facial Expression Synthesizer and the Lip Synchronizer. Research is ongoing to achieve better control of the animations using complicated algorithms. One example of this is the motion-capture skeletal animations can be exaggerated or attenuated based on emotional changes in the avatar.

3.4 LifeLike Database

Currently, LifeLike-related data is stored in a Microsoft Access database. This includes contextualized knowledge along with data necessary to populate the XML grammar file. This database contains three main tables: the Context table, the University Names table and the Center Contact table.

The Context table contains agreed-upon context-related words that are used as the rule name in the grammar XML file. Each word has a numerical valued associated with it. This value is passed from the DM to the SR during recognition to activate specific grammars in the XML file. Figure 8 below shows the different Contexts with their associated numerical values.



CONTEXT

ID	Context	Context #	Add New Field
1	ROLES	0	
2	UNIVERSITIES	1	
3	PINAMES	2	
4	EVALUATORNAMES	3	
5	CENTERNAMES	4	
6	YESNO	5	
(New)			

Figure 8: Screenshot showing LifeLike's Context table

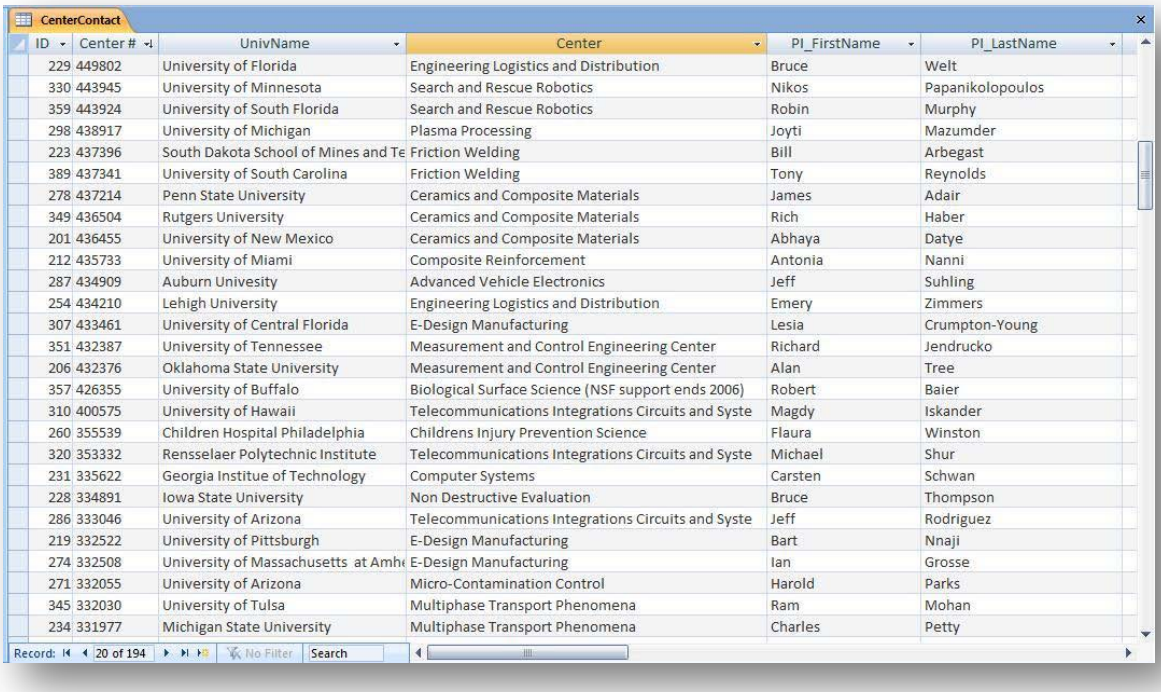
Each module in the LifeLike domain has shared access to this table to allow for synchronization of the entire system. Of the six contexts showing in Figure 8, only four are used, context number 0, 1, 2 and 5.

The University Names table contains the names and acronyms of approximately one hundred and sixty eight universities in the USA. Each university has one or more centers

and each center has a director associated with it. Currently, there are about seventy two different centers. The names of each university is extracted, manipulated and written the XML grammars file. Manipulation of the university name allows the addition of special characters before or after each word in the university name so as to facilitate recognition of a wider variety of responses with only a single line being written to the XML grammars file. An example of this is "Washington State University" which is commonly referred to as "Washington State". Instead of writing two lines in the XML grammar file to accommodate either response from the user, this university name is written as "Washington State ?University." The inclusion of the symbol before the word "University" indicates that this word is optional. University acronyms are also read from this table and written to the XML grammar file under a different rule, without any manipulation. When the numerical context for Universities is sent from the DM to the SR, both rules are simultaneously activated.

The Center Contact table contains the names of the current directors of the different centers at each university. Multiple schemes were devised and tested to include these names in the XML grammars file in order to achieve optimal name recognition. Effectively identifying the user of the LifeLike system is important because this holds the key to creating the perception of a life-like conversation. The first and last names were split into two separate rules since, when a person is asked his name he replies with either his first or last name but rarely both. However, systems are in place to recognize

complete names. When the SR correctly recognizes a name, it passes the name to the DM which then uses the database to perform search queries to retrieve information about the user. If a user responds to the system with only his first name, and there are multiple persons with the same first name from different centers, the avatar then asks the user which university he is from in order to discern his true identity. The Center Contact table is also used to form relations between different directors of related centers at different universities. This way the avatar is allowed to introduce directors with the same interest to foster new relations into new research ideas or work collaboratively on a single idea.



CenterContact

ID	Center #	UnivName	Center	PI_FirstName	PI_LastName
229	449802	University of Florida	Engineering Logistics and Distribution	Bruce	Welt
330	443945	University of Minnesota	Search and Rescue Robotics	Nikos	Papanikolopoulos
359	443924	University of South Florida	Search and Rescue Robotics	Robin	Murphy
298	438917	University of Michigan	Plasma Processing	Joyti	Mazumder
223	437396	South Dakota School of Mines and Te	Friction Welding	Bill	Arbegast
389	437341	University of South Carolina	Friction Welding	Tony	Reynolds
278	437214	Penn State University	Ceramics and Composite Materials	James	Adair
349	436504	Rutgers University	Ceramics and Composite Materials	Rich	Haber
201	436455	University of New Mexico	Ceramics and Composite Materials	Abhaya	Datye
212	435733	University of Miami	Composite Reinforcement	Antonia	Nanni
287	434909	Auburn Univesity	Advanced Vehicle Electronics	Jeff	Suhling
254	434210	Lehigh University	Engineering Logistics and Distribution	Emery	Zimmers
307	433461	University of Central Florida	E-Design Manufacturing	Lesia	Crumpton-Young
351	432387	University of Tennessee	Measurement and Control Engineering Center	Richard	Jendrucko
206	432376	Oklahoma State University	Measurement and Control Engineering Center	Alan	Tree
357	426355	University of Buffalo	Biological Surface Science (NSF support ends 2006)	Robert	Baier
310	400575	University of Hawaii	Telecommunications Integrations Circuits and Syste	Magdy	Iskander
260	355539	Children Hospital Philadelphia	Childrens Injury Prevention Science	Flaura	Winston
320	353332	Rensselaer Polytechnic Institute	Telecommunications Integrations Circuits and Syste	Michael	Shur
231	335622	Georgia Institue of Technology	Computer Systems	Carsten	Schwan
228	334891	Iowa State University	Non Destructive Evaluation	Bruce	Thompson
286	333046	University of Arizona	Telecommunications Integrations Circuits and Syste	Jeff	Rodriguez
219	332522	University of Pittsburgh	E-Design Manufacturing	Bart	Nnaji
274	332508	University of Massachusetts at Amh	E-Design Manufacturing	Ian	Grosse
271	332055	University of Arizona	Micro-Contamination Control	Harold	Parks
345	332030	University of Tulsa	Multiphase Transport Phenomena	Ram	Mohan
234	331977	Michigan State University	Multiphase Transport Phenomena	Charles	Petty

Record: 20 of 194 No Filter Search

Figure 9: Screenshot showing a portion of LifeLike's Center Contact table

Figure 9 above shows a few records of the Center Contact table. As can be seen in the diagram there are multiple centers with the same name but located at different universities. These ambiguities must be dealt with using grammars and present unique regression testing challenges.

CHAPTER 4**: AUDIO REGRESSION TESTER**

The *Audio Regression Tester (ART)* is a standalone automated regression testing interface that has used LifeLike as a testbed to develop and demonstrate its capabilities. ART, in addition to running regression tests, allows a user to record and store voice clips. One of its most signifgicant features is that it also automatically creates new XML files from the LifeLike database. Oracle speech-to-text files, that contain the results of testing against the old grammar file, will be created. Once this has been done and the voice files recorded and put in place, regression testing can begin. Since we have already established the oracles and know that recognition performs well with the current XML grammar file, we can now assume that new data has been added to the database. New data being added to the database necessarily means that a new grammar XML file needs to be created. ART will then be used to test the previously recorded voice samples against the new XML grammar file. The results are compared to the oracle speech-to-text files and if there is mismatch in the comparison, ART's built in file comparator is invoked to show the differences between the two XML files. These differences necessarily have caused recognition to deteriorate.

4.1 ART System Components

ART consists of three modules that operate independent of each other; the XML translator module, the audio capture module and test sequence module. The user is

afforded the privilege of selecting a LifeLike database from which to read data in order to create a new XML file.

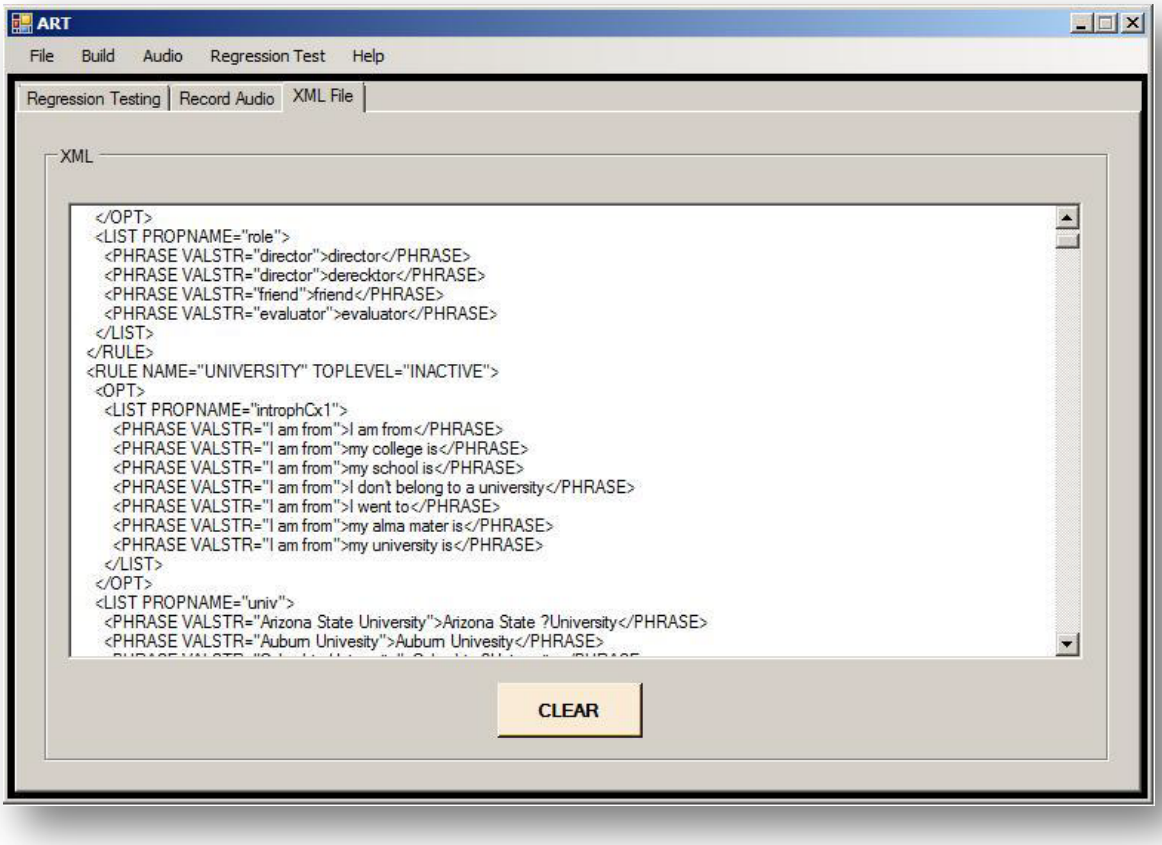


Figure 10: Screenshot of ART showing a newly created grammar XML file

After selecting a database and it has been successfully loaded, a message will be shown indicating that the process was successful and the disabled “Create XML” menu item under the “Build” menu will be enabled. When the user clicks on “Create XML,” the program then connects to the database and selects data from the relevant tables in order to build an XML grammar file. Special grammar markup tags are written to the file along

with the information from the database. Upon creation, the XML file is loaded in the text window of the XML translator as shown in Figure 10 at which time a message box pops up, indicating that the file was successfully created. This window affords a scrollable view of the newly created XML grammars file. The user is not allowed to manually edit this XML file with ART since doing so defeats the concept of automation. The user can now either record audio or perform regression tests with pre-recorded audio.

Audio capture is based on Chant's SpeechKit 5 (Chant Inc., 2008) and Microsoft's SAPI 5 (Microsoft Corporation, 2008). SpeechKit 5 provides the capability to record what the user is saying and store it as a wave file. After loading the appropriate grammar file, the check boxes corresponding to the rules in the XML grammars and the "START RECORDING" button, are enabled. Using the check boxes, grammar rules can be selectively activated. When the user clicks on any of the check boxes, the already selected XML grammar file is compiled to ensure it adheres to the SRGS before it is actually used. If the compile process has failed, an error message is displayed in the text window on the audio capture pane. This error is an indication that the integrity of the data in the database might have been compromised. Certain special characters are not allowed in the XML file; one such character is the ampersand ("&"). If however, the file has been compiled properly and is ready for use, a message indicating same will be presented to the user at which time he can begin recording by clicking the "START RECORDING" button. When recording begins, the speech recognizer will simultaneously output

recognized speech events as the user is speaking. There are two modes of speech recognizing: free speech and grammar-based speech. Free speech is automatically activated and aids grammar-based speech recognition when the user deviates from the phrases contained in the XML grammar file. The output wave file will however contain anything the user says whether it was recognized as regular or grammar constrained speech. The user can click “STOP RECORDING” to stop the actual audio capture and create a wave file with the recorded data. The filename of the saved file is stamped with the current month, day, year, and time. Audio recording can be performed with a single phrase or multiple phrases i.e. the user is allowed to store a single utterance per file or store multiple utterances in the same file. In either case, the regression tester will be able to use the file to output all recognized events.

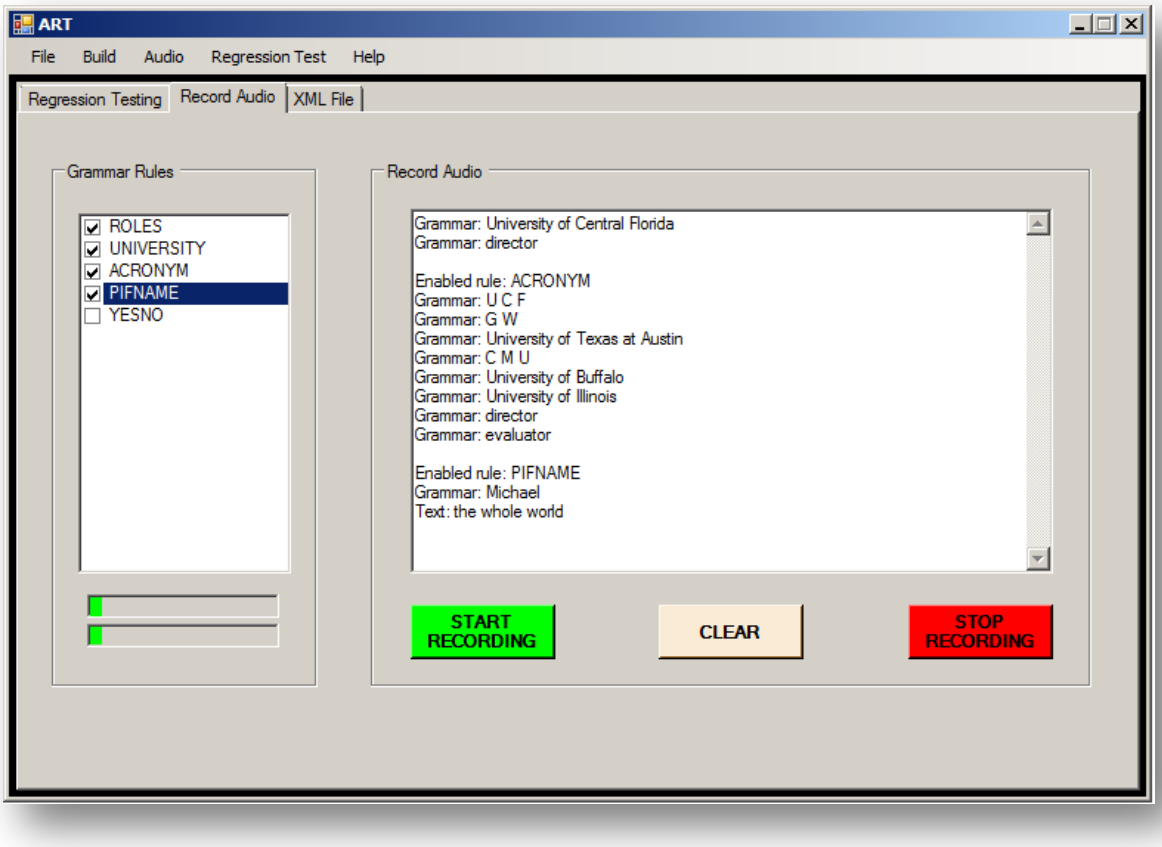


Figure 11: Screenshot of ART showing the Audio Capture Window

Figure 11 shows the audio capture window in ART. The "Grammar Rules" group box contains several check boxes whose names correspond to different rules in the currently loaded XML grammar file. As the rule are enabled or disabled (by selecting or deselecting the checkboxes), a message indicating this is shown in the text window of the audio capture pane. Figure 11 shows that the "ROLES" and "UNIVERSITY" rules were initially selected and some recording was performed. Later on, the "ACRONYM" and "PIFNAME" rules were enabled. This indicates that the speech recognizer will enable these four rules in the XML grammar file and try to constrain any utterance match to the

words contained in these rules. If a suitable match is found, it is output; otherwise the fail-safe free speech mode is automatically activated and tries to recognize what the user had said. In Figure 11, the initial two recognized words were "University of Central Florida" and "director," each of which are contained in separate grammar rules in the XML file. The last utterance detected is not part of the XML grammars and was recognized as free speech. Below the check boxes in the "Grammar Rule" group box, is a volume meter. The meter measures the audio level from the sound card to give the user an indication that ART is receiving signals adequately from the sound capture device. The volume meter uses a small buffer which it periodically queries to receive the latest samples in order to update the progress bar values.

The third and most important module is the test sequence. After creating a new XML grammar file, and recording wave files (or using pre-recorded audio files), the user can proceed with regression testing. Before performing tests, the user is required to load into the program the appropriate wave file and XML grammar file that he needs to do testing on. After this has been completed, messages will be displayed on the regression test window indicating whether the operation was successful or not. If the two load operations are successful, the "RUN TEST" button is enabled. Figure 12 below is a screenshot of ART showing the files being successfully loaded along with two grammar rules being enabled.

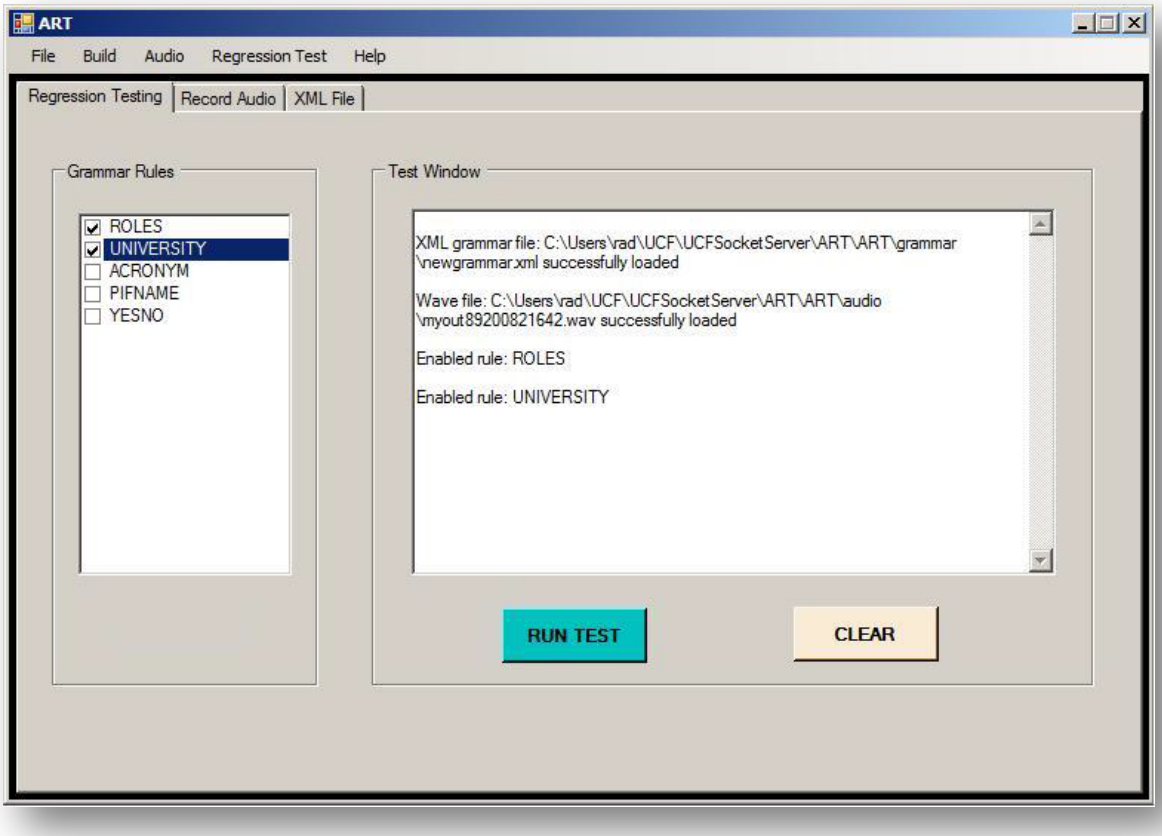


Figure 12: Screenshot of ART showing the Regression Testing pane

At this point the user can begin running tests. When the "RUN TEST" button is clicked, the primary playback function is invoked and the sound file is played back through the system with the grammar-based speech recognition mode enabled via the rules in the checkbox. Upon completion, the test could either pass or fail.

4.2 ART Operation

If the test has passed then a message is displayed informing the user of the success of the test. Since a passed test means that the output results using a new grammar file matches

the output results when the old file was used, the user is prompted to discard the original grammar file and use the new XML file as the grammar oracle. This essentially involves deleting the old XML grammar file and renaming the newly created file. The user is allowed to run more tests before eventually deciding to use the new XML file as the grammar oracle.

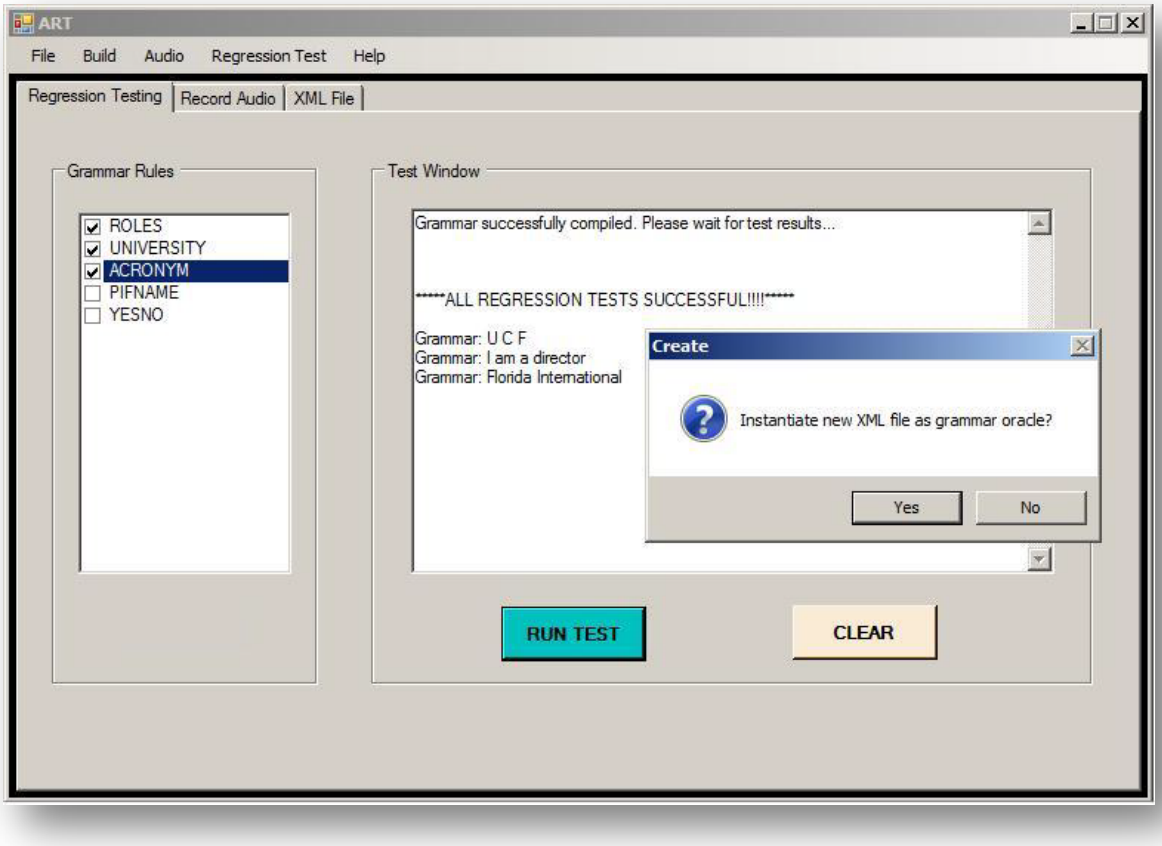


Figure 13: Screenshot of a successfully run test in ART

Figure 13 shows a successful regression test. The user can either choose “No” in the message box to run further tests with the new grammar file and different audio files; or, choose “Yes” to use the new file as the grammar oracle.

If the test has failed, a new pane, the “Result Pane,” is created and both the original grammar file and the newly created grammar file are shown side by side in scrollable windows. A line-by-line comparison is done between the two files and the difference is highlighted. It can be argued that this difference caused recognition to deteriorate or ultimately fail.

The results of the line by line comparison are color coded to allow for easy interpretation. If a line is highlighted ‘red’ in the source file and ‘grey’ in the destination file it means that the specific line is present in the source but not in the destination. This is an indication that some database records were removed. If a line is highlighted ‘red’ in the source file but ‘green’ in the destination file it can be interpreted that the lines correspond but there are changes within the line. A ‘green’ highlight in the destination with a corresponding ‘grey’ highlight in the source indicates that lines are missing in the source. These results indicate that records were added to the database which is why the destination output XML file contained extra records not found in the original oracle grammar. Table 2 summarizes these color codes with their meaning

Table 2: Test Discrepancy

Source Color/Destination Color	Interpretation of Discrepancy
Red/Grey	Line present in source but not in destination
Red/Green	Line present but contain changes in destination
Grey/Green	Line present in destination but not in source

Figure 14 below shows ART's comparison window. The oracle grammar is loaded in the left pane while the newly created grammar file is presented in the right pane. These scrollable panes allow the user to easily navigate through the XML files and when a real comparison is done, to see the difference in the output. If there is a difference in the output, ART will color code the lines as explained above to allow the user an easy view of the errors.

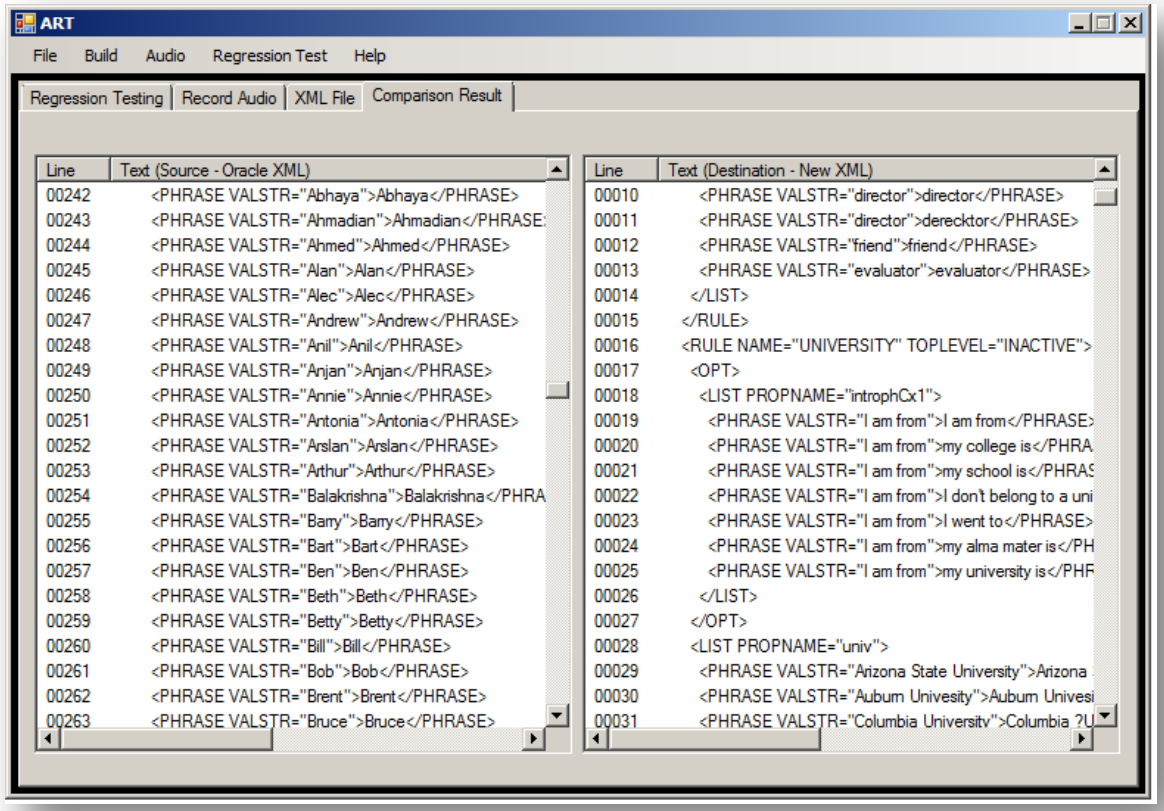


Figure 14: ART's Comparison Window

CHAPTER 5**: TESTING AND EVALUATION**

5.1 Overview

Three users were allowed to use the system and record their voices with the audio capture features found in ART. Using these results, the correlation between the speech-to-text results obtained by using the users' real voice compared to that of using their recorded voice will be calculated. If the correlation is high, it will provide sufficient evidence to support the use of a regression testing system with recorded human voice. Results indicating this are provided below.

ART offers tremendous benefits since we will only require the user to store his voice once and perform multiple tests with that single stored voice file as compared to doing manual testing where the user will have to be present throughout all the tests. Test cases will be created using error-injected XML grammar files to show that ART actually captures the differences between the two files, in its Comparison Result Window.

An evaluation has been done on ART to show the effectiveness derived from this system. Many evaluation models for regression testing techniques are available but many omit important factors and render some types of comparisons between techniques impossible. However, one recently published cost-benefit model (Rothermel & Do, 2006) seems to

contain sufficient information to be able to perform an effective comparison between the automated regression testing and manual regression testing.

5.2 ART Test Results

A sample regression test was done with ART to show the output of the tester. A newly created XML file along with a sample test case wave file, were chosen to do testing. The wave file contained the spoken words “director” and “friend.” The original grammar oracle contained phrases to recognize the words “director” and “friend.” This implies that when the user chose the wave file, the created speech-to-text oracle should contain the words “director” and “friend.” After the necessary files have been loaded, the “RUN TEST” button was clicked. Figure 15 and Figure 16 below show the result of the test.

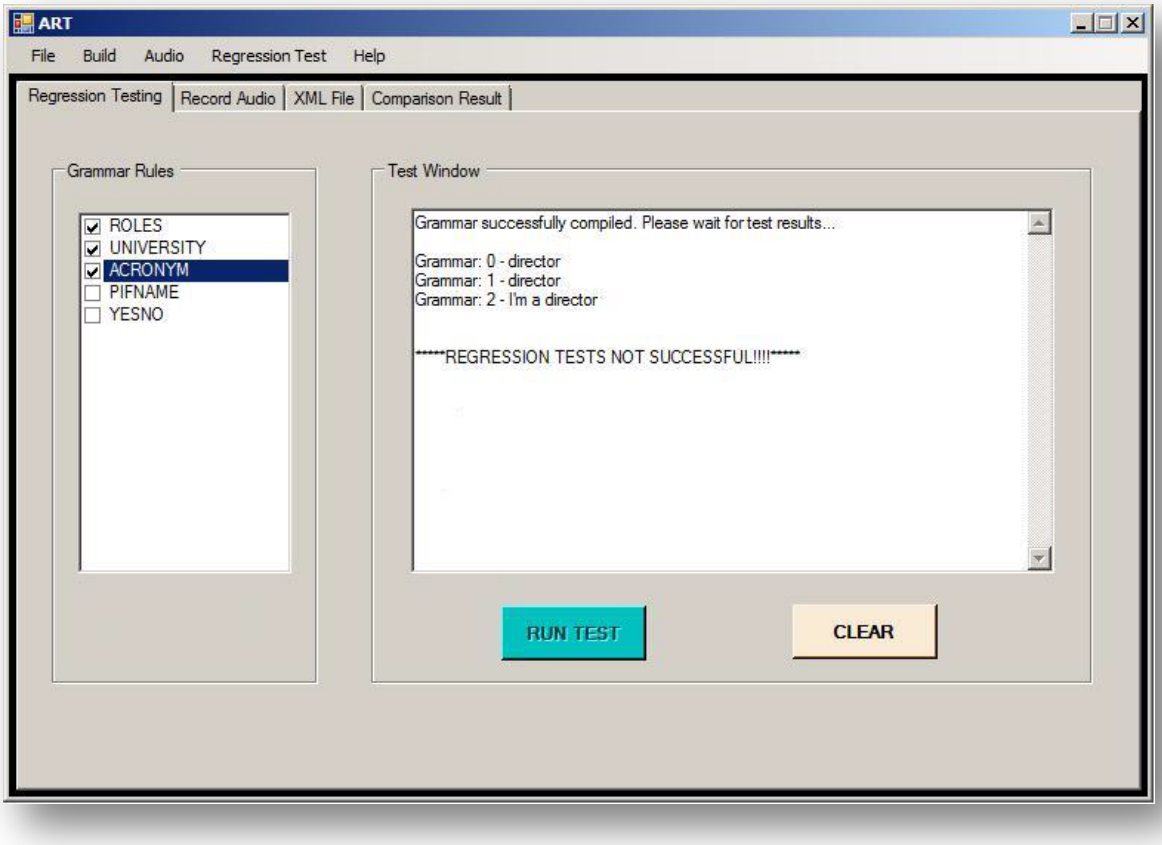


Figure 15: Failed regression test in ART

In Figure 15, the output of the regression tester indicates that the results from the test uncovered hidden anomalies in the grammars. Instead of the expected outcome of "director" and "friend" we now have a speech-to-text translation of "director," "director," and "I'm a director," a clear indication that the addition to or deletion from the grammars had precipitated unwanted recognition behavior. If the window view is changed, by clicking on the Comparison Result tab in ART (shown in Figure 16), we are presented with a line-by-line comparison of the oracle grammar and the newly created XML grammar files.

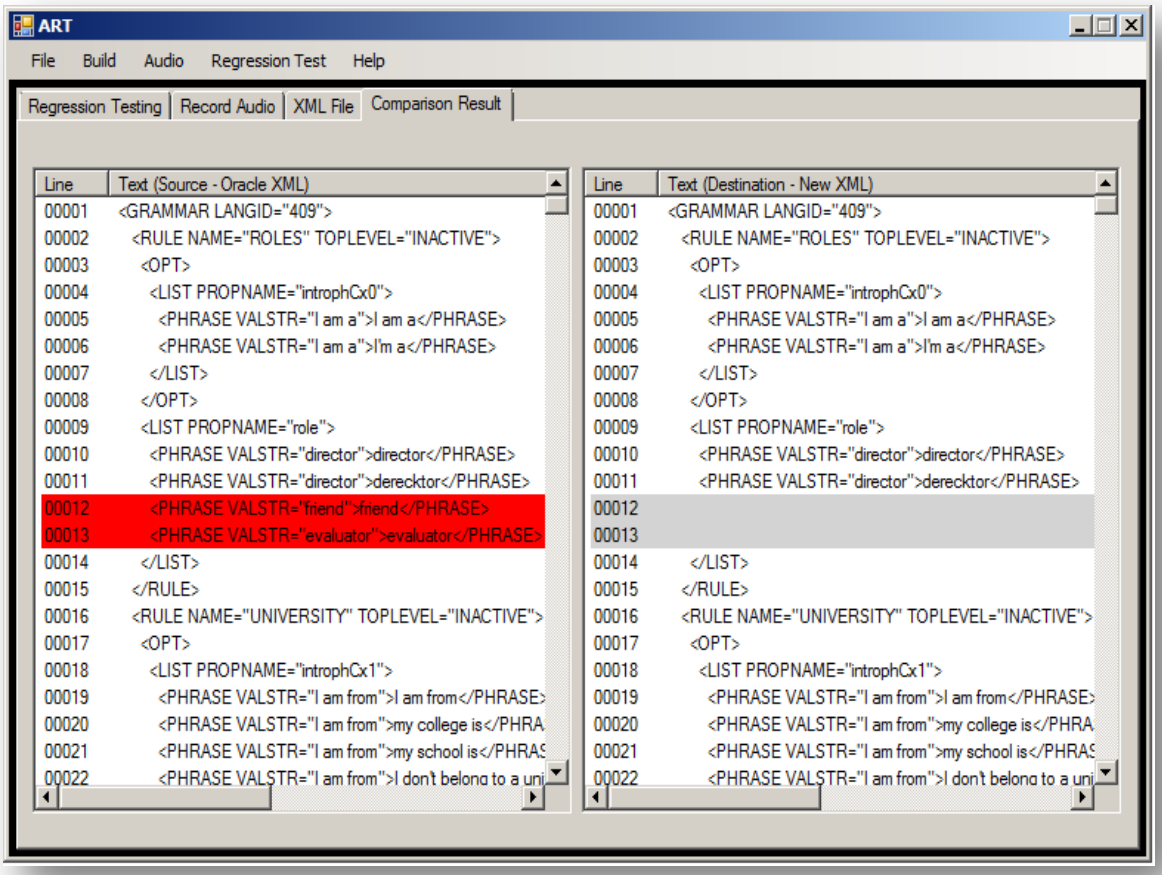


Figure 16: Comparison Result view of failed regression test in ART

Figure 16 above shows highlights of the difference between the two grammar files. It can be clearly seen that lines 12 and 13 from the file on the left (the oracle grammar) are missing from the file on the right (the newly created grammar file). Since the phrase in line 12 is necessary for the tester to properly translate the wave file to text and was missing from the XML file, we can now argue that this has caused recognition to deteriorate. A quick check revealed that these words were mistakenly left out in the

database used to create the new XML file. Hence, the database's integrity had been compromised and would have caused recognition to deteriorate in the next iteration of the LifeLike system, hadn't the regression tester detected the error.

It should be noted that ART does not only detect errors when omitted grammar phrases causes deterioration in recognition quality, but also allows for detection when phrases may have been incorrectly input into the database and hence into the new grammar XML file.

5.3 User Test Cases

Three uses were given three sets of grammar phrases, G_1, G_2, G_3, from the LifeLike domain to conduct a series of recognition tests. The first set of phrases, G_1, comprises fifteen randomly chosen directors' names from different universities that receive funding from NSF. The recognition rates using the users' natural voice was compared to the recognition rates when their recorded voice was used. The recognition observed with recorded voice was obtained by using the regression testing abilities of ART to see how well the system could use a recorded voice sample to do speech-to-text.

Table 3 below shows the raw data collected from the three users. A checkmark in the table indicates that the name was correctly recognized.

Table 3: Recognition data for Directors' Names (G_1)

Director Name (G_1)	User 1		User 2		User 3	
	Natural Voice	*Recorded Voice*	*Natural Voice*	*Recorded Voice*	*Natural Voice*	*Recorded Voice*
Betty Cheng	✓	✓			✓	
Charles Petty	✓		✓		✓	✓
David Goodman	✓	✓	✓	✓	✓	
Frank Allen	✓	✓	✓	✓	✓	✓
Jay Lee	✓	✓			✓	✓
Shah Jahan			✓	✓		
Balakrishna Haridas	✓	✓	✓			
Don Taylor	✓	✓			✓	✓
Samuel Oren					✓	✓
Ram Mohan	✓		✓	✓	✓	✓
Nikos Papanikolopoulos						
Richard Muller	✓				✓	✓
Rahmat Shoureshi	✓	✓	✓	✓	✓	✓
Steven Liang	✓	✓			✓	✓
Sami Rizkalla	✓	✓				

Table 4 below shows the recognition data for fifteen randomly chosen university names.

Table 4: Recognition data for University Names (G_2)

University Name (G_2)	User 1		User 2		User 3	
	Natural Voice	*Recorded Voice*	*Natural Voice*	*Recorded Voice*	*Natural Voice*	*Recorded Voice*
University of Central Florida	✓	✓			✓	✓
University of Texas at Austin	✓				✓	✓
North Carolina State University	✓	✓	✓	✓		
Oregon State University	✓	✓			✓	✓
Purdue University			✓	✓	✓	
University of Utah	✓	✓				
Ohio State University	✓	✓	✓	✓	✓	✓
Michigan State University	✓	✓			✓	✓
Clemson University	✓	✓	✓		✓	✓
Iowa State University	✓	✓	✓	✓	✓	✓
University of Maryland	✓		✓	✓	✓	✓
University of New Mexico	✓					
George Washington University	✓	✓			✓	✓
Carnegie Mellon University	✓				✓	
University of Houston	✓	✓	✓	✓	✓	✓

Table 5 below contains the data accumulated when the users were asked to test the system with the acronyms of fifteen different university names.

Table 5: Recognition data for University Name Acronym (G_3)

University Name Acronym (G_3)	**User 1**		**User 2**		**User 3**	
	Natural Voice	*Recorded Voice*	*Natural Voice*	*Recorded Voice*	*Natural Voice*	*Recorded Voice*
UND	✓	✓			✓	✓
UCF	✓	✓	✓	✓		
FIU	✓		✓	✓	✓	
UL	✓	✓	✓	✓	✓	✓
UU	✓		✓		✓	✓
OSU	✓	✓	✓		✓	✓
UNL	✓	✓	✓	✓	✓	
UP	✓		✓	✓		
UCLA	✓		✓		✓	✓
USD	✓	✓	✓			
RU	✓	✓	✓		✓	✓
ASU	✓	✓	✓		✓	✓
CU	✓	✓	✓	✓	✓	
FSU	✓	✓	✓	✓	✓	
TAMU	✓	✓			✓	✓

From the results gathered, the recognition rates of the system using natural voice and recorded voice were computed for each data set G_1, G_2 and G_3. It is important to note that there was only a single wave file for each grammar set for each user and speech-to-text undoubtedly performs better with shorter utterances than longer ones. Also, in the LifeLike domain each user response will be captured in a single wave file containing no more than two utterances. Table 6 summarizes the recognition rates observed when the three users did voice recognition with ART.

Table 6: Recognition rates for three different grammar sets

Grammar Set	**User 1 Recognition Rate**		**User 2 Recognition Rate**		**User 3 Recognition Rate**	
	Natural Voice	*Recorded Voice*	*Natural Voice*	*Recorded Voice*	*Natural Voice*	*Recorded Voice*
G_1	80%	60%	46.7%	33.3%	73.3%	60%
G_2	93.3%	66.7%	46.7%	40%	80%	66.7%
G_3	100%	73.3%	86.7%	46.7%	80%	53.3%

5.4 Product-Moment Correlation Coefficient

These results obtained will be used to calculate the Pearson's Product-Moment Correlation Coefficient (SpringerLink, 2001) of each data set. This coefficient, which lies

between -1.00 (a perfect negative correlation) and +1.00 (a perfect positive correlation), will establish if and how natural voice and recorded voice are related.

The Pearson's Product-Moment Correlation Coefficient (r) is given by:

$$r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{X_i - \mu_x}{\sigma_x}\right)\left(\frac{Y_i - \mu_y}{\sigma_y}\right)$$

where $\left(\frac{X_i-\mu_x}{\sigma_x}\right)$ is the standard score, μ_x, is the mean and σ_x is the standard deviation, for X. In these calculations we will take $n = 3$ (since this represents the number of users in the test), X will represent Natural Voice and Y will represent Recorded Voice.

The calculated correlation coefficients for grammar set, G_i, where $1 \leq i \leq 3$ are:

Table 7: Correlation between natural voice and recorded voice

Grammar Set (*G*)	**Correlation (*r*)**
G_1	0.98175
G_2	0.96086
G_3	0.83431

The results shown in Table 7 indicate for each grammar set there is a high positive correlation between the Natural Voice and Recorded Voice of the users. With these results it can now be conclusively stated that our claims for using recorded voice to do

automated regression testing has been substantiated; i.e. the recorded voice of a subject is just as good as using the real voice of the subject.

5.5 Error Detection Rate

The audio wave files used in gathering the data for the correlation results were used to calculate the error detection rates in ART. For the initial set of grammars, G_1, two words were randomly picked to be excluded from the set of fifteen words, for each user. This means the words will be excluded from the new grammar XML file to see whether ART can determine if there was an error or not.

For User 1 and grammar set G_1, the names "Don Taylor" and "David Goodman" were both removed from the grammar XML file. When ART was run on the modified XML file, the speech-to-text results produced did not contain those two names. The file comparator was invoked and the difference was shown in ART.

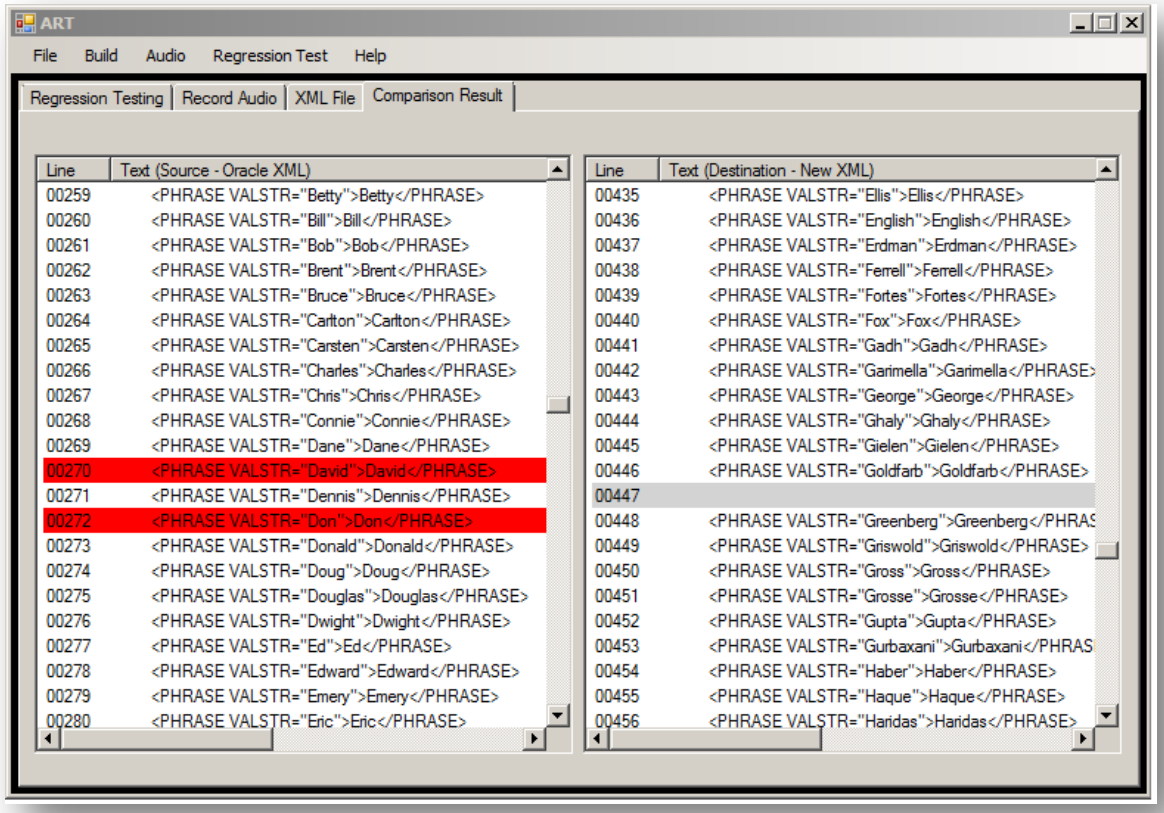


Figure 17: Results of test with G_1 and User 1

In the oracle pane, both directors' first names are highlighted in red because of their omittance from the newly created XML grammar file. The directors' last names were also omitted and the right pane shows in line 447, a blank line which should've contained "Goodman."

A series of tests were conducted with different phrases being omitted for each grammar set for a different user and the results are summarized below in Table 8. A checkmark in the table indicates that ART successfully detected the condition.

Table 8: Error Detection

Omitted Word (User *i*)	**Error Detected by ART**
David Goodman (User 1)	✓
Don Taylor (User 1)	✓
Clemson University (User 1)	✓
Oregon State University (User 1)	✓
TAMU (User 1)	✓
UNL (User 1)	✓
Frank Allen (User 2)	✓
Shah Jahan (User 2)	✓
University of Maryland (User 2)	✓
North Carolina State University (User 2)	✓
UCF (User 2)	✓
FSU (User 2)	✓
Samuel Oren (User 3)	✓
Charles Petty (User 3)	✓
University of Texas at Austin (User 3)	✓
George Washington University (User 3)	✓
UU (User 3)	✓
RU (User 3)	✓

Of the 18 omitted words, ART was able to detect every case where a word was missing from the new XML grammar file due to deteriorated speech-to-text. It should be noted that the words picked to be omitted were all words that were correctly translated from speech to text by ART. Based on the data collected, ART had a 100% error detection rate. This was expected since the oracle grammar contains all the words that were omitted and if the wave file is played against the oracle, there would be correct speech-to-text translation (see Table 3, Table 4 and Table 5).

5.6 Cost-Benefit Analysis

Rothermel and Do (2006) presented a cost-benefit model for regression testing systems that incorporates various factors. ART will adapt to this model to show the benefits derived from the system as opposed to having a human do testing. The two primary equations that comprise their model are as follows:

$$Cost = PS * \sum_{i=2}^{n} \left(CS(i) + CO_i(i) + CO_r(i) + b(i) * CV_d(i) + c(i) * CF(i) \right)$$

$$Benefit = REV * \sum_{i=2}^{n} \Big(ED(i) - \big(CS(i) + CO_i(i) + CO_r(i) + a_{in}(i-1) * CA_{in}(i-1) + a_{tr}(i-1) * CA_{tr}(i-1) + CR(i) + b(i) * \big(CE(i) + CV_i(i) + CV_d(i) \big) + CD(i) \big) \Big)$$

In this model it is assumed that we are considering a regression technique *R*, *n* releases of software system *S* denoted S_1, S_2, …, S_n, and *n* versions of test suite *T* (one per release of *S*) denoted T_1, T_2,…, T_n (Rothermel & Do, 2006).

The terms and coefficients used in the equations defined by Rothermel and Do (2006) are as follows:

- *i* is an index denoting a particular release S_i of *S*.
- *u* is a unit of time (e.g. hours of days)
- *REV* is an organization's revenue in dollars per time unit *u*, relative to *S*.
- *ED*(*i*) is the expected time-to-delivery in units *u* for release S_i when testing begins.
- *PS* is a measure of the cost (average hourly salary) associated with employing a programmer per unit of time *u*.

- $CS(i)$ is the setup cost for testing release S_i.
- $CO_i(i)$ is the cost of identifying obsolete tests for release S_i.
- $CO_r(i)$ is the cost for repairing obsolete tests for release S_i.
- $CA_{in}(i)$ is the time needed to instrument all units in i^2.
- $CA_{tr}(i)$ is the time required to collect traces for test cases in T_{i-1} for use in analyses needed to regression test release S_i.
- $CR(i)$ is the time required to execute R itself on release S_i.
- $CE(i)$ is the time required to execute test cases on release S_i (either all of the test cases in T_i or some subset of T_i).
- $CV_d(i)$ is the cost of applying automated differencing tools to the output test cases run on release S_i (all test cases in T_i or some subset of T_i).
- $CV_i(i)$ is the (human) cost of checking the results of test cases determined to have produced different outputs when run on release S_i all test cases in T_i or some subset of T_i).
- $CD(i)$ is the cost associated with delayed fault detection feedback on release S_i.
- $a_{in}(i)$ is the coefficient used to capture reductions in costs of instrumentation required for release i following change, in terms of the ratio of the number of units instrumented in i to the total number of units in i:

$$a_{in}(i) = \frac{numberOfUnits\ Instrumented}{totalNumberOfUnits}$$

When all units are instrumented, this ratio is 1.

- $a_{tr}(i)$ is a coefficient used to capture reductions in cost of the trace collection required for *i* following changes, in terms of the ratio of the reduced number of traces collected when focusing on changes in *I* to the total number of traces that would need to have been collected otherwise.

$$a_{tr}(i) = \frac{numberOfTracesCollected}{totalNumberOfTraces}$$

  When all traces are collected, this ratio is 1.

- $b(i)$ is a coefficient used to capture reductions in cost of executing and validating test cases for *I*, when only a subset of *T* is rerun:

$$b(i) = \frac{numberOfTestRerun}{totalNumberOfTestsInT}$$

  When all test cases are run, this ratio is 1.

- $c(i)$ is the number of faults that could be detected by *T* on release *i* but that are missed due to execution of subsets of *T*

This model keeps track of the cost and benefits across entire sequences of system releases. In the case of ART we will consider two sequences of the software when calculating the cost and benefit. The cost-benefit analysis will be conducted for automated regression testing (*A*) and computed as $Cost_A$ and $Benefit_A$ while $Cost_B$ and $Benefit_B$ will represent the cost and benefit of using human (manual) regression testing (*B*). We can determine the difference in value between *A* and *B* by calculating:

$$\Theta_{ART} = (Benefit_A - Cost_A) - (Benefit_B - Cost_B)$$

with positive values indicating that *A* has greater value than *B*, and negative values indicating that *B* has a greater value than *A*.

To carry out the cost-benefit analysis of ART, the following values were used for the different variables in the calculations:

- u will be measured in minutes.
- Assume the average pay for a programmer (PS) to do these regression tests is $50 per hour ($0.833 per minute) which will remain constant throughout all calculations.
- $CS(i)$ for *A* is approximately 0.5 minutes and for *B* is approximately 4 minutes.
- $CO_i(i)$ for *A* and *B* would remain constant at 10 minutes since ART doesn't have a way to automatically detect obsolete test cases.
- $CO_r(i)$ will be 0 since we won't consider repairing a test case.
- $b_i(i)$, $a_{in}(i)$ and $a_{tr}(i)$ will be set to 1 for *A* and for *B* hence we will assume all test cases are run, all units are instrumented and all traces are collected.
- $CV_d(i)$ will be 0.0001667 minutes. This was calculated by the automated differencing tool used by ART to compare the two grammar XML files.

- $CF(i)$ as stated by Rothermel and Do (2006) is difficult to calculate and we shall assume there were ordinary faults and use the default cost of 96 minutes.
- REV will be approximated at \$5 per unit time u.
- $ED(i)$ for LifeLike is a few days and in these calculations we will set it at 4320 minutes (3 days).
- $CA_{in}(i)$ could not be measured directly and will be assumed as 3 minutes for both A and B.
- $CA_{tr}(i)$ will be set to the value 5 minutes for B and 0.1 minutes for A throughout all calculations.
- $CR(i)$ measured on average for technique A is approximately 0.0166667 minutes and for technique B approximately 2.2 minutes.
- $CE(i)$ was measured at 0.1666667 minutes for technique A and 3 minutes for technique B.
- $CV_i(i)$ is approximated as 14 minutes since this is the time it took on average to manually compare the two XML files to determine if there were any differences between the two.
- $CD(i)$ will be set to 0 since we are not considering delayed fault detection feedback.

Table 9: Cost-Benefit Analysis for techniques *A* and *B*

Technique	Cost ($)	Benefit ($)
A (automated regression testing with ART)	88.71	21461.08
B (manual regression testing)	91.63	21393.99

From the results gathered above it can clearly be seen that:

$\Theta_{ART} > 0$ which implies $(Benefit_A - Cost_A) - (Benefit_B - Cost_B) > 0$

This means that applying automated regression testing has a larger benefit than manual regression testing. This evaluation provides substantial justification for the need and use of an audio automated regression testing technique like ART provides, in domains akin LifeLike.

CHAPTER 6**: CONCLUSION**

6.1 Summary

Automated regression testing has been in use for over four decades and has provided a cost-saving alternative to manual testing. The real advantage of regression testing is seen in software systems that contain enormous test buckets and require rigorous testing to ensure effective usability. It is impractical to hire humans to do the same tests a machine can do in a fraction of the time with accuracy and precision beyond human comprehension.

LifeLike requires quick effective prototyping which is tedious on the part of the SR. Grammars need to be built and tests need to be conducted to ascertain if the new set of grammars has affected previous recognition. Manual grammar building and testing simply is not able to perform as effectively as automated building and testing, in this regard.

Audio Regression Tester (ART) has been designed, and evaluated based on metrics which seek to show its advantage in the domain of regression testing. ART has accomplished what it had set out to do. We have shown substantial evidence to support the use of ART in the realm of automated audio regression testing. This method of testing, as shown in CHAPTER 5, has a greater benefit than manual regression testing

and costs less. ART's high error-detection rate is due to its effective speech to text translation and file comparison method employed. It can be argued that manual file comparison is still faulty; one reason being that it is hard to detect a single space between characters with the naked eye. This single space, although negligible to humans, negatively impacts computer systems and could cause recognition to deteriorate in domains such as LifeLike.

It is imperative that we seek to improve the quality of software and decrease the time between specification and production of the system; ART is just one step in this direction. Not limited to LifeLike, ART can be used in other audio domains that require regression testing, for example call reservation. Since many of these systems employ a similar speech recognition strategy, updates to the system might require testing with previously collected data to ensure that recognition had not significantly deteriorated.

6.2 Future Work

This research can be extended to allow a better test selection policy. Since tests may become obsolete as the software ages, this selection policy will only select valid tests to be performed with the system. This will definitely reduce the time it takes to sort through tests to decide which have expired and which have not.

Usability testing was not conducted for the current prototype and doing so may open a doorway to allow us to improve the current design and provide additional functionality as needed.

Additional tests need to conducted over the lifetime of LifeLike to ascertain whether or not the quality of recognition has increased by using automated regression testing as compared to manual regression testing. This gives a fair idea of how well ART has performed in the domain of regression testing with speech recognition grammars.

LIST OF REFERENCES

Baker, J. K. (1975). The Dragon System - An Overview. *IEEE Trans. Acoustic, Speech, and Signal Processing* , 24-29.

Bezier, B. (1990). *Software Testing Techniques.* New York: Van Nostrand Reinhold.

Chant Inc. (2008). *Chant SpeechKit*. Retrieved August 6, 2008, from Chant: http://chant.net/Products/SpeechKit/Default.aspx

Chen, S., Kingsbury, B., Mangu, L., Povey, D., Saon, G., Soltau, H., et al. (2006). Advances in Speech Transcription at IBM Under the DARPA EARS Program. *IEEE Transaction on Audio, Speech, and Language Processing* , 1596-1608.

Chen, Y., Rosebblum, D., & Vo, K. (1994). TestTube: A system for Selective Regression Testing. *Proceedings of the 16th International Conference on Software Engineering* (pp. 211-222). Sorrento, Italy: IEEE.

Chow, Y., Dunham, M., Kimball, O., Kranser, M., Kubala, G., Makhoul, J., et al. (1987). BYBLOS: The BBN Continuous Speech Recognition System. *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'87* (pp. Volume 12, 89-92). IEEE.

Chung, S., DeMara, R., & Moldovan, D. (1993). PASS: A Parallel Speech Understanding System. *Ninth Conference on Artificial Intelligence for Applications* (pp. 136-142). Orlando, FL: IEEE.

DeMara, R., & Moldovan, D. (1993). The SNAP-1 parallel AI prototype. *IEEE Transactions on Parallel and Distributed Systems*, (pp. 841-854).

DeMara, R., Gonzalez, A., Hung, V., Leon-Barth, C., Dookhoo, R., Jones, S., et al. (2008). Towards Interactive Training with an Avatar-based Human-Computer Interface. *Interservice/Industry Training Simulation & Education Conference (I/ITSEC).* Orlando, Florida.

Donner, C., & Jensen, H. W. (2005). Light Diffusion in Multi-Layered Translucent Materials. *ACM SIGGRAPH 2005* (pp. 1032-1039). Los Angeles, California: ACM.

Dragon Naturally Speaking. (n.d.). *History of Speech Recognition and Transcription Software*. Retrieved July 12, 2008, from History of Speech Recognition and Transcription

Software, Dragon Naturally Speaking, transcription: http://www.dragon-medical-transcription.com/historyspeechrecognition.html

Dupont, S., & Luettin, J. (2000). Audio-Visual Speech Modeling for Continuous Speech Recognition. *IEEE Transaction on Multimedia* , 141-151.

Embedded ViaVoice. (2008). Retrieved August 4, 2008, from IBM: http://www-306.ibm.com/software/pervasive/embedded_viavoice/

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database.* Cambridge: MIT Press.

Gupta, R., Harrold, M., & Soffa, M. (1992). An Approach to Regression Testing Using Slicing. *Software Maintenance, 1992, Proceedings., Conference on* (pp. 299-308). Orlando, Forida: IEEE.

Hartmann, J., & Robson, D. (1988). Approaches to Regression Testing. *Conference on Software Maintenance* (pp. 368-372). Scottsdale: IEEE.

Heinrichs, A., Müller, M. K., Tewes, A. H., & Würtz, R. P. (2006). Emergent Graphs with PCA-features for Improved Face Recognition. *Information Optics: 5th International Workshop.*

Juang, B. H., & Rabiner, L. R. (1991). Hidden Markov Models for Speech Recognition. *Technometrics* , Vol. 33, 3, pp. 251-272.

Korel, B., & Al-Yami, A. (1998). Automated Regression Test Generation. *SIGSOFT Software Enginering Notes* , 143-152.

Kung, D., Gao, J., Hsia, P., Toyoshima, Y., & Chen, C. (1996). On Regression Testing of Object-Oriented Programs. *Journal of Systems and Software* , Volume 32, Issue 1, pp. 21-40.

Lee, K. F., Hon, H. W., & Reddy, R. (1990). An Overview of the SPHINX Speech Recognition System. *IEEE* , 34-45.

Lenat, D. B. (n.d.). *Hal's Legacy*. Retrieved August 1, 2008, from http://www.cyc.com/cyc/technology/halslegacy.html

Leung, H., & White, L. (1991). A Cost Model to Compare Regression Test Strategies. *Software Maintenance, 1991., Proceedings, Conference on* (pp. 201-208). Sorrento, Italy: IEEE.

Leung, H., & White, L. (1992). A Firewall Concept for both Control-Flow and Data-Flow in Regression Integration Testing. *Software Maintenance, 1992. Proceedings., Conference on* (pp. 262-271). Orlando, Florida: IEEE.

Leung, H., & White, L. (1990). A Study of Integration Testing and Software Regression at the Integration Level. *Software Maintenance, 1990., Proceedings., Conference on* (pp. 290-301). IEEE.

Leung, H., & White, L. (1989). Insights into Regression Testing. *Software Maintenance* (pp. 60-69). Miami: IEEE.

Levinson, S., Rosenberg, A., & Flanagan, J. (1977). Evaluation of a Word Recognition System Using Syntax Analysis. *Acoustics, Speech and Signal Processing* (pp. 483-486). IEEE.

Li, Y., & Wahl, N. (1999). An Overview of Regression Testing. *SIGSOFT Software Engineering Notes* , 69-73.

Lieberman, H., Faaborg, A., Daher, W., & Espinosa, J. (2005). How to Wreck a Nice Beach You Sing Calm Incense. *Intelligent Conference on Intelligent User Interfaes* (pp. 278-280). San Diego: ACM.

Lowerre, B. (1977). Dynamic Speaker Adaptation in the Harpy Speech Recognition System. *Acoustics, Speech and Signal Processing* (pp. 788-790). 1977.

McCarthy, A. (1997, February 1). Unit and Regression Testing. *Dr. Dobb's Journal* .

Memon, A., Banerjee, I., Hashmi, N., & Nagarajan, A. (2003). DART: A Framework for Regression Testing "Nightly/daily Builds" of GUI Applications. *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on* (pp. 410-419). IEEE.

Microsoft Corporation. (2008). *Microsoft Speech API (SAPI) 5.3*. Retrieved August 5, 2008, from Microsoft Speech API (SAPI) 5.3: http://msdn.microsoft.com/en-us/library/ms723627(VS.85).aspx

Microsoft Corporation. (2008). *Speech SDK 5.1*. Retrieved January 10, 2008, from Microsoft SAPI 5.1: http://www.microsoft.com/downloads/details.aspx?FamilyId=5E86EC97-40A7-453F-B0EE-6583171B4530&displaylang=en

Miller, E. J. (1979). Program Testing Technology in the 1980s. *Procs. of the Conf. on Computing in the 1980's* (pp. 72-79). IEEE.

Nuance Communications. (2008). *Nuance*. Retrieved August 4, 2008, from Nuance - VoCon 3200: http://www.nuance.com/vocon/3200/

Nuance Communications. (2008). *Nuance*. Retrieved August 4, 2008, from Dragon Naturally Speaking: http://nuance.com/naturallyspeaking/

Nuance Communications. (2005, September 15). *ScanSoft and Nuance Close Merger*. Retrieved July 10, 2008, from Nuance Press Releases 2005: http://www.nuance.com/news/pressreleases/2005/20050915_closemerger.asp

Qin, Y., Shi, Q., Liu, Y., Aronowitz, H., Chu, S., Kuo, H., et al. (2006). Advances in Mandarin Broadcast Speech Transcription at IBM Under the DARPA Gale Program. In *Chinese Spoken Language Processing* (pp. 410-421). Springer Berlin/Heidelberg.

Rabiner, L., Wilpon, J., & Soong, F. (1989). High Performance Connected Digit Recognition Using Hidden Markov Models. *Acoustics, Speech And Signal Processing* (pp. 1214-1225). 1989.

Rothermel, G., & Do, H. (2006). An Emperical Study of Regression Testing Techniques Incorporating Context and Lifetime Factors and Improved Cost-Benefit Models. *Proceedings of the 14th ACM SIGSOFT* (pp. 141-151). Portland, Oregon: ACM.

Rothermel, G., & Harrold, M. (1997). A Safe, Efficient Regression Test Selection Technique. *ACM Transactions on Software Engineering and Methodology* , 173-210.

Rothermel, G., & Harrold, M. (1996). Analyzing Regression Test Selection Techniques. *Software Engineering, IEEE Transactions on* , 529-551.

Rothermel, G., & Harrold, M. (1994). Selecting Regression Tests for Object-Oriented Software. *Software Maintenance, 1994. Proceedings., International Conference on*, (pp. 14-25). Victoria, British Columbia.

SpringerLink. (2001). *Pearson product-moment correlation coefficient*. Retrieved September 3, 2008, from Springer Online Reference Works: http://eom.springer.de/P/p130060.htm

Stensrud, B. S., Barrett, G. C., Trinh, V. C., & Gonzalez, A. J. (2004). Context-Based Reasoning: A Revised Specification. *FLAIRS.*

W3C. (2004, March 16). *Speech Recognition Grammar Specification Version 1.0*. Retrieved August 19, 2008, from Speech Recognition Grammar Specification Version 1.0: http://www.w3.org/TR/speech-grammar/

White, L. (1996). Regression Testring of GUI Event Interactions. *Software Maintenance 1996., Proceedings., International Conference on* (pp. 350-358). Monterey, California: IEEE.

Wilks, Y. (2005). The History of Natural Language Processing and Machine Translation. In *Encyclopedia of Language and Linguistics.*

Young, S. (1996, September). A Review of Large-Vocabulary Continuous Speech. *Signal Processing Magazine, IEEE* , p. 45.