A ROBUST ARCHITECTURE FOR HUMAN LANGUAGE TECHNOLOGY SYSTEMS

By

Theban Stanley

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2006

A ROBUST ARCHITECTURE FOR HUMAN LANGUAGE

TECHNOLOGY SYSTEMS

By

Theban Stanley

Approved:

Julie Baca Research Professor at Center for Advanced Vehicular Systems (Co-Major Advisor and Director of Thesis)	Joseph Picone Professor of Electrical and Computer Engineering (Major Advisor)
Georgios Lazarou Assistant Professor of Electrical and Computer Engineering (Committee Member)	Nicholas H. Younan Professor of Electrical and Computer Engineering (Graduate Coordinator)

Roger L. King
Associate Dean for Research and Graduate Studies

Name: Theban Stanley

Date of Degree: August 5, 2006

Institution: Mississippi State University

Major Field: Electrical Engineering

Major Professor: Dr. Joseph Picone

Pages in Study: 83

Title of Study: A ROBUST ARCHITECTURE FOR HUMAN LANGUAGE TECHNOLOGY SYSTEMS

Candidate for Degree of Master of Science

In recent years, distributed framework has become a widely accepted platform for implementation of human language technology. The Defense Advanced Research Program Agency (DARPA) Communicator program has been highly successful in implementing this distributed approach. The program has fueled the design and development of impressive human language technology applications with complex inter-process communication between modules.

This latter feature, though beneficial, introduces complexities which reduce overall system robustness to failure. In addition, the ability to handle multiple users and multiple applications is not innately supported. This thesis describes the enhancements to the original Communicator architecture that address robustness issues and provide a multiple multi-user application environment by enabling automated server startup, error detection and correction. Extensive experimentation and analysis were performed and a 7.2% improvement in robustness was achieved on

the address querying task, which is the most complex task in the human language technology system.

DEDICATION

I would like to dedicate this work to my family and friends for their support and encouragement in all my endeavors.

ACKNOWLEDGMENTS

This thesis would never have been possible without the constant support and guidance from Dr. Julie Baca, especially during the final stages of my work. I would to like to thank Dr. Joseph Picone for his constant mentoring through my masters program which has instilled a sense of professionalism in the work I do. I am really grateful to have had an opportunity to be a part of an elite team which always showed a passion for excellence. I also would like to thank Dr. Georgios Lazarou for his support whenever it was needed.

I am deeply grateful to Hualin Gao who mentored me during my initial stages as a graduate assistant at the Institute of Signal and Information Processing (ISIP). He had worked with me to revamp our human language technology package which is the basis of my thesis. I would like to thank Naveen Parihar and Sridhar Raghavan for their support and help during my graduate program. I extend my thanks to all the members of the Intelligent Electronic Systems (IES) program and friends who have made my graduate studies memorable.

I sincerely thank the Center for Advanced Vehicular System (CAVS) for funding and supporting the In-Vehicle Dialog Systems project.

TABLE OF CONTENTS

CHAPTER Page

APPENDIX Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Humans have long dreamt of creating machines which rival their own intellectual abilities. The invention of the typewriter was a step forward in the interaction between humans and computers. In the years following World War II, a fervent search began for alternative methods for human computer interaction. Speech recognition appeared to offer a promising alternative. In the late 1980s, the first successful speech recognition systems were deployed. Current commercial systems offer performance that has improved upon these earlier systems, but none can adequately handle natural, spontaneously spoken language. This capability remains the provenance of research, in systems, which though predominantly prototypes, have become increasingly powerful and complex. As their complexity has increased, however, robustness to failure has presented significant obstacles. In addition, the execution of simultaneous multiple multi-user prototype applications is not readily supported. This thesis has identified and addressed critical barriers to the development of robust multi-user, multiple application prototype systems for human language technology (HLT).

Initial HLT systems, like many software systems of the same era, were designed in a monolithic fashion [1]. As these systems became more complex, development and maintenance requirements made this design approach untenable for progress in the field. This led to the concept of distributed processing in which this monolithic structure was

decomposed into a number of functional components that could interact through a common protocol [1]. This distributed framework was readily accepted by the research community and has been the cornerstone for the advancement in cutting edge HLT systems.

The Defensive Advanced Research Projects Agency (DARPA) Communicator program has been highly successful and many state-of-the-art systems have been built on this architecture [1]. The DARPA Communicator architecture was developed and optimized for HLT systems. Though the DARPA program has concluded, the Communicator architecture is available in the public domain and provides a feasible environment for long-term research in HLT [2]. Many labs including Carnegie Mellon University (CMU) [3], Center for Spoken Language Processing (CSLR) [4] and SRI [5], [6] continue to conduct fundamental HLT research using complex systems designed on this open source architecture. The plug-and-play ability facilitates intermixing of components developed by different sites. It has a programmable hub that allows flexible control over the interaction between servers. These notable features made the DARPA Communicator a viable architecture for the implementation of HLT systems.

Despite the DARPA Communicator's advantages, it suffered critical robustness issues which grew in magnitude as more complex systems were developed. Also, multi-user and multiple application capability were innately supported. This thesis has addressed these issues by incorporating a series of enhancements that were implemented and formally evaluated on a prototype HLT system that consists of four main applications. Details of experiments conducted to measure and evaluate enhancements are given in Chapter IV.

The prototype HLT system consists of four major components: speech analysis, automatic speech recognition (ASR), speaker verification and a dialog system. The speech analysis component records and plays back audio and displays the waveform, spectrogram, and energy. The speaker verification component verifies the authenticity of the speaker by comparing a statistical model [7] of the test utterance to a model of the claimed speaker's voice. The dialog system component is a navigation system [9] that responds to queries about directions and places at Mississippi State University (MSU) and the adjacent city of Starkville, Mississippi. All these applications use a public domain, hidden Markov model (HMM)-based speaker-independent continuous speech ASR system [1], [11], [12] developed by the Institute for Signal and Information Processing (ISIP) at Mississippi State University.

1.1 Thesis Scope and Contribution

In the past decade, advances in distributed computing technologies have led to the realization of complex HLT systems. The distributed framework of the Communicator architecture has supported the development of systems with highly complex communication among software modules and processes. While this has increased the overall power and capability of HLT systems, the complexity of the inter-process communication has decreased robustness and significantly degraded the performance of the systems built on this framework. Further, the Communicator architecture does not inherently support the ability to handle multiple users running multiple applications from a common interface. Multi-user and multiple-application capability are crucial to widespread acceptance of this technology.

The main goals of the thesis were to identify and address the critical barriers to the development of robust multi-user HLT systems. The key contributions of the thesis include:

1.1.1 Robustness Enhancements

A finite state machine architecture and a basic handshaking protocol were incorporated into the original Communicator architecture to address robustness issues. Experiments conducted to quantify the robustness improvements are discussed in detail in Chapter IV. The robustness enhancements are as follows:

- State Machine Architecture: The servers were redesigned to use a state machine architecture to explicitly monitor the invocation process. The flow of the invocation process was partitioned into stages which correspond to finite states in the server. At each state, the server expects to receive a specific message in the form of a Communicator frame, from a specific server. If the message received at a given state is different from the expected message, the server generates an error and exits the program. If the correct message is received, the server transitions to the next state and waits for further invocation. This architecture detects errors more readily and allows for more graceful error recovery. It also helps in debugging as information about the state and message that caused the error can be retrieved from the server. This enhancement is discussed in detail in Chapter III.
- Basic Handshaking: The state machine architecture and basic handshaking, which work hand in hand, have proven effective in detecting errors. The states are defined on the basis of the different stages of handshaking between the servers. The

Communicator architecture provides a basic structure called a “frame” for communication among servers and processes. This structure implicitly allows a strict handshaking protocol, but does not require or provide an implementation of such a protocol. Implementing and enforcing such a protocol became critical for system robustness as the number and complexity of the applications increased in magnitude. The system was redesigned to include basic handshaking capabilities in the client-server communication. Chapter III discusses these enhancements in detail.

1.1.2 Qualitative Enhancements

Enhancements that provided automated handling of multiple users and multiple applications were also introduced. Scenarios further detailing these enhancements and their evaluation are discussed in detail in Chapter IV. The qualitative enhancements are as follows:

- Multi-user Capability: The system architecture has been redesigned to support simultaneous execution of multiple multi-user applications. The DARPA Communicator package includes a Python interface called the “Process Monitor.” It provides a visual interface to observe the processes, and terminate/restart them when necessary. Though a useful interface, it required human observation of processes and manual termination and restarting of processes if problems were observed. Overcoming this disadvantage required developing a process manager module with intelligence to automatically detect failed processes and initiate restarts. The process manager controls all the processes running on the server side, by encapsulating them in a Java™ process [13] object. This empowers the process manager to create a

process, wait on a process, perform input/output on a process, check the exit status on a process and terminate a process. The process manager enables a multiple-user capability by keeping track of client and server associations. It also handles port allocation which is necessary when different processes need to communicate through ports.

- Multiple-application Capability: The demo selector is the client user interface module which allows the user to choose from multiple applications. Once the user chooses an application, the demo selector invokes the user interface needed for the corresponding application. The demo selector also notifies the process manager of the application chosen by the user. The process manager responds to the demo selector by creating the needed servers for the application. It continually tracks the status of these servers and notifies the client program of any developments.

1.2 Structure of the Thesis

Chapter II describes the original DARPA Communicator architecture, including basic terminology, functionality, and architectural strengths. The chapter concludes by discussing vulnerabilities in the architecture that affect the robustness of large, multi-application demonstrations. Chapter III presents in detail the enhancements to the system architecture to address these vulnerabilities, including the addition of multi-user and multiple-application capabilities, the redesign of the servers to include a state machine architecture, and the addition of basic handshaking to the client-server communication. Chapter IV presents evaluation data that measures improvements in the robustness of the

HLT system, built on the enhanced architecture. Chapter V presents conclusion and future areas of research.

CHAPTER II

THE ORIGINAL DARPA COMMUNICATOR

Once distributed systems became the widely accepted implementations of human language technologies, there arose a need for a common architecture that would support reusability and compatibility between modules developed at different sites. Many projects were conducted with the goal of creating a common open source platform for HLT. Some notable projects which were initiated in the mid-1990s included the Advanced Language Engineering Platform (ALEP) [14] the General Architecture for TEXT Engineering (GATE) [15] and the TIPSTER project [16]. The common goal of these projects was to produce a common architecture which would improve document processing efficiency and cost effectiveness for a diverse range of text-based applications such as information retrieval, information extraction and automatic text summarization.

The Communicator program was funded by DARPA for the purpose of creating an open source architecture for spoken language applications. It was one of the first architectures to provide a conversational and multi-modal interface for human language technologies [17]. The Communicator architecture was designed using the MIT Galaxy II system [19]. The wide availability of Communicator compatible components, such as speech recognition, dialog management and a spoken telephone interface [20] made it valuable to speech researchers. Its success is evident from the wide variety of applications that were developed using the Communicator architecture, which include

navigation systems [9], weather information systems [21] and travel planning systems [22].

Most of the abovementioned architectures have been predominantly research-oriented. Widespread commercialization of HLT has led to web-based technology platforms. Some examples of successful technology include Nuance's SpeechObjects [23], which is based on VoiceXML [24], and Philip's SpeechMania [25], which is an online architecture based Philip's special purpose programming language known as the High-level Dialogue Definition Language (HDDL) [26].

2.1 Communicator Architecture

The DARPA Communicator architecture was developed and optimized for HLT systems. The Communicator has a "hub and spoke" architecture with a programmable hub that allows flexible control of interaction among servers. Figure 1 illustrates the architecture of the prototype dialog system. The servers include the speech recognition [9], database and dialog management servers [9], all developed at the MSU Center for Advanced Vehicular Systems (CAVS) and the natural language parser [27] developed at CSLR, University of Colorado. The plug-and-play capability of the Communicator architecture is well known for reducing prototype development time by enabling sharing of components across sites. It also provides a standard platform for evaluation of systems developed by different laboratories.

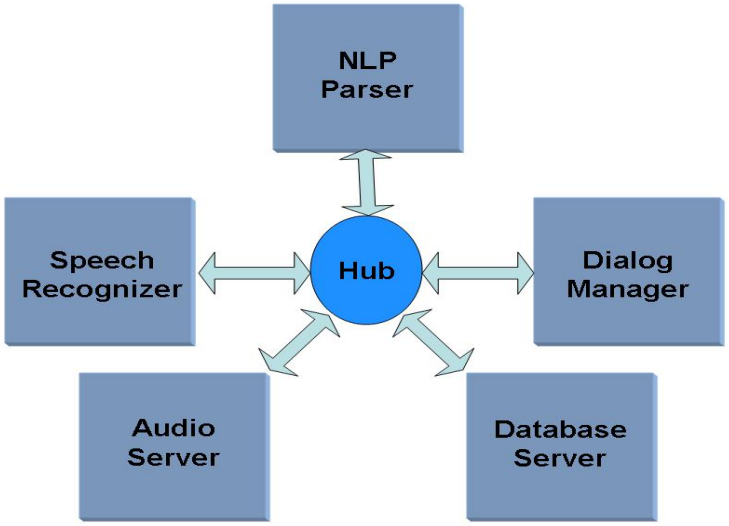

Figure 1 A common architecture for a dialog system.

2.2 Basic Terminology

This section introduces terminology specific to the Communicator architecture, as well as its usage specific to the demonstration system. Some terminology that will be used frequently in this thesis includes:

- Server: Any process that communicates with another process through a hub is called a server. All processes running in the prototype system are servers.
- Client: This thesis uses the term “client” to denote the user interface module on the user’s machine (e.g. laptop). Note that Communicator views all processes as servers irrespective of their functionalities.
- Hub: The hub is the backbone of the Communicator architecture. The hub routes messages from one server to another making it possible for the servers to communicate.

- Frame: The hub monitors all communication among servers. Supporting this communication requires a standard protocol, which for the Communicator architecture is based on an entity called a frame. A Communicator frame is a data structure consisting of a name, a set of key-value pairs [28]. By convention the keys start with a colon followed by a name for the key. The key can be assigned any standard data types such as integer, float, string, frame, list, etc.
- Message: A message is a frame which is passed from a server to the hub or vice versa. A message can trigger a new message or can simply be an acknowledgement of a message received. When a message is sent to the hub, the rule corresponding to that message is triggered. This rule, in turn, triggers a new message which is sent to the recipient servers.
- Dispatch Function: A function that can be directly invoked from the hub is called a dispatch function. When the hub receives a message, it looks for the appropriate rule and invokes the dispatch function of the intended server. In most cases, the dispatch function sends back a new message which triggers further communication.
- Token: A token is a copy of the incoming message stored in the hub to keep track of the messages it receives. Once the hub finds a dispatch function corresponding to the message, it sends the message to the appropriate server and deletes the token.
- Hub Rule: When the hub starts up, it reads a program file called the hub script. The hub script has the list of servers, port numbers and logging instructions. It also has a set of rules which the hub uses as guidelines to take the appropriate action for a given message frame. These rules are called hub rules and they dictate the response of the hub.

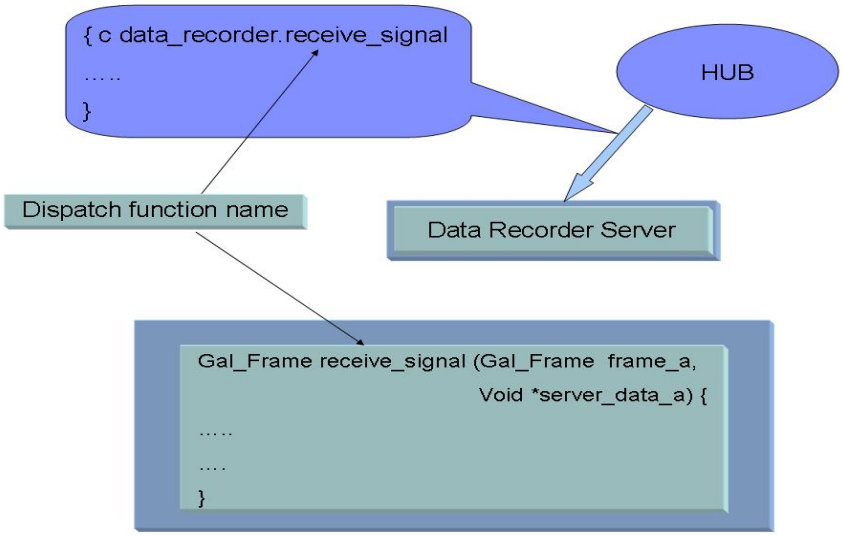

Figure 2 A server dispatch functio n is called by the hub.

2.3 The Invocation Process

When a hub sends a message to the server, this message also includes a dispatch function which is supported by the server and needs to be invoked by the hub. Usually the name of the dispatch function matches the name of the message sent by the hub.

A message is called qualified if it has the information about which server should receive the message. Messages can be unqualified in some cases. Figure 2 illustrates the invocation process. It can be noted that the message from the hub contains information on the server for which the message is intended and corresponding dispatch function that needs to be invoked.

Figure 3 shows a sample hub rule which has information about the dispatch function to invoke, in which server to invoke it, and the keys that need to be sent to the server. There are usually two modes of interaction for the hub. In a scripted interaction, the hub searches for the hub program which matches the name of the incoming message.

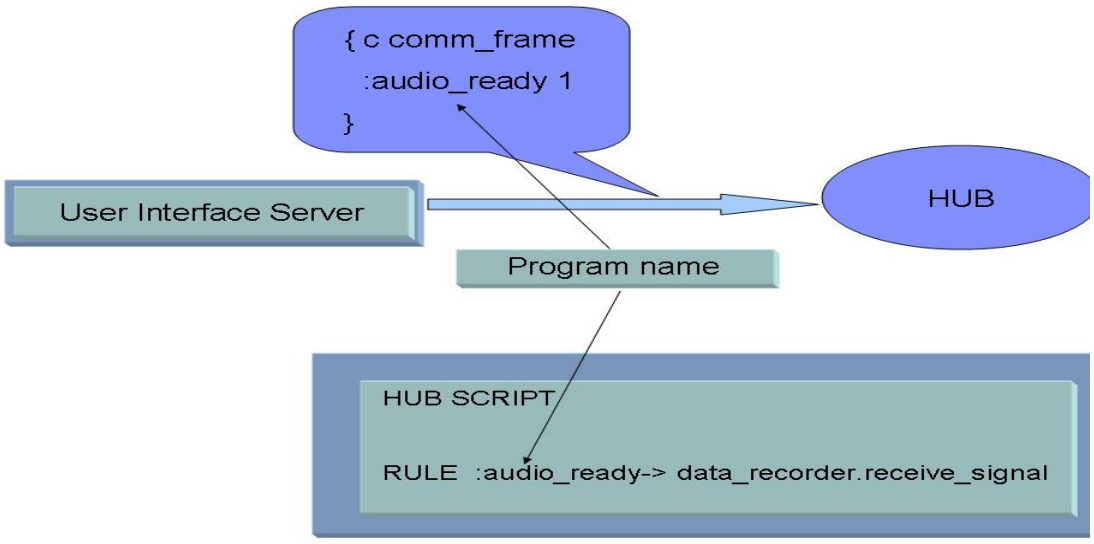

Figure 3 Illustration of a hub triggering an appropriate hub rule.

If the hub cannot find a hub program which matches the name of the incoming message, it tries to find a dispatch function in one of its servers. This is called scriptless interaction. If no hub script or dispatch function is found with the specified name, the message is discarded with a warning. Figure 3 illustrates a scriptless interaction. The user-interface server sends a message frame containing the key ":audio_ready" to the hub. The hub searches the hub script and finds a rule for a frame with ":audio_ready" key. The rule invokes the receive_signal function of the data recorder server.

2.4 Initiation of the Communicator System

This section discusses the series of actions that are triggered when the Communicator system is started. To start the system, the hub can be invoked first followed by the servers or vice versa. In the initial prototype system, the servers were invoked first followed by the hub.

Figure 4 illustrates the steps involved in starting the Communicator system. The following describes the most common sequence of steps by which the Communicator system can be invoked and executed:

- Step 1: All the servers are started. The servers listen to the port waiting for communication from the hub.
- Step 2: The hub is started. The hub reads the hub script file. From the hub script, the hub gets information about the servers it needs to contact, port information and the location it needs to write the log files.
- Step 3: The hub initiates a connection with the servers. During this initialization message, it invokes the “reinitialize” dispatch function in the servers.
- Step 4: At this instant, all the servers are initialized and are in their ready states waiting for message frames. Usually the trigger comes from an event invoked by the user from the user interface server. For example in case of the speech analysis application, pressing of the record button initiates the control flow.

2.4.1 Initial Prototype System Servers

The final demonstration system described in Chapter I was designed and developed iteratively. Figure 1 shows the initial prototype system, a spoken language dialog navigation and information system, that was the first phase of an iterative design and development process. The success achieved in developing this system created a foundation for building a speaker verification system which authenticates users using the voice samples. The following subsections describe the different servers in the initial prototype system.

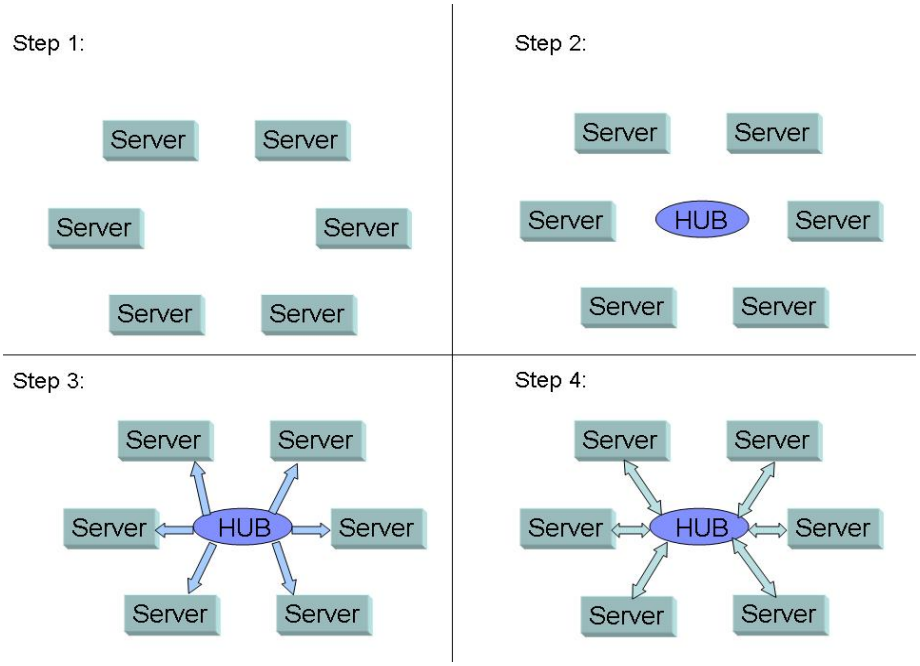

Figure 4 Steps involved in starting the Communicator system.

2.4.2 Audio Server

The Audio server that was used in the initial prototype was developed by MITRE Corporation. The Java™ Desktop Audio Server (JDAS) [20] was intended to provide a cross-platform, Communicator-compliant, desktop audio interface to the speech recognition and synthesis servers. JDAS features an event-driven, multi-threaded architecture which interacts appropriately with legacy servers and introduces the capability of sending and receiving audio in a variety of formats supported by the Java™ Sound API [30]. Telephony support is simulated using a keypad graphical user interface (GUI).

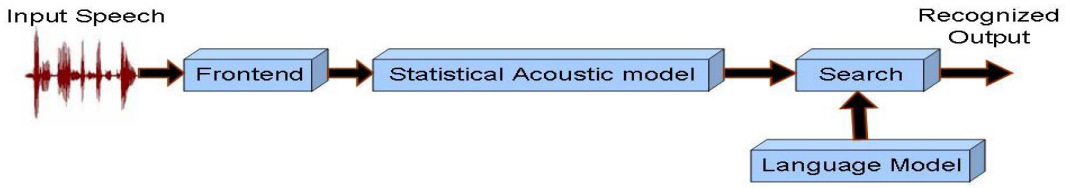

Figure 5 The block diagram of the recognition process.

2.4.3 Recognition Server

The speech recognition server uses a public domain HMM-based speaker-independent continuous speech recognition system [31] which is based on a generalized hierarchical time-synchronous Viterbi beam search decoder [32]. Figure 5 shows the different components that are used in the recognition server. The front end block is responsible for feature extraction which is the process of extracting mel-frequency cepstral coefficients (MFCCs) [33] from the speech signal. The acoustic model is trained using state-of-the-art statistical techniques to learn the characteristics of the speech signal [34]. The search module uses the Viterbi algorithm and determines the best hypotheses. A language model is used to guide the search process with prior information about the language.

2.4.4 Natural Language Parser Server

The natural language parser server uses the Phoenix parser [35], an open source software developed by CSLR, University of Colorado. The grammar and the dialog manager code were developed at CAVS to post-process the natural language parser output. The parser attempts to map the decoded output to a set of semantic frames. A frame is a named set of slots [9]. Each slot has a context-free grammar (CFG) that

specifies a word sequence. The grammars are compiled into recursive transition networks.

An example of a frame requesting for driving information is shown below:

FRAME: Drive

[route]

[distance]

The [route] slot is used to fit in the queries related to specific routes. The [distance] slot accommodates distance queries from one place to another. A subset of CFG rules are shown below [9]:

[route]

(*IWANT * [go_verb] [arriveloc])

IWANT

(I want *to) (I would *like *to) (I will) (I need *to)

[go_verb]

(go) (drive *to) (get) (reach)

[arriveloc]

[*to [placename] [cityname]]

This type of grammar is useful for HLT systems because spontaneous spoken language is often ungrammatical.

2.4.5 The Dialog Manager

The dialog manager coordinates the activities between the speech recognition, parser and back-end application servers [9]. The dialog manager obtains the N-best parse

from the natural language parser and selects the best parse by scoring the slots. The information is merged with a set of context frames. The dialog manager attempts to resolve the user's request by creating a database query. The database server responds to the database query by retrieving the reply from the SQL database. The reply is formatted by the dialog manager and sent to the client program to be displayed in the user interface. If the dialog manager does not understand the query or if the query is ambiguous, it prompts the user for the missing information.

2.4.6 Back-end Application Server

The back-end application server consists of a Structured Query Language (SQL) database and a generic interface to access the internet to retrieve the requested information. Geographic resource sites such as Travelocity [36], Expedia [37] and Mapquest [38] are widely used in the research community especially for navigation [39] and Geographical Information Systems (GIS) [40] research. The server uses Mapquest as Mapquest is more suited for the address and direction querying functionality of the dialog system application.

The dialog manager sends the query frame to the database server. The records in the database are searched for a response using basic SQL commands. If no match is found, an HTTP-based request is submitted to a travel website via the Internet. A Perl script performs the function of logging onto the website and parsing the results from the HTML page. The records obtained from querying the website are inserted as rows into the SQL database. In case the same query is made by the user, the server need not contact the travel website as it can be found in its database.

2.4.7 Speaker Verification Server

Functionally, the speaker verification server is similar to the recognition server. The audio data is converted to features and are processed to obtain likelihood scores. These likelihood scores are calculated based on a set of trained models on a per frame basis [7]. The likelihood scores are then combined via an HMM to yield an overall utterance score, which is a value used by the system to make a decision on whether to accept or reject the claimed identity. The server maintains two speaker models, i.e., authorized user and imposter. The overall utterance scores obtained from both models are compared using a simple threshold test. The server outputs an acceptance or rejection hypotheses based on this threshold test. Figure 6 shows the block diagram of the speaker verification server.

Once the initial prototype system was extended to accommodate two applications, there arose a need for a more multi-user/application-friendly architecture. This issue along with other disadvantages in the original architecture provided a strong case for a transition to an enhanced, more robust architecture.

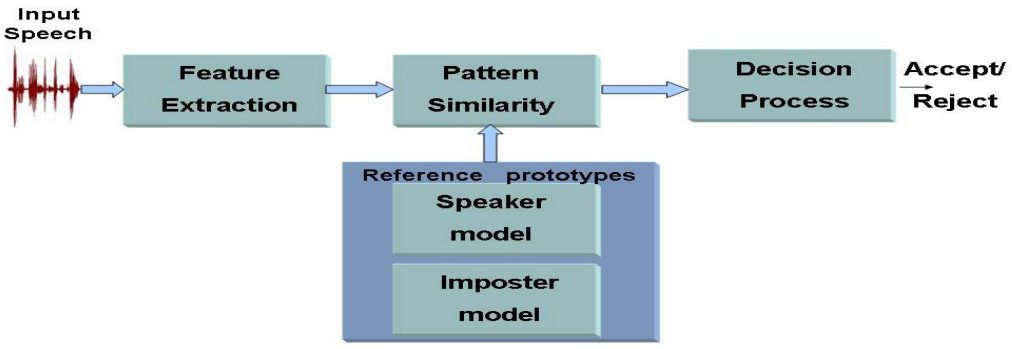

Figure 6 The block diagram of the speaker verification process.

2.5 Disadvantages of the Prototype System

During the initial design phase, communication deadlocks among servers were common as were memory management issues that were difficult to debug. Basic logging mechanisms were provided to address some of these issues, but certain desirable features were not available, such as automated server startup, error detection and correction. The original DARPA architecture did not have an interface to choose from different applications and had to be manually started. The original architecture serviced multiple users, but required manual server startup, including manual port allocation to avoid port conflicts. It was anticipated that such issues would grow in number and complexity as multiple multi-user applications were added. The following subsections discuss the three major issues which required immediate attention.

2.5.1 Deadlocks in Communication between Servers

Frequent deadlocks in the communication between servers were experienced in the initial prototype. The two main reasons for these deadlocks were server failures and misfiring of user interface events. The system also lacked a mechanism for logging communications between servers, which made debugging very cumbersome. This created a requirement for restructuring the servers and modules to monitor and detect server failures. To support this requirement, a more organized mechanism of logging the communication between servers was needed.

2.5.2 Automated Recovery from Server Failures

The client and the servers of the prototype system ran on different machines and communicate through sockets. When a process failed on the server side, the user has to manually restart the process. Even though Galaxy's process monitor provided an interface to start and terminate the servers, it required manual monitoring. This led to the necessity of a module with the intelligence to start servers, check their status and terminate all processes when the application is closed.

2.5.3 Multiple Simultaneous Users

One of the main disadvantages of the DARPA Communicator system is that there was no mechanism to handle multiple users. The appropriate servers for each user had to be manually started and ports manually allocated to ensure no port conflicts occurred. This disadvantage acted as a barrier for multiple user support. This led to the need for modules that can keep track of the client-server association and automatically allocate ports for an application started by the user.

2.5.4 A Common User Interface

Supporting multiple applications required a common interface that allows the user to choose from a host of applications and coordinates interprocess communication between servers and the client process. Each of these applications had its own user interface and set of computational servers. The original architecture lacked modules which can offer a common interface and subsequently start the appropriate user interface depending on the chosen application.

These disadvantages made the transition to a more automated and robust architecture inevitable. Chapter III discusses in detail the different enhancements that were made to overcome the above mentioned issues.

CHAPTER III

ENHANCEMENTS TO THE COMMUNICATOR ARCHITECTURE

To overcome the disadvantages discussed in Chapter II, careful redesign of the architecture was necessary followed by implementation and rigorous testing. There arose a need for a simple application that could serve as a test bed. This led to the implementation of the speech analysis application which is a basic audio recording/playback utility with some enhanced features such as energy, waveform and spectrogram plots. Figure 7 shows the control flow of the speech analysis application. The following sections discuss the various modular and architectural enhancements that were initially implemented on the speech analysis application and later extended to complex applications such as the dialog system.

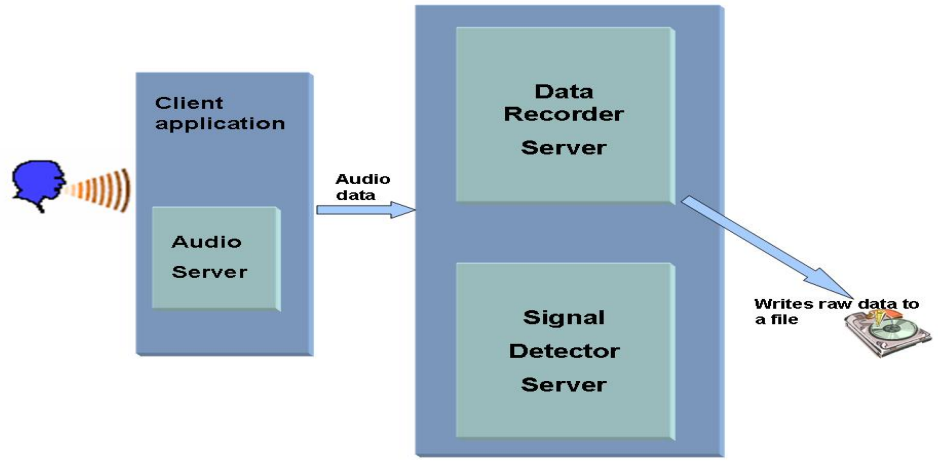

Figure 7 Control flow in the speech analysis application.

3.1 Modularity Enhancements

The knowledge gained from implementing and testing the initial prototype system enabled the identification of certain modules to be added or removed to address the deficiencies found in the prototype system. The following subsections discuss the added modules in detail.

3.1.1 Audio Server

The JDAS audio server which was used in the initial prototype had cross-platform compatibility issues [20]. MITRE corporation abandoned the development and subsequent support of the JDAS server. This led to the development of an indigenous Java™-based Audio server with record/playback capabilities. It uses Java™ Sound package [30] to interact with the audio hardware and is integrated into the client programs of each application.

3.1.2 Data Recorder

The data recorder works in unison with the audio server. While the audio server is a Java™ program that interacts with the audio hardware on the client side, the data recorder is responsible for collecting the audio samples and writing it to disk. This is a C++ program that uses ISIP Foundation classes (IFC) [42] and supports most of the commonly used audio formats including Sof [43], which is an ISIP internal format.

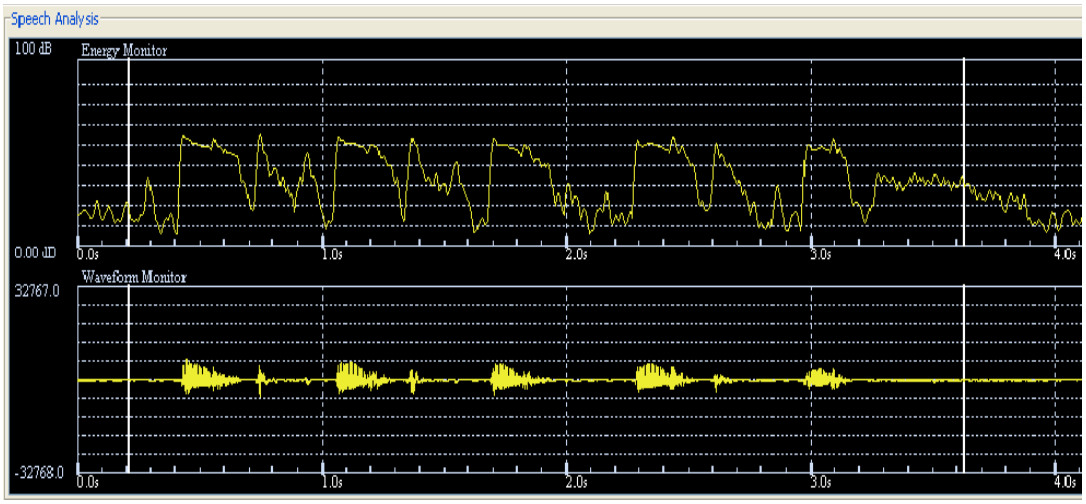

Figure 8 Overall energy and waveform plots, along with utterance endpoints.

3.1.3 Signal Detector

The JDAS audio server had the ability to detect audio activity but did not provide an efficient and flexible mechanism to control its utterance detection algorithm. The replacement of JDAS server by an indigenous audio server led to the need for a speech activity server to perform utterance detection. The signal detector server performs this function by employing basic digital signal processing algorithms, i.e., energy and zero crossing to detect the audio signal.

During the recording process, the audio server streams the audio data in real time to the signal detector server. The signal detector server computes the energy of each audio frame and assigns it an energy state [44]. The signal detector keeps track of these energy states and uses these states to identify changes in speech activity. This process is depicted in Figure 8. The energy and the waveform plots are shown for a typical speech utterance. The white vertical lines represent the start and end of utterance as determined by the signal detector. We refer to these time coordinates of these marks as the utterance

endpoints. The signal detector computes these endpoints in real time as audio data is streamed to it, and hence allows downstream applications in the demo system to begin processing data as soon as these endpoints are located. The net effect of this server is to give the demonstration system the ability to do voice-actuated recording and processing.

3.1.4 Display Module

The display module provides the ability to plot the energy, waveform and the spectrogram of the recorded audio data. A common constraint while plotting the signal is that the number of pixels on the screen is usually less than the number of samples to be plotted. In rare cases, the reverse is also possible. The display algorithm adapts to these changes by calculating the pixel to sample ratio and then branching to the appropriate display algorithm depending on the ratio. The energy is plotted by computing the root mean square of the sample values for each frame of audio data. The waveform is drawn by plotting the minimum and maximum amplitudes in a frame of audio data.

The spectrogram [34], [45] is more computationally intensive than the energy and the waveform plots. The audio data is windowed, zero padded and the Fast Fourier Transform (FFT) [46] of every audio frame is computed. These spectral magnitudes are transformed into the log domain and the decibel (dB) values are normalized to a specific range of colors specified by the color map. The display module can operate in two modes, *real-time* and *overall*. For the energy and the waveform plots, the *real-time* and *overall* modes differ only in the amount of data plotted on the screen, with no significant differences otherwise. In the case of the spectrogram plots, the *real-time* and *overall* plots have some fundamental differences which are explained below in detail.

One major issue addressed in the spectrogram implementation is the normalization of the dB range to the number of distinct colors in the color map. In *real-time* mode, as there is no knowledge regarding the dB range of the audio data, default values with a dynamic range of 60 dB are used. These ranges can defined by the user by varying the minimum and maximum spectral magnitude values. The spectrogram scaling can be varied by changing parameters such as brightness, contrast, minimum and maximum spectral magnitudes which user definable using the configuration menu.

Figure 9 shows the configuration menu which enables the user to alter the default configurations of the audio and display settings. The following equation scales the dB spectral values to corresponding color maps:

$$a = (b - c) * ((d * (e + 0.5) / ((f + g) - (h + g)))$$

where,

a $\rightarrow$ computed colormap
b $\rightarrow$ spectral magnitude in Decibels
c $\rightarrow$ minimum spectral magnitude in Decibels
d $\rightarrow$ color levels in the choosen color map , (1)
e $\rightarrow$ contrast value in (0,1) range
f $\rightarrow$ maximim spectral magnitude in Decibels
h $\rightarrow$ brightness in Decibels
g $\rightarrow$ minimum spectral magnitude in Decibels

Changing the brightness produces a shift in the maximum and minimum spectral ranges while contrast produces a linear compression within the range. As dB values are calculated in *real-time* mode, a probability distribution function of the dB values is computed. Once the mode switches to *overall* mode, the cumulative distribution of dB values is computed and a user-specified percentage is used to calculate the weighted

minimum and maximum dB range. The overall plot uses these weighted spectral magnitudes instead of user defined magnitudes which are used in *real-time* plotting..

Figure 10 shows the overall spectrogram plot of the word “drown” pronounced by a female speaker. Extensive memory optimization was performed to make the plots computationally less intensive during the *real-time* mode.

3.2 Architectural Enhancements

Among the numerous disadvantages discussed in the previous chapter, the most critical were automating server startup, error detection and correction, and application control from a single common interface. In addition, automating the server startup and adding the multiple user and multiple application capability emphasized the need for improved debugging capabilities.

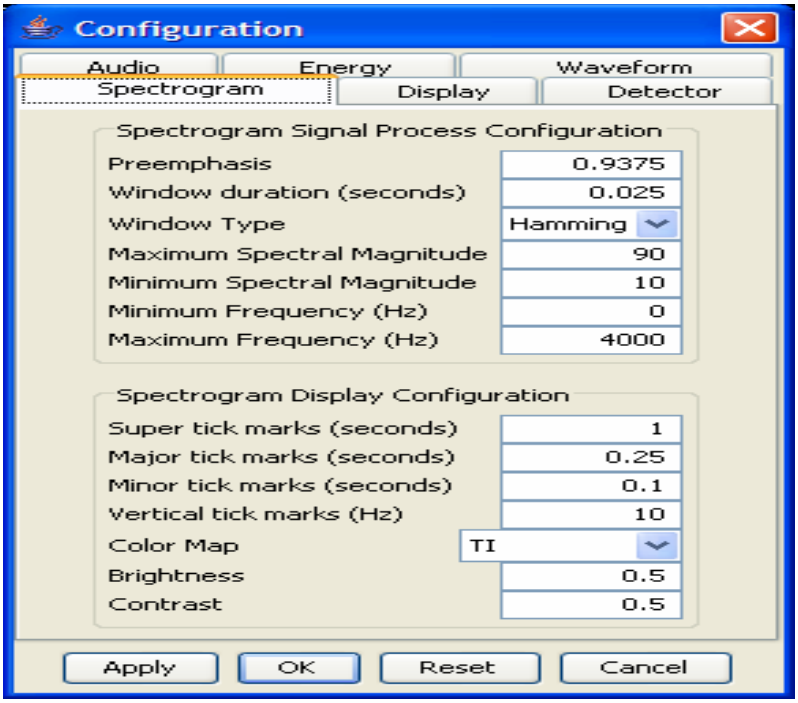

Figure 9 The configuration menu

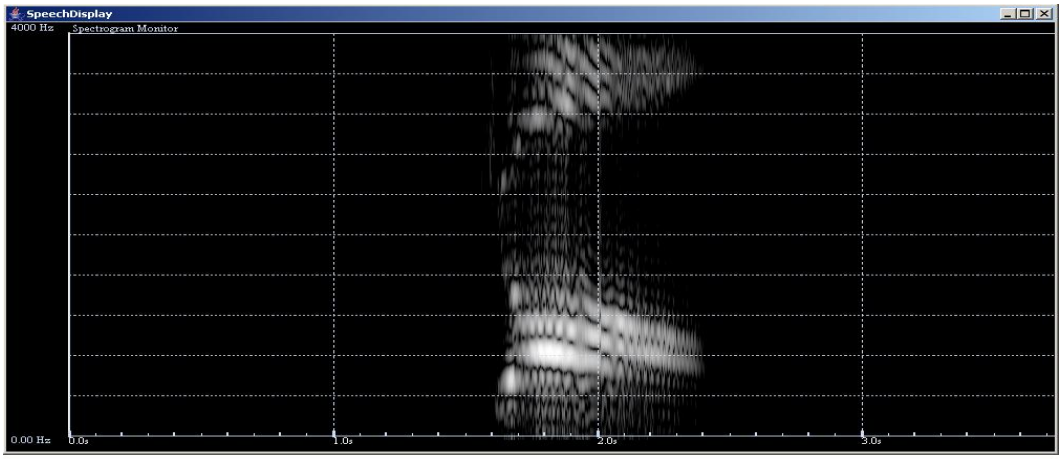

Figure 10 Spectrogram of the word “drown” pronounced by a female speaker.

3.2.1 Automated Server Management

Automated server management became critical with the requirement to run multiple applications simultaneously [47]. Though Communicator’s process monitor provides an interface to start and terminate servers, it requires manual monitoring. To address this issue, a process manager module was designed to automatically start and control all server processes in the prototype system architecture. Figure 11 shows an overview of the multi-user architecture for multiple applications. The process manager controls and monitors all server applications. The user’s client application must contact the process manager to start the required servers, before directly establishing connections with them. When the user begins interacting with the common interface, the client program displays the different applications from which the user can choose. When the user selects a certain application, the client program requests the process manager to start the respective servers and the hub. The process manager maintains information about what servers are required for each application.

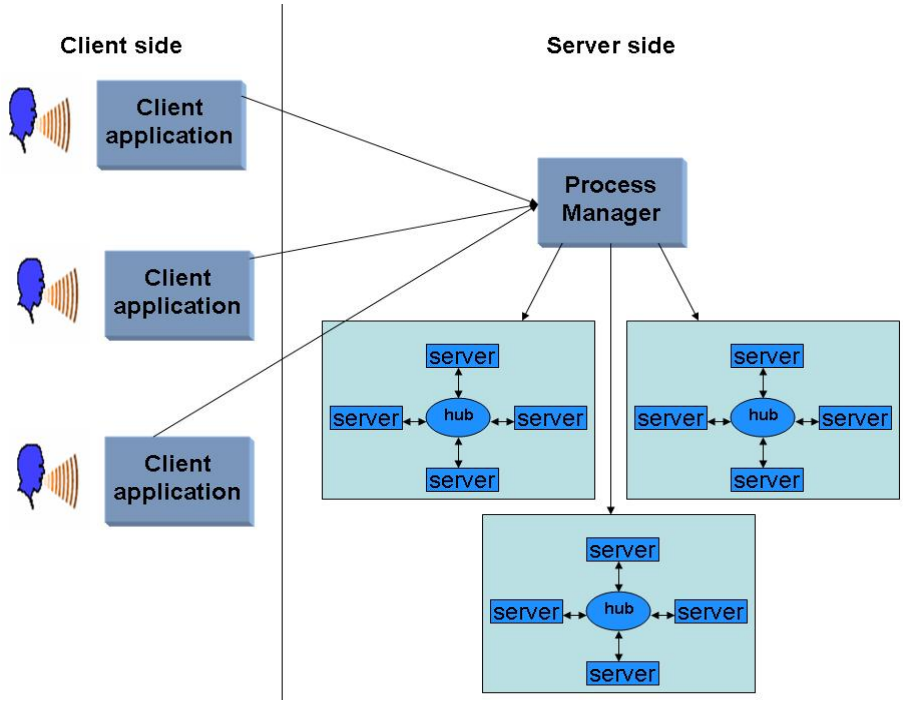

Figure 11 The process manager managing different users requesting different HLT applications.

The process manager starts these processes by encapsulating them in a Java™ Process Object. The Java™ Process Object enables the process manager module to control all the server processes. The Process Object gives a host of capabilities by which the processes can be monitored. Java provides methods to send inputs to the process, pipe an output stream from the process, and detect errors that occurred in the process. It even provides a "wait for" feature where the current thread that is running waits until the process is executed (i.e., the main thread is blocked until the process has finished executing). By tapping into these capabilities, the process manager can create a process, wait on a process, perform input/output on the process and also check the exit status of the process. If a server process fails for any reason, the process manager detects the failure and terminates the cluster of servers associated with the failed process. It also

sends a message to the corresponding client application forcing the user to close an application and restart it.

As can be seen in Figure 11, in a multi-user environment, many processes are running and communicating with each other through sockets [48]. One major issue in such an environment is port allocation. Any two servers trying to listen to the same port may lead to server failures or unpredictable behavior of the applications. The process manager handles port allocation by making sure each new process created listens to a port number unique to itself.

3.2.2 Common Application Interface

Support for multiple applications required providing a common interface from which users could select an application of interest [47]. The demo selector module was designed to provide the desired interface and coordinate with the process manager module to start the required servers. The demo selector interface displays a single screen with icons for each of the four applications. Once the user selects an application, the demo selector loads and displays the appropriate user interface. Though each user interface is designed to fit the needs of the specific application, they each share common modules including the display module and the configuration menus which allow the user to change the default settings of the application.

Figure 12 shows the demo selector interface for the four applications, superimposed with the user interface for the speech analysis application, after it has been selected. The client program sends a Communicator frame with a key-value pair containing the name of the application that was selected. The process manager has prior

information about each application and the corresponding servers needed to run the particular application. Upon receiving the message in this frame, the process manager extracts the application name from the Communicator frame and starts the required servers. Once the user closes a certain application, the demo selector window is displayed again. The user can either choose another application or simply exit from the interface.

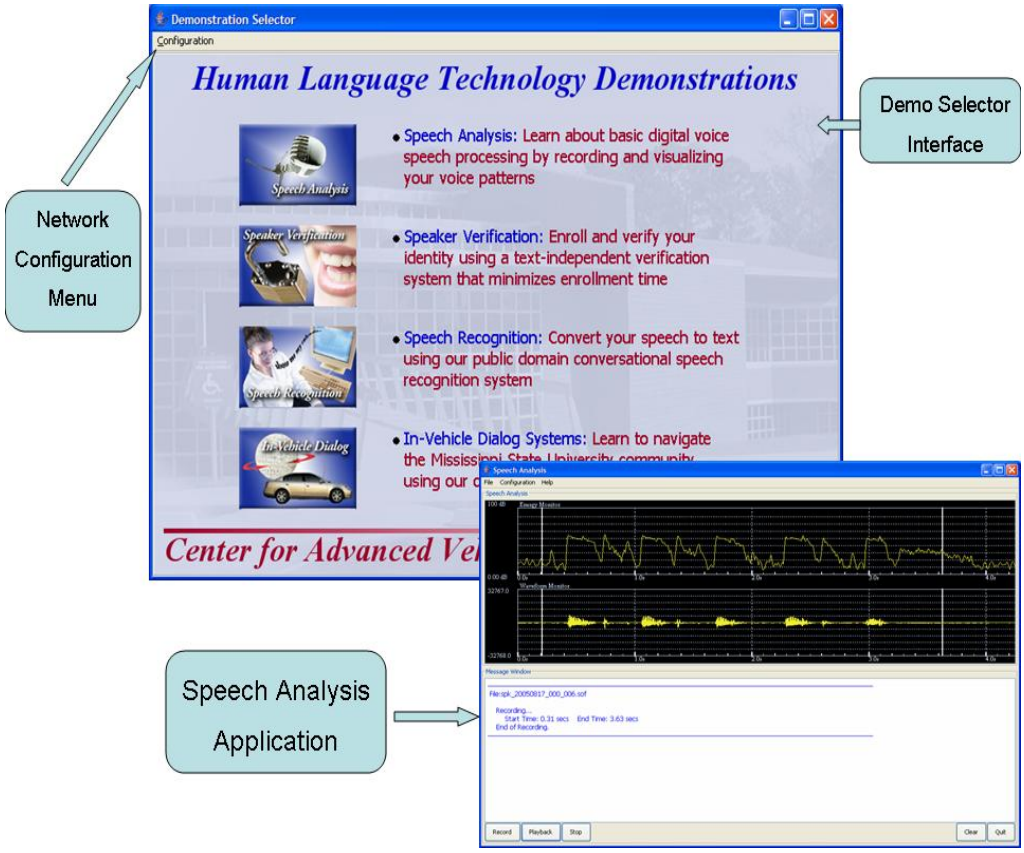

Figure 12 Demo selector and speech analysis user interface.

The demo selector also has a network configuration menu as referenced in Figure 13. The client application must have the IP addresses of the machine in which the process manager and the hub are running. The network configuration gives the user the capability to change these default settings. As discussed earlier, the port allocation between

different servers are handled by the process manager, but the user must specify the port number through which the client application can communicate with the process manager. Similarly, the user must specify the port number the client application has to listen to for further communication.

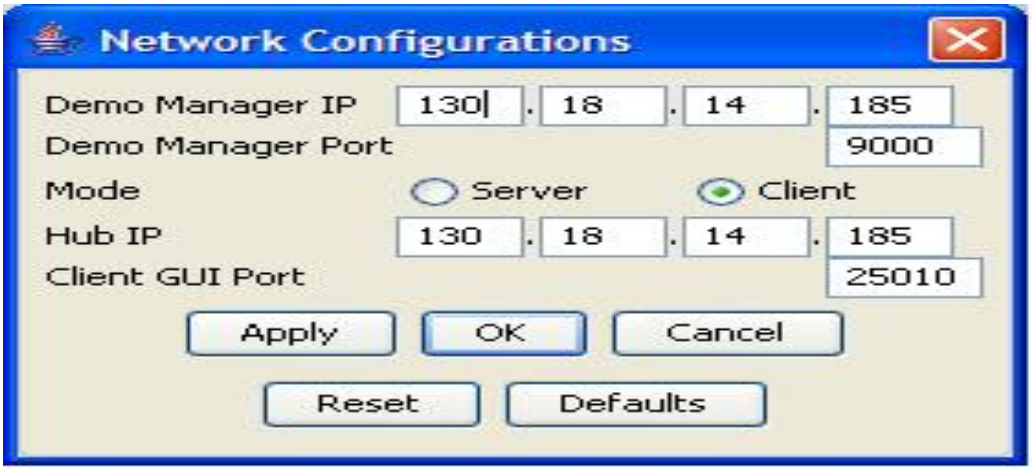

Figure 13 The network configuration menu.

The original Communicator architecture allows a given process to act as a server or a client. This mode can be reversed by using the network configuration window.

3.2.3 Improvements to System Robustness

Improving system robustness with respect to system failure is the primary focus of the thesis. For the foundation of the redesign strategy, a simple application, speech analysis was targeted. The approach taken for the demonstration system entailed using the implicit capabilities of the Communicator to enhance reliability of inter-process communication between clients and servers [49]. This section describes how a state machine architecture [50], [51] was implemented to support a basic handshaking protocol

between the client and servers using frames. Figure 14 shows an overall view of the client-server modules for speech analysis. Note that even this simple application requires two servers, audio recorder and signal detector.

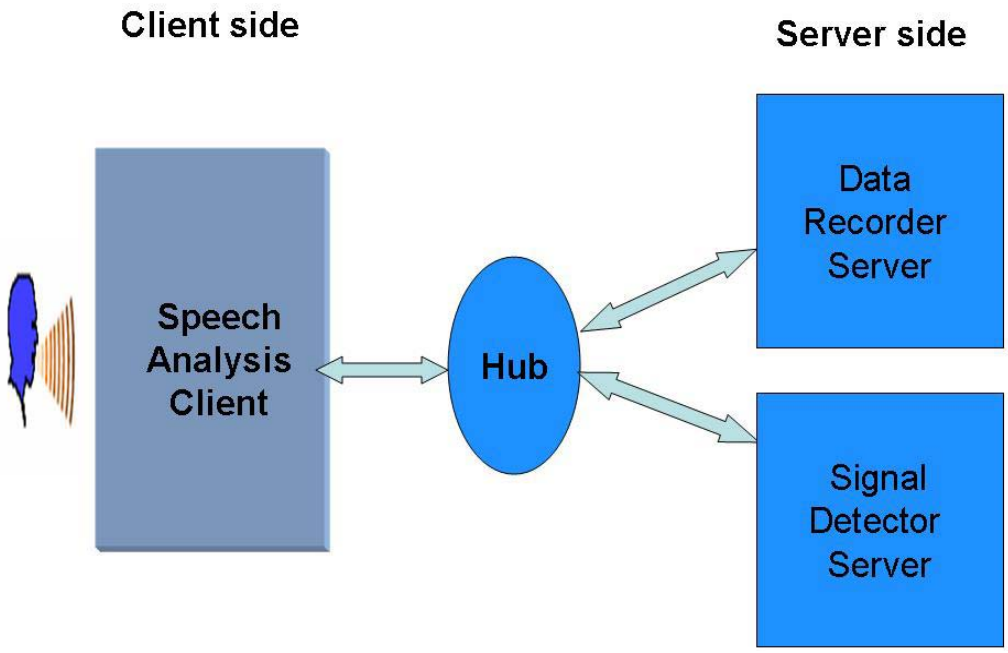

Figure 14 Speech analysis application (client and server).

Figure 15 shows the state machine architecture and basic handshaking supported between the speech analysis client and the signal detector server. A simple handshaking protocol was implemented with signals and acknowledgements, each implemented as Communicator frames sent via the hub. The states and handshaking protocol support three major interaction phases between client and server, 1) preparing for data transfer; 2) data transfer itself, and 3) end of data transfer. For Phase 1, the client begins in the Initialization state, during which it establishes a connection with the hub. It then transitions to the Audio_Ready state and sends an Audio_Ready signal to the signal

detector server to prepare it for audio data transfer. The client then waits for an acknowledgement of the Audio_Ready signal from the signal detector server, and once it is received, it transitions to the Audio_Ready_Ack state.

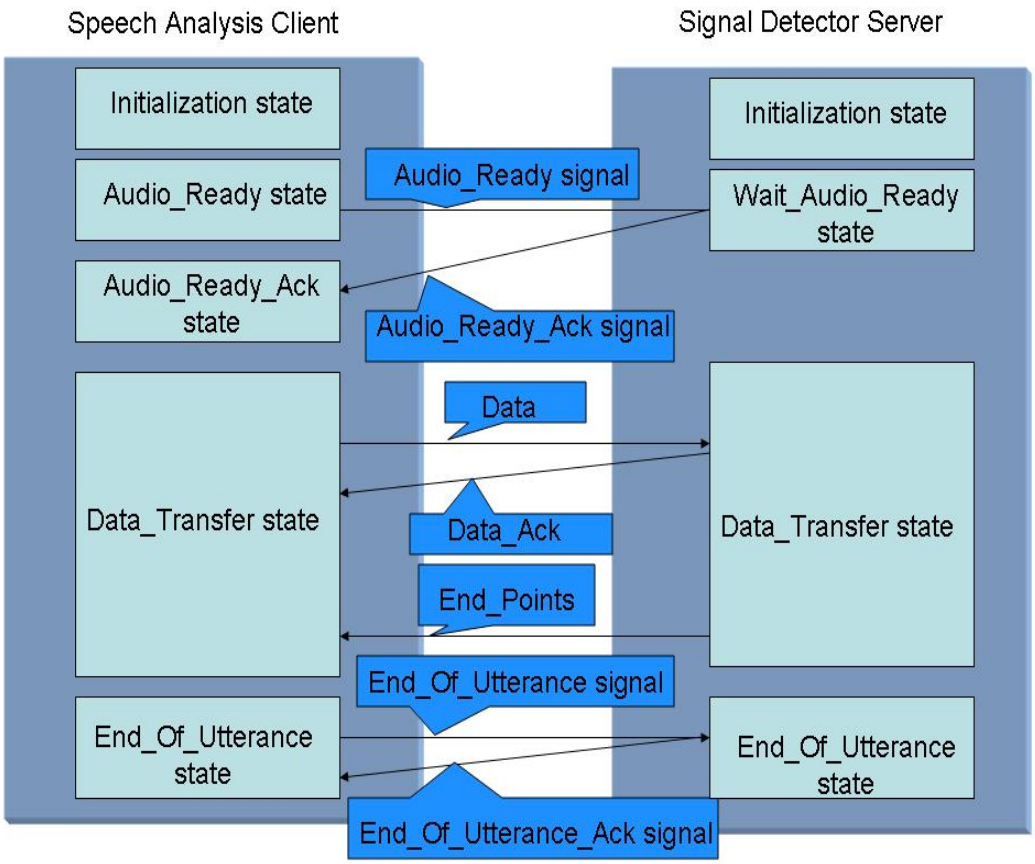

Figure 15 Handshaking between the speech analysis client program and the signal detector server.

In Phase 2, data transfer begins when the client transitions to the Data_Transfer state and sends packets of audio data in Communicator frames to the server. For each frame of data sent, the client waits for an acknowledgement from the server, which checks each for validity. If the server receives a frame that is invalid, it does not send an

acknowledgement signal, but generates an error message, written to a log file. The client will not send further data until it receives an acknowledgement.

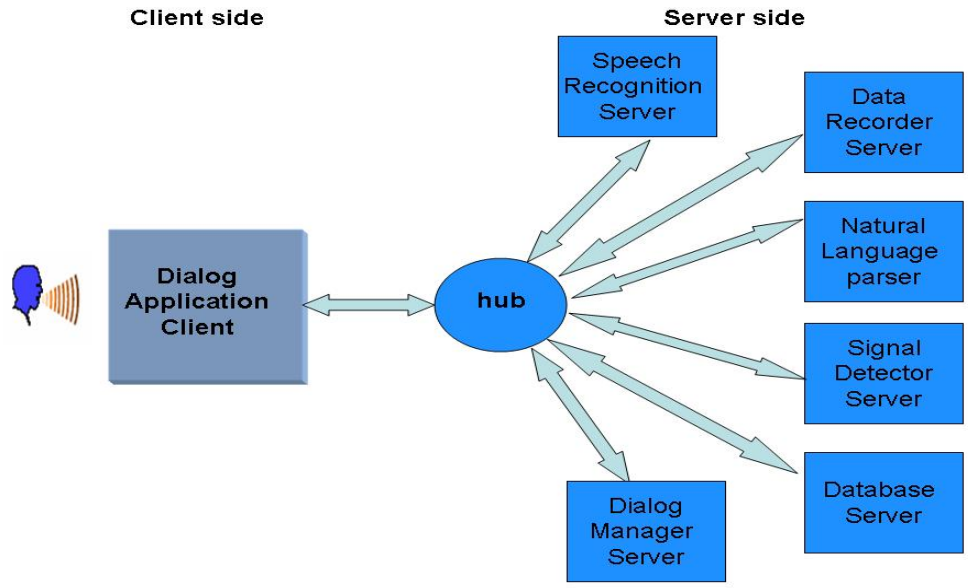

Figure 16 Block diagram of the dialog system.

If data transfer completes successfully, the signal detector server detects endpoints and passes the endpointed data to the client. The client then sends an end of utterance signal to the signal detector server and waits for an acknowledgement. On receiving the end-of-utterance signal, the signal detector server sends an acknowledgement signal to the client and resets itself to the initial state. The handshaking protocol described in this example is implemented for all applications and has eliminated server failures and deadlocks due to communication errors.

The abovementioned mechanism that was illustrated for a simple application was extended to more complex applications such as the dialog and the speaker verification systems. Figure 16 illustrates a block diagram of the servers involved in the dialog

application. Figure 17 shows the states associated with each of these servers. The speech analysis and the dialog system application have similar communication patterns during the recording process. Once the recording ends, the speech recognition server transitions to the Data_Processing state and decodes the utterance. Once the utterance is decoded, the decoded text is sent to the natural language parser and the speech recognition server resets to its initial state.

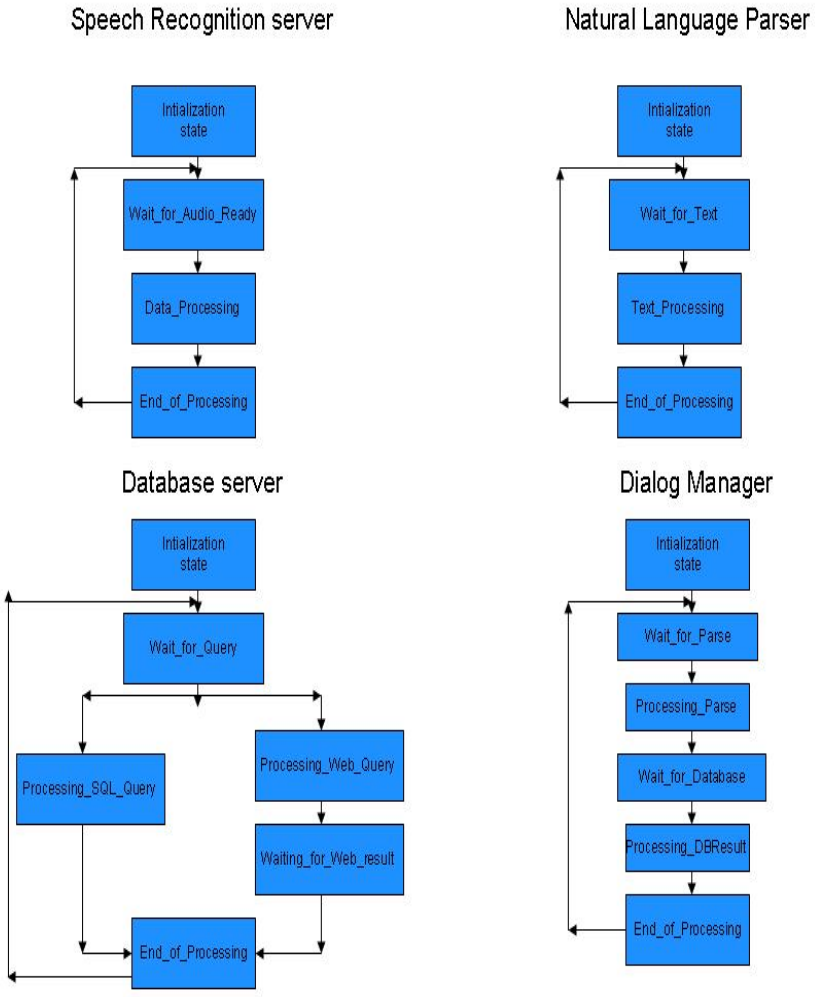

Figure 17 The state machine architecture of the dialog system servers.

The parser transitions to the Text_Processing state and computes a parse for the decoded utterance. Usually, the parser can generate more than one parse for a query. The parser can be run in an N-best mode where a list of best parses are generated and the dialog manager is used to select the best parse depending upon a given criteria. The parser sends the parsed output to the dialog manager and resets to the initial state. The dialog manager computes the best parse after transitioning to the Processing_Parse state. Once the query is formulated, the dialog manager transitions to the Wait_for_Database state and sends the query to the database server.

The database server branches to Processing_Sql_Query or Processing_Web_Query depending on the query type. Once the database server sends the response to the dialog manager, it resets to the initial state. The dialog manager receives the database response and transitions to the Processing_DB_Result state. The dialog manager sends the response to the user interface and resets to the initial state. This systematic handling of communication has improved the robustness of complex applications such as the dialog system.

In this chapter, the modular and the architectural enhancements that were made to the original architecture were discussed in detail. In order to evaluate the robustness improvements achieved by these enhancements, experiments were designed to formally evaluate and compare the performance of both the architectures. Chapter IV discusses the experiments that were conducted to evaluate the enhanced architecture. The results of these experiments are further analyzed and inferences about the efficacy of the architecture are drawn.

CHAPTER IV

RESULTS AND ANALYSIS

This chapter analyzes the enhancements made to the original DARPA architecture. The first section describes experiments that were conducted to measure the improvement in the robustness of the architecture due to the enhancements. The second section presents scenarios that demonstrate better error handling and debugging capabilities.

4.1 Quantitative Analysis

The following section consists of four experiments that were conducted to measure the quantitative improvements in the robustness of the system. The first experiment consists of comparing the results obtained by testing utterances from the extended pilot database on the original and the enhanced architecture. In the second experiment, a set of tasks were randomly selected from a pool and tested on the original and the enhanced architecture. The third experiment consists of tasks performed under a series of scenarios by the user for specific time duration on both architectures. Robustness improvements in the enhanced architecture are measured by comparing the number of interactions that were successfully completed. The fourth experiment consists of users performing tasks in specific scenarios using the dialog system with spoken language input, the most complex application interaction in the HLT system.

4.1.1 Pilot Corpus Experiment

As mentioned earlier, our initial prototype system consisted of automatic speech recognition (ASR), natural language processing (NLP), dialog management (DM), and a database back-end. The language model and grammar for the ASR and NLP systems were derived from a pilot corpus that consisted of 276 queries spontaneously entered by users over a series of three experiments. After initial prototyping, a series of pilot experiments were conducted on the original DARPA architecture. These pilot experiments consisted of first testing the system on the collected data, then making the necessary modifications to the grammar/language model and retesting the system.

During this phase, the NLP system was iteratively refined with a simulated ASR system using a series of Wizard of Oz (WOZ) experiments [52], [53]. The refinements showed improvements on the error rates, especially for the utterances containing out of vocabulary words (OOVs). The pilot corpus was extended by adding the utterances collected during the WOZ experiments. These refinements were tested using 403 utterances from the extended pilot corpus which spanned 10 different categories that included Address (98), Direction (219), Distance (23), List of places (36), Building (10), Turn (5), Bus (7), Intersection (2), Which Way (2) and Special (1). During these experiments conducted for the original architecture, approximately 4% of the utterances resulted in a server error or a deadlock.

Procedure

Once the enhancements discussed in Chapter III were made to the DARPA architecture, each of the 403 utterances from the pool was retested using the enhanced

architecture. These utterances were tested by one non-native male speaker with the dialog system running in text mode. All the utterances were successfully queried using the enhanced architecture while only 386 utterances were successfully queried in the original architecture. Table 1 shows the server errors and deadlocks in the original and the enhanced architecture. In the original architecture, it can be noted that approximately 4% of overall failures occurred due to system failures and deadlocks.

Table 1 Performance data for the dialog application.

		Before Enhancements			After Enhancements		
Queries	# of utter-ances	Passed (%)	Failed (%)		Passed (%)	Failed (%)	
			Server Errors	Deadl-ocks		Server Errors	Deadl-ocks
Address	98	100.00	0.00	0.00	100.00	0.00	0.00
Direction	219	95.43	2.28	2.28	100.00	0.00	0.00
Distance	23	91.31	8.70	0.00	100.00	0.00	0.00
List of places	36	100.00	0.00	0.00	100.00	0.00	0.00
Building	10	100.00	0.00	0.00	100.00	0.00	0.00
Turn	5	100.00	0.00	0.00	100.00	0.00	0.00
Bus	7	57.15	42.85	0.00	100.00	0.00	0.00
Intersе-ction	2	0.00	100.00	0.00	100.00	0.00	0.00
Which way	2	100.00	0.00	0.00	100.00	0.00	0.00
Special	1	100.00	0.00	0.00	100.00	0.00	0.00
Total	403	95.78	2.97	1.24	100.00	0.00	0.00

Conclusions

Results for the enhanced architecture show a reduction in server errors and deadlocks. Although server errors and deadlocks were eliminated on this specific test set,

this clearly cannot be argued in general. Nonetheless, it demonstrates the overall occurrence of errors has been reduced and further, handling of errors is improved.

For example, for the address query “Give me directions from Bryan Field to Hunter Henry Center”, the dialog manager fails as it does not have the capability to handle this query. In the original architecture, this server error leads to a system failure. In the case of the enhanced architecture, the process manager detects the error and reports these errors to the client process. The enhanced architecture also takes the necessary steps to restart the servers. Thus the server errors are gracefully handled by preventing a failure of the entire system.

One limitation of the experiment is that it tested the system against baselines established early in the original architecture development using only text mode (i.e., the NLP modules). Though necessary to test against these established baselines first, these baselines are not sufficient results to fully measure overall robustness improvements, including for example, those for the signal detector, data recorder and speech recognition servers. The data transfer stage requires more inter-process communication and is thus vulnerable to inter-process communication errors; therefore experimental testing of these features is imperative.

The utterances used in these experiments were collected from users by giving them a general scenario and asking them to fill the details. A sample scenario is shown below:

> “You’ve arrived at the Golden Triangle Airport, gotten a rental car and must get to your first meeting of the day. Your meeting is at the _____. You’re at the airport exit onto Highway 82. “

The user was given a list of places from which to choose for the meeting place in the above scenario. Due to this restriction, all these queries had only place names, hotel names, and restaurant names for which the system had prior knowledge. This limitation was overcome in the following experiments by allowing the user to freely query the system.

4.1.2 Task Pool Experiment

The second experiment consists of one speaker performing a set of tasks which were randomly selected from a task pool. In this experiment, a task consisted of one or more interactions of the user with the system. An example of a task is “Use speech mode in the dialog system to query the distance between two places.” The pool includes all tasks from anticipated and observed usage over the development of the system. Of most importance to the experiment, it includes two major categories:

1) Tasks which were hypothesized to result in server errors which will lead to system failures for the original architecture, but hypothesized to generate just server errors under the enhanced architecture. These tasks were basic recording and querying tasks which were performed under certain system specifications [see Appendix A]. Examples of the system specifications include trying to write to a location that does not exist, receiving an inappropriate frame during data transfer, a buffer overflow during data transfer and attempting to access a null frame. There were eight such tasks in the pool.
2) Tasks that were hypothesized not to result in server errors or system failures under neither the original nor the enhanced architectures. Examples of these tasks

included recording tasks for varying time durations and a wide range of address-querying tasks. Figure 18 shows the sub categorization of these tasks. There were 30 such tasks in the pool.

The second experiment overcame the limitations of the previous experiment by allowing a wide range of tasks which tested both the spoken and natural language processing capabilities of the system. The experiment was conducted by one user who had prior experience using the HLT system. Unlike the first experiment, the user did not have any constraints or prior information guiding him/her through these tasks.

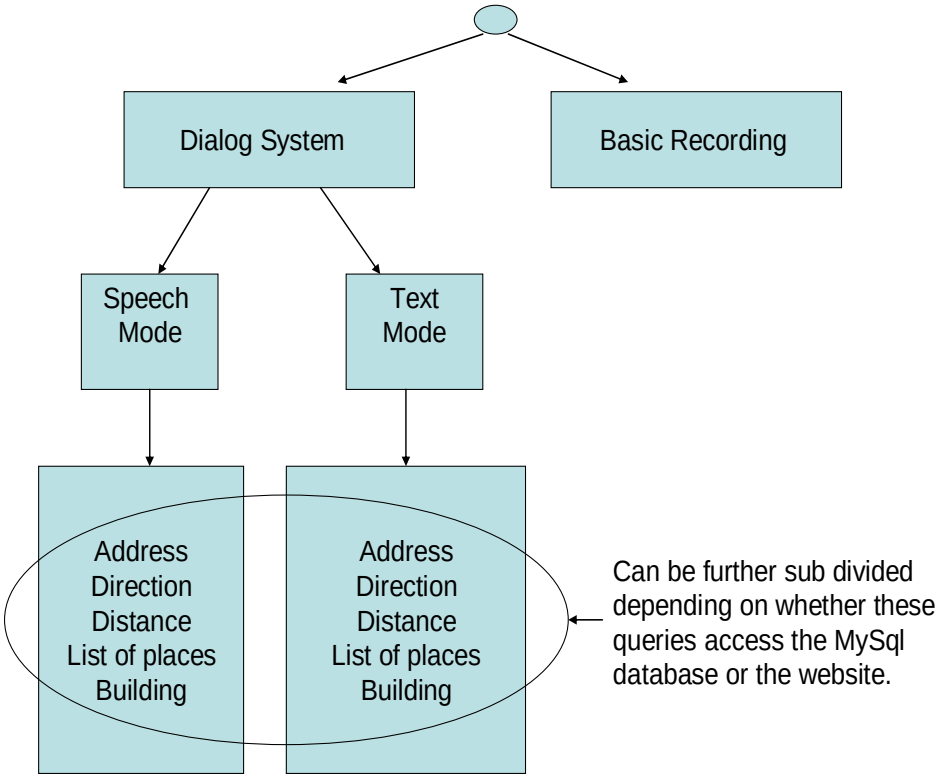

Figure 18 Categorization tree of the scenarios.

Procedure

Again, the pool consisted of 38 total tasks which tested a wide range of capabilities. The tasks were numbered in order [see Appendix A] and a random number generator was used to select a number that corresponds to a specific task on the list. For example, consider that the number 10 was randomly generated on a certain trial. The number 10 corresponds to a recording task for time duration of 5 seconds [see Appendix A], which the user performed. This process was repeated until 30 trials were performed. These trials were performed by one non-native, male speaker. In this experiment each task corresponds to one interaction between the user and the system.

Results and Analysis

All 30 tasks passed the enhanced architecture. In the case of the original architecture, 24 tasks passed while the other 6 resulted in a server error. The 24 tasks that did not generate errors under the original architecture included those for recording for varying time durations and querying the dialog system. The six tasks that failed under the original architecture involved programmer errors which led to a system failure.

One such task consists of basic recording under the system specification that the signal detector server receives an inappropriate frame during data transfer. This inappropriate frame can be defined as any frame that does not contain audio data during the data transfer stage. This inappropriate frame can be received by the server due to inter-process communication error or a programmer error in setting the hub rules. In the original architecture, the signal detector server tries to extract data from the frames. Since the inappropriate frame does not have any audio data, the server errors and exits.

Because there is no functionality in the original architecture to detect the server error, a system failure occurs.

For the enhanced architecture as the states are well defined, the signal detector server checks whether the frame contains audio data before it is extracted. Even in the worst case scenario, if the server errors and exits, the process manager provides a graceful handling of the error and prevents a system failure.

Conclusions

As can be seen in Table 2, 24 tasks passed the original architecture while six failed. The six tasks that failed belonged to task set from the pool that was expected to fail in the original architecture. This experiment confirmed that these tasks did fail in the original architecture. In this experiment, a random number generator was used to produce an unbiased selection of tasks from the pool. Though this removes a level of bias, it is not based on observed system usage, and, as such, does not necessarily capture typical usage patterns. The tasks which failed could constitute a greater percentage of typical daily usage. Another limitation of this experiment was that the system was tested by a single user.

Table 2 Performance results for task pool experiment.

	Number of tasks that passed the test	Number of tasks that failed the test
Original architecture	24	6
Enhanced architecture	30	0

4.1.3 General Usage Scenarios Experiment

In this experiment, five different users performed tasks pertaining to 24 usage scenarios. The terms scenario, task and interactions are used frequently in this section, thus their meanings should be clearly defined. A scenario is a general situation under which the user is asked to use the system. A scenario may require performing one or more tasks. For example, consider a user is planning a vacation to the city of his/her choice. She needs to decide on a travel itinerary by using the system. This is a general scenario and the user may choose to accomplish this by performing several tasks using the system.

Each task may require a single query or multiple queries to accomplish. This is referred to as an interaction which is defined as one response from the system to accomplish a specific task. All results in the following experiments have been tabulated in terms of the number of interactions. All participants of this experiment were first time users of the HLT system and had little or no knowledge of this technology. Users were not restricted in their queries so that the queries more closely resemble the usage patterns of a typical user.

Procedure

Five users were asked to engage in 24 usage scenarios using the original and the enhanced architecture [see Appendix B]. Among the five users, there were three males and two females. The user pool consisted of one native speaker and four non-native speakers. These usage scenarios required performing such tasks as recording for varying time durations and querying for information. The scenarios were carefully drafted not to prompt the user for a specific query. For example, the user is asked to role play that she is

attending a conference in a big city, and to assume she wants to visit sites of interest after that day's conference proceedings. She has no prior knowledge about the layout of the city or a city map. She is asked to use the system to plan her visits.

Before participating in the experiment, each user was presented with a set of instructions, which introduced them to the system [see Appendix C]. Each user was allowed a 10-minute practice session to get familiar with the functionality of the system. The practice session included basic recording sessions and dialog tasks related to some predefined queries that the user can use to gain familiarity with the system. Once the practice session was over, the user engaged in the usage scenarios for the experiment, using both architectures, with a time limit of 30 minutes for each. The user performed the tasks, first on the enhanced architecture followed by the original architecture to prevent any robustness improvement trend that may occur due to user's familiarity with the system. The entire experiment took approximately 1 hour and 30 minutes. The user was asked to cease testing if there was a system failure or she exceeded the allotted time of 30 minutes.

Results and Analysis

Table 3 tabulates the number of interactions that were successfully completed by each user, for each of the architectures. It can be noted that a total of 129 interactions were successfully completed using the enhanced architecture while only 76 interactions could be performed successfully using the original architecture.

Table 3 The number of interactions that passed the original and the enhanced architecture in general usage scenarios experiment.

Users	Number of interactions that were successfully attempted in the Enhanced architecture	Number of interactions that were successfully attempted in the Original architecture
User 1	22	23
User 2	26	24
User 3	23	**10**
User 4	**32***	**11**
User 5	26	**8**
Total number of interactions that were successfully attempted	129	76
* indicates that a server error was experienced but the enhancements prevented a system failure.		

The entries that are highlighted in Table 3 failed during the testing process. The system failed three times due to a server error in the original architecture. A server error was experienced once during use of the enhanced architecture but was appropriately handled by the process manager to prevent a system failure. The scenarios that led to a system failure were reconstructed and analyzed. The server errors that occurred during the experiment were traced to two types of scenarios. They included:

1) Inter-process Communication Error: While the user records audio data, the signal detector detects the endpoints. The speech recognition server decodes the endpointed audio data and the decoded text is sent to the natural language parser. The dialog manager receives the parsed output and queries the database for a response. If no record was found, the Mapquest website is queried. While the dialog manager was querying the Mapquest database, the user attempted to record

another utterance which in turn triggered a new set of communication. The dialog manager failed as it was in a different state still querying for a response. The system failed twice during execution of this scenario using the original architecture. Even though this scenario occurred once during use of the enhanced architecture, the process manager detected the error and prevented a system failure.

2) Hub Connection Error: The hub experienced a connection error while contacting the servers. The DARPA Communicator documentation states this error can occur if the hub cannot connect to a server or if the number of connections has exceeded the maximum value. This is an internal error with the Communicator's hub. Even though this error did not occur during the testing of the enhanced architecture, the process manager should be able to gracefully handle this error.

In order to further analyze the results, the 24 usage scenarios were categorized on the basis of its purpose. Figure 18 shows the sub-categorization of these scenarios. Though the scenarios were carefully written to avoid prompting or biasing the user to issue a specific query, they were also crafted to elicit and test all capabilities of the system, from basic recording to the most complex dialog response capabilities. The scenarios can be categorized into the following major categories:

1) Basic recording capabilities
2) Dialog response capabilities - speech mode
3) Dialog response capabilities - text mode

Among the 24 different scenarios, six scenarios belonged to the "Basic Recording" category and the other 18 scenarios belonged to the "Dialog system: Speech/

Text mode" category. The data presented in Table 4 shows no improvement for the "Basic Recording" and "Dialog system: Text mode" category as no system failure was experienced in these categories for either architecture. The data only indicate robustness improvements in the "Dialog system: Speech mode" category using the enhanced architecture.

Table 4 Three categories of experimental data.

Catego-ry	Number of interactions successfully attempted by User 1		Number of interactions successfully attempted by User 2		Number of interactions successfully attempted by User 3		Number of interactions successfully attempted by User 4		Number of interactions successfully attempted by User 5	
	E	O	E	O	E	O	E	O	E	O
Basic record-ing	9	8	8	6	7	6	12	8	9	8
Dialog system: Speech mode	8	7	9	9	9	4	12	3	9	0
Dialog system: Text mode	5	8	9	9	7	0	8	0	8	0
Total number of interact-ions	**22**	**23**	**26**	**24**	**23**	**10**	**32**	**11**	**26**	**8**
E – Enhanced architecture , 0 – Original architecture										

To analyze these improvements in greater depth, the scenario categories were further subdivided into different levels of inter-process communication needed to successfully complete an interaction. Table 5 summarizes the number of interactions per

category. Table 6 tabulates the different communication levels needed to complete an interaction in each of the three major categories. It can be noted that among the different categories, the “Dialog system: Speech mode” category is the most complex and accounts for the maximum number of inter process communication exchanges. Therefore, more failures are to be expected in scenarios from the “Dialog system: Speech mode” category compared to other categories.

Table 5 Summary of experimental data for the three categories.

Categories	Enhanced Architecture	Original architecture
Basic Recording	45	36
Dialog system: Speech mode	47	23
Dialog system: Text mode	37	17
Total number of interactions that were successfully attempted	129	76

Conclusions

The results of this experiment have shown an evident improvement in robustness of the enhanced architecture over the original architecture. Table 5 shows that 129 interactions successfully passed the enhanced architecture while only 76 interactions passed the original architecture. This shows a 37% improvement in robustness compared to the original architecture on this specific dataset. Further categorization of these results illustrates that for the “Dialog system: Speech mode” category, only 23 interactions successfully completed during tests using the original architecture while 47 interactions

successfully completed during tests using the enhanced architecture. This shows a 51% improvement in the robustness of the system on this specific dataset.

These numbers may overstate the actual improvement in robustness as the user was asked to abort the experiment following a system failure which prevented him/her from performing the subsequent tasks. Since this was more likely to occur in the original architecture, this could significantly reduce the number of interactions on that architecture. To obtain a more focused measure of robustness improvement, further experimentation was needed to target the "Dialog system: Speech mode" category since it scenarios from this category require execution of the most complex tasks in the HLT system, and results show a notable variation in the performance between the two architectures. Therefore, an additional experiment was required to allow the user to continue performing all the listed scenarios irrespective of system failures.

Table 6 The different stages of communication needed to complete successfully an interaction in three major categories.

Category	List of sub interactions in each category
Basic Recording	1) The user records and the audio data is transferred to the server. 2) The end points are detected and the recording ends.
Dialog system: Speech mode	1) The user records and the audio data is transferred to the server. 2) The end points are detected and the recording ends. 3) The utterance is decoded. 4) The decoder output is parsed. 5) The query response is retrieved and sent to the user interface.
Dialog system: Text mode	1) The user queries the system in text mode. 2) The query is parsed. 3) The query response is retrieved and sent to the user interface.

4.1.4 Dialog System-Speech Mode Experiment

This experiment was designed to target the "Dialog system: Speech mode" category to measure the robustness improvement on the most complex tasks in the HLT system. The limitations of previous experiment were overcome by asking the user to request a system restart in the event of a system failure and to continue testing for the full 30 minutes.

Procedure

Five users were asked to perform nine usage scenarios restricted to the "Dialog system: Speech mode" category [see Appendix B]. Among the five users, there were four males and one female. The user pool consisted of one native speaker and four non-native speakers. Each user was provided with a series of scenarios originating from the user's visit to Starkville from his/her city of residence. The users were initially presented with a set of instructions, which introduced him/her to the system [see Appendix B]. The user was allowed to take a 10-minute practice session to get familiar with the functionality of the system. The user performed tasks from nine different scenarios with a maximum time duration of 30 minutes per session on each architecture. The user first performed the tasks on the enhanced architecture followed by the original architecture to prevent any robustness improvement trend that may occur due to user's familiarity with the system. In case of a system failure, the user sought assistance in restarting the application and continued testing the system.

Results and Analysis

Table 7 tabulates the number of interactions that could be completed successfully using each of the architectures. The system experienced a server error twice during use of the enhanced architecture but the process manager module prevented a system failure. The system failed three times during experiments in which the original architecture was used. The scenarios for these failures were reconstructed and are discussed in detail. The two error scenarios were:

Table 7 The number of interactions that passed the original and the enhanced architecture in dialog system-speech mode experiment.

Users	Number of interactions that passed successfully the Enhanced architecture	Number of interactions that passed successfully the Original architecture
User 1	10	**8**
User 2	10	11
User 3	**9***	9
User 4	**16***	**13**
User 5	10	**10**
Total number of interactions that passed successfully	55	51
* indicates that a server error was experienced but the enhancements prevented a system failure.		

1) Inter-process Communication Error: This is the same error that was mentioned in the analysis section of previous experiment. This error occurs when the user attempts to record when the dialog manager is still attempting to respond to the previous query. This occurred once during use of the original architecture which led to a system failure. Though this occurred during use of the enhanced

architecture, the process manager provided a graceful handling of the error and prevented a system failure.

2) Dialog Manager Error: The dialog manager errors and exits while trying to send the query results to the client process for display to the user. Once the database server returns the query results for the direction query, the dialog manager wraps the results in a Communicator frame and sends it to the hub. In this specific case, the query result contained more than 200 lines of instructions which were retrieved from the database tables. The string that holds these results were not dynamically allocated to fit any size and resulted in a failure of the dialog manager. This can be classified as a programmer error. This error was experienced twice during the testing of the original architecture. This error occurred once in the enhanced architecture and was gracefully handled by the process manager.

4.1.5 Conclusions on Quantitative Analysis

To obtain a quantitative measure on the robustness improvements, 10 different users were asked to perform approximately 200 interactions on each of the architectures with a total experimentation time of around 10 hours. There were two native speakers and eight non-native speakers in the user pool. Among the 10 different users there were seven males and three females. On the fourth and final experiment, 55 interactions were completed successfully using the enhanced architecture while only 51 interactions were completed successfully using the original architecture. All experiments were designed using scenarios carefully crafted to elicit the most natural, spontaneous interaction from

users on the widest range of system functionality. Quantitative results of the final experiment show a 7.2% improvement in robustness on the most complex set of tasks defined for the HLT system. It can be concluded that the enhanced architecture has provided a lower bound of 7% improvement in the robustness of the system.

Though allowing users exposure to the system continuously for longer time periods would yield additional data, this would not necessarily yield more meaningful data without carefully designed and controlled experiments,. All facets of this experimental design, including subject selection techniques and scenario design and presentation, can serve as a critical foundation for more comprehensive studies.

4.2 Qualitative Analysis

Most of the enhancements to the DARPA architecture were developed out of necessity for a better error handling and debugging capabilities. This section discusses the various qualitative enhancements made to the original architecture and describes two scenarios where these enhancements have improved error handling and debugging capabilities of the system. Although most of the enhancements discussed in Chapter III contribute in some way to the qualitative enhancement of the system, discussed below are the two main qualitative enhancements to the system. They include:

- Process Manager Module: The process manager is a powerful module that enables automated server management. The ability of the process manager to keep track of servers and handle port allocation has provided a better platform for spoken language applications. The enhanced architecture provides a built-in capability to handle multiple users, which was not supported in the original architecture. The demo

selector interface provides a simple interface for the user to choose from a host of applications, while the original architecture needed manual assistance to accomplish this capability.

- Improved Debugging: The enhanced architecture has provided better logging of communication which, along with the state machine architecture and basic handshaking capabilities, has provided a more efficient debugging paradigm for the system. Each server including the hub logs all the communication it sends/receives and also logs information related to states and subsequent state transitions. These logs can be used to reconstruct the specific scenario in case of a server or system failure. The debug window, a component of the user interface module, records all communication that is routed through the hub. This provides an excellent interface for the user to debug the system when she has no access to the log files on the server side.

To demonstrate these capabilities, two scenarios are presented that illustrate improvement in error handling and debugging capabilities of the system.

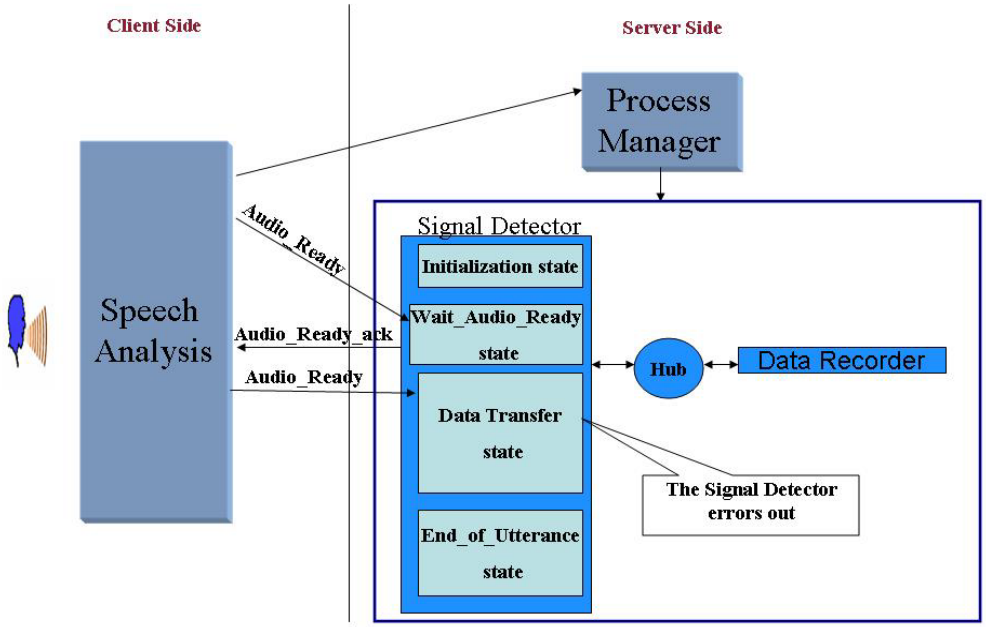

Figure 19 The process manager handling server errors.

4.2.1 Server Management and Error Handling

Figure 19 illustrates a case where the signal detector errors out due to an inter-process communication error. In this scenario, the signal detector server receives two Audio_Ready signals which occurred due to a programming error. Initially, the user starts a speech analysis application and the client process contacts the process manager to start the required servers. On client's request, the process manager starts the data recorder and the signal detector server along with the hub. Once the recording starts, the client process sends the Audio_Ready signal to inform the servers that the recording has started.

As mentioned earlier, in this particular scenario, the signal detector server receives two Audio_Ready signals. This can happen due to a programming error in the client process or because the hub script has been inappropriately programmed to send two Audio_Ready signals to the signal detector. When the signal detector receives the first

Audio_Ready signal from the client process, it sends back an Audio_Ready_Ack signal and transitions to Data_Transfer state. Due to a programming error, the signal detector gets another Audio_Ready signal. Given the state of the server, the state machine architecture in the signal detector server detects the received signal as an inappropriate signal. The signal detector server errors and exits with an appropriate error message. The process manager detects the server error, terminates the associated processes and informs the client to restart the application. Thus, the process manager prevents a system failure and provides a graceful handling of the errors.

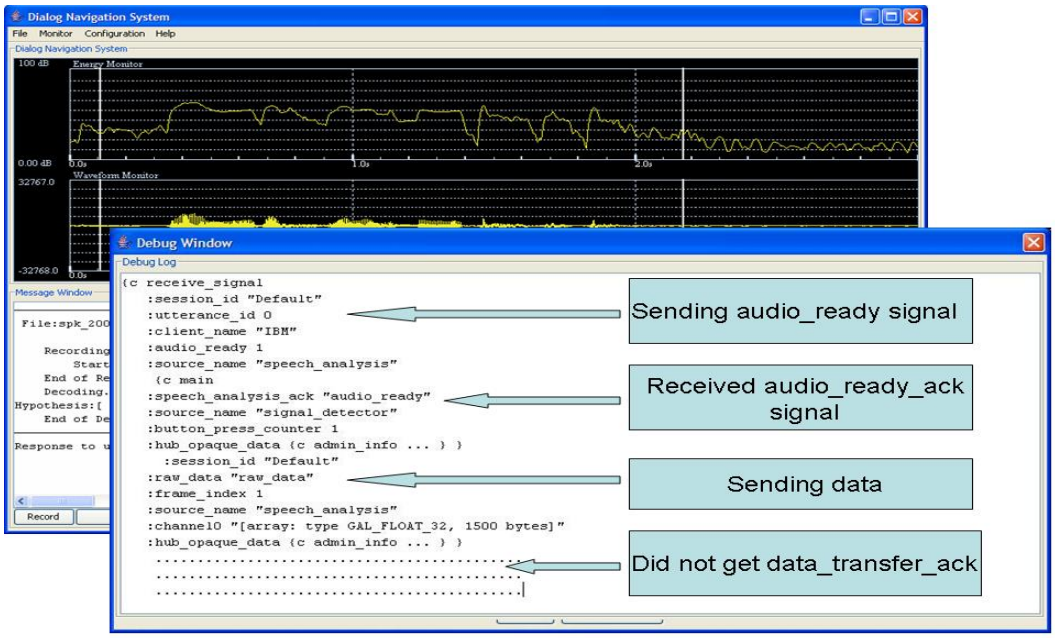

Figure 20 A debug window showing an audio data transfer error.

4.2.2 The Debug Window

The debug window was designed as an integral part of the user interface to help the user debug the system from the client process. This is a critical feature when the user

is not at close proximity to the server machine or she does not have authorized access to the server machine. Figure 20 illustrates a scenario where an inter-process communication error occurs during data transfer.

During the data transfer, the recipient server acknowledges every packet of data sent by the client process. In this scenario, an inter-process communication error occurs as the recipient server could not respond with an acknowledgement or the acknowledgement did not reach the client process. The user can view the debug window and browse through Communicator messages to reconstruct the exact scenario that led to the failure. Thus the debug window provides a debugging interface for the user which never existed in the original architecture.

4.2.3 Conclusions on Qualitative Analysis

The scenarios discussed above have illustrated the various qualitative enhancements performed on the original architecture. The qualitative enhancements include first a multi-user and multiple application capability that were not available in the original architecture. Therefore, these two enhancements cannot be evaluated against a baseline. Nonetheless, these capabilities clearly extend the complexity of applications that can be deployed, and thereby, the fundamental research issues that can be investigated using this architecture.

The enhancements related to better debugging capabilities were achieved through rigorous design meetings and reviews. In addition, the user interface was designed with a team including experts in human computer interaction and graphic design. Further, both the debugging and user interface enhancements were reviewed and evaluated by two

categories of users respectively, 1) software developers programming this technology and 2) principle investigators who presented these technologies to research sponsors. Additional feedback and evaluations will be collected from other end users of the HLT system as well as developers who will apply this enhanced architecture for future software development.

4.3 Overall Conclusions

The quantitative results discussed in this chapter provide evidence that the enhancements to the original architecture have improved the robustness of the system. The results show a 7.2% improvement in robustness on the most complex task in the HLT system. The qualitative enhancements, which may not directly contribute to robustness, have contributed to improvements in the overall functioning of the system. Therefore, though more difficult to quantify, it can be viably argued that these enhancements to the original architecture have also indirectly contributed to enhanced system robustness.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

The fundamental modularity and extensibility of the DARPA Communicator architecture marked a new era in HLT research and significantly advanced the complexity of problems which could be studied. It also provided a capability for evaluating and comparing research results that did not previously exist. Nonetheless, it suffered many critical robustness issues. In addition, a multiple multi-user application capability was innately supported. This thesis has identified and addressed these issues and experimental analysis discussed in Chapter IV has shown a 7.2% improvement in robustness on the most complex task in the HLT system.

5.1 Thesis Contributions

As discussed in Chapter IV, essential qualitative and quantitative enhancements were implemented. The key contributions of the thesis include:

- The robustness of the system has shown significant improvements under the enhanced architecture. Chapter IV discusses a series of experiments that were conducted to measure the improvement in robustness of the system. The experiments show an improvement of 7.2% on the address querying task which is the most complex task in the HLT system.
- The initial architecture required the user to possess detailed knowledge about each

application. It involved manual startup of the servers and manual allocation of ports. Modular enhancements such as the process manager have eliminated the need for manual assistance in starting and managing the servers.

- The enhanced architecture provides a carefully designed graphical interface for the user to choose among different applications with a mouse-click; all subsequent tasks, from server startup to port allocation, are automated. When the user exits the application, all related processes are terminated automatically.
- Debugging complex HLT applications has always been challenging. Most of the client applications are multithreaded, making it difficult to retrace the events and isolate the bug. This problem increases in magnitude in a multi-user environment. In the enhanced architecture, servers have been redesigned as state machines with basic handshaking incorporated in the communication between servers. These enhancements have been successful in trapping server errors and provide functionality for effective tracing of potential bugs.

5.2 Future Work

Further experiments should be conducted to obtain additional measures of the robust improvements due to the enhanced architecture. These experiments should include at least 20 additional users unfamiliar with the system and allow the system to respond to user queries continuously for prolonged time periods. These prolonged experiments must be carefully controlled using scenarios that properly exercise system functionality, such as those in the third and fourth experiments conducted for this thesis, so that meaningful data are collected. This data would give more insight into the robustness of the system to

complex inter-process communication for an extended period of time. Due to the constraints of the original architecture, experiments comparing its performance to the enhanced architecture can be performed only on a single-user platform.

Performance improvements to our initial prototype dialog system were made by running a series of pilot experiments followed by a set of Wizard of Oz experiments. Modifications were made to the grammar and the language model that improved the performance of the system to queries with OOVs. Further experiments can be performed to improve the grammar and the language model which will allow the dialog system to handle a wider range of user queries.

Improvements can also be made to the way the context information is currently used in the dialog system. The availability of state-of-the-art statistical techniques has made a significant impact on the way natural language processing works. Statistical parsers have attracted extensive attention because of their performance and ability to adapt to different data sets with ease. However, the availability of data sets to train these statistical models has always proved to be an obstacle. A future enhancement would be to extend the dialog system to accommodate a statistical parser which can be trained on any data set. This feature will expand our query response capabilities.

Under the current architecture, the HLT system runs on a distributed framework where a single client communicates with a single server machine or multiple clients communicate with a single server machine. The process manager has not been tested to manage multiple clients communicating with multiple server machines. The system also needs to be tested on supercomputer clusters. This would enhance application execution

speed as the computational power available for each application would be considerably increased.

In conclusion, this thesis has addressed vulnerabilities in the DARPA Communicator architecture through several important enhancements, including increased system robustness to failure, automated server startup, error detection and correction, support for multiple multi-user applications, and improved debugging capabilities. Future work includes experimentation to validate the enhanced architecture and building other robust, complex, state-of-the-art human language technologies on this enhanced platform.

REFERENCES

[1] K. Hacioglu, B. Pellom, "A Distributed Architecture for Robust Automatic Speech Recognition," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Hong Kong, April 2003.

[2] S. Bayer, C. Doran, B. George, "Dialogue Interaction with the DARPA Communicator Infrastructure: The Development of Useful Software," *Proceedings of First International Conference on Human Language Technology Research*, San Diego, California, March 2001.

[3] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, "Creating Natural Dialogs in the Carnegie Mellon Communicator System," *Proceedings of the European Conference on Speech Communication and Technology*, Budapest, Hungary, September 1999.

[4] W. Ward, B. Pellom, "The CU Communicator System," *IEEE Workshop on Automatic Speech Recognition and Understanding*, Keystone, Colorado, USA, December 1999.

[5] M. Walker, "DARPA Communicator Dialog Travel Planning Systems: The June 2000 Data Collection," *Proceedings of the European Conference on Speech Communication and Technology*, Aalborg, Denmark, 2001.

[6] "The SRI Communicator System," *http://www.ai.sri.com/~communic/*, Speech Technology and Research Laboratory, SRI International, Menlo Park, California, USA.

[7] C. S. Liu, H. C. Wang, C. H. Lee, "Speaker Verification using Normalized Log-likelihood Score," *IEEE Transactions on Speech and Audio Processing*, vol. 4, pp. 56-60, 1996.

[8] D. A. Reynolds, "Speaker Identification and Verification using Guassian Mixture Speaker Models," *Speech Communications*, vol. 17, pp. 91-108, 1995.

[9] J. Baca, F. Zheng, H. Gao, J. Picone, "Dialog Systems for Automotive Environments," *European Conference on Speech Communication and Technology*, Geneva, Switzerland, September 2003.

[10] L. R. Rabiner and B. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

[11] J. Picone, "Continuous Speech Recognition Using Hidden Markov Models," *IEEE Acoustics, Speech and Signal Processing Magazine*, vol. 7, no. 3, pp. 26-41, July 1990.

[12] "Internet-Accessible Speech Recognition Technology," *http://www.isip.msstate.edu/projects/speech/*, Institute of Signal and Information Processing, Mississippi State University, Mississippi, USA.

[13] "Java 2 Platform SE 5.0 API Specifications," *http://java.sun.com/j2se/1.5.0/docs/api/*, Sun Microsystems Inc., Santa Clara, California, USA.

[14] N. K. Simpkins, "An Open Architecture for Language Engineering: The Advanced Language Engineering Platform (ALEP)," *Proceedings of Linguistic Engineering Convention*, Paris, July 1994.

[15] H. Cunningham, *Software Architecture for Language Engineering*, Ph.D. Dissertation, University of Sheffield, Sheffield, UK, 2000.

[16] R. Grishman, "Tipster Architecture Design Document Version 2.2," Technical report, The Defense Advanced Research Projects Agency, 1996.

[17] J. Aberdeen, et. al., "Implementing Practical Dialogue Systems with the DARPA Communicator Architecture," *Proceedings of Conference on Knowledge and Reasoning in Practical Dialogue Systems*, Stockholm, Sweden, 1999.

[18] F. Olsson, "A Requirement Analysis for an Open Set of Human Language Technology Tasks," *Proceedings of International Conference on Language Resources and Evaluation*, Las Palmas, Spain, June 2006.

[19] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, V. Zue, "Galaxy-II: A Reference Architecture for Conversational System Development," *Proceedings of International Conference on Spoken Language Processing*, Sydney, Australia, 1998.

[20] "Java™ Desktop Audio Server," *http://communicator.sourceforge.net/sites/MITRE/distributions/OSTK20020125/audio_io/jdas/doc/jdas.html*, MITRE Corporation, Bedford, Maryland, USA.

[21] V. Zue, et. al., "JUPITER: A Telephone-based Conversational Interface for Weather Information," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 1, pp. 100-112, January 2000.

[22] A. Rudnicky, et. al., "Creating Natural Dialogs in the Carnegie Mellon Communicator System," *Proceedings of the European Conference on Speech Communication and Technology*, Budapest, Hungary, September 1999.

[23] "SpeechObjects: An Architectural Overview," *http://www.nuance.com/speech/whitepapers/*, Nuance Inc., Burlington, Maryland, USA.

[24] S. Weinschenk, D. T. Barker, "Designing Effective Speech Interfaces," John Wiley and Sons Publishers, Indianapolis, Indiana, USA, 2000.

[25] L. Rothkrantz, D. Datcu, M. Beelen, "Personal Intelligent Travel Assistant: A distributed approach," *Proceedings of International Conference on Artificial Intelligence*, Las Vegas, Nevada, USA, June 2005.

[26] M. Baum, G. Erbach, M. Kommenda, G. Niklfeld, E. Waldmüller, "Speech and Multimodal Dialogue Systems for Telephony Applications based on a Speech Database of Austrian German," *Journal of Austrian Society for Artificial Intelligence*, vol. 20, no. 1, pp. 29-34, 2001.

[27] B. Pellom, W. Ward, S. Pradhan, "The CU Communicator: An Architecture for Dialogue Systems," *Proceedings of International Conference on Spoken Language Processing*, Beijing, China, November 2000.

[28] "Galaxy Communicator Manual," *http://communicator.sourceforge.net/sites/MITRE/distributions/GalaxyCommunicator/docs/manual/index.html*, MITRE Corporation, Bedford, Maryland, USA.

[29] S. Bayer, C. Doran, B. George, "Exploring Speech-enabled Dialogue with the Galaxy Communicator Infrastructure," *Proceedings of First International Conference on Human Language Technology Research*, San Diego, California, March 2001.

[30] "Programmer's Guide to Java Sound", *http://java.sun.com/j2se/1.5.0/docs/guide/sound/programmer_guide/contents.html*, Sun Microsystems Inc., Santa Clara, California, USA.

[31] A. Ganapathiraju, N. Deshmukh, J. Hamaker, V. Mantha, Y. Wu, X. Zhang, J. Zhao, J. Picone, "ISIP Public Domain LVCSR System," *Proceedings of the Speech Transcription Workshop*, Linthicum Heights, Maryland, USA, June 1999.

[32] N. Deshmukh, A. Ganapathirahu, J. Picone, "Hierarchical Search for Large-vocabulary Conversational Speech Recognition," *IEEE Signal Processing Magazine*, vol. 16, no. 5, pp. 84-107, 1999.

[33] S. B. Davis, P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," *IEEE*

Transactions on Acoustics, Speech and Signal Processing, vol. 28, pp. 357-366, August 1980.

[34] X. Huang, A. Acero and H. Hon, *Spoken Language Processing – A guide to Theory, Algorithm and System Development*, Prentice Hall, Upper Saddle River, New Jersey, USA, 2001.

[35] "The Phoenix Parser Manual", *http://cslr.colorado.edu/~whw/phoenix /Phoenix_Manual.pdf*, Center for Spoken Language Research, University of Colorado, Boulder, USA.

[36] "Travelocity," http://www.travelocity.com/, Travelocity Inc., SouthLake, Texas, USA.

[37] "Expedia," *http://www.expedia.com/*, Expedia Inc, Bellevue, Washington, USA.

[38] "Mapquest," *http://www.mapquest.com/*, Mapquest Inc., Denver, Colorado, USA.

[39] J. H. L. Hansen, J. Plucienkowski, S. Gallant, B. L. Pellom, W. Ward, "CU-Move: Robust Speech Processing for In-Vehicle Speech Systems," *Proceedings of International Conference on Spoken Language Processing*, vol. 1, pp. 524-527, Beijing, China, October 2000.

[40] G. Alippi, A. Giussani, C. Micheletti, F. Roncoroni, G. Stefini, G. Vassena, "Global Positioning and Geographical Information Systems," *IEEE Instrumentation & Measurement Magazine*, vol. 7, pp. 36-43, December 2004.

[41] J.P. Campbell, "Speaker Recognition: A Tutorial," *Proceedings of IEEE*, pp. 1437-1462, September 1997.

[42] "ISIP Foundation Classes," *http://www.cavs.msstate.edu/hse/ies/projects/speech/ software/documentation/class/index.html*, Institute of Signal and Information Processing, Mississippi State University, Mississippi, USA.

[43] "Sof Format," *http://www.cavs.msstate.edu/hse/ies/projects/speech/software/tutor ials/production/fundamentals/current/section_02/s02_01_p06.html*, Institute of Signal and Information Processing, Mississippi State University, Mississippi, USA.

[44] K. Bush, A. Ganapathiraju, P. Korman, J. Trimble , L. Webster, "A Comparison of Energy-based Endpoint Detectors for Speech Signal Processing," *http://www.cavs.msstate.edu/hse/ies/publications/courses/ece_4773/projects/1995 /conference/paper_speech.pdf*.

[45] S. Mitra, *Digital Signal Processing*, McGraw-Hill, New York, New York, USA, 1998.

[46] J. K. Proakis, D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1998.

[47] T. Stanley, J. Baca, M. Elliott and J. Picone, "Enhancements to the DARPA Communicator Architecture," *Proceedings of World Congress in Computer Science, Computer Engineering and Applied Computing*, Las Vegas, Nevada, USA, June 2006.

[48] A. Leon-Garcia, I. Widjaja, *Communication Networks: Fundamental Concepts and Key Architecture*, McGraw-Hill, New York, New York, USA, 2000.

[49] M. Liu, T. Stanley, J. Baca, J. Picone, "Robust Architecture for Human Language Technology," *Proceedings of IEEE SoutheastCon*, Memphis, Tennessee, USA, March 2006.

[50] P. Linz, *An Introduction to Formal Languages and Automata*, Jones & Bartlett Publishers, Sudbury, Maryland, USA, 2001.

[51] A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, New Jersey 1996.

[52] S. Whittaker, M. Walker, J. Moore, "Fish or Fowl: A Wizard of Oz Evaluation of Dialogue Strategies in the Restaurant Domain," *Proceedings of Language Resources and Evaluation Conference*, Gran Canaria, Spain, 2002.

[53] H. Cheng, et. al., "A Wizard of Oz Framework for Collecting Spoken Human-computer Dialogs," *Proceeding of International Conference on Spoken Language Processing*, Jeju Island, Korea, 2004.

APPENDIX A

LIST OF TASKS

Table 8 List of tasks

Serial No.	User Tasks	System Specifications
1	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	The Data Recorder server tries to write to a location that does not exist (programmer error).
2	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	Server is tries to read a parameter file that does not exist (programmer error).
3	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	A buffer overflow during the data transfer in the recording stage (programmer error).
4	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	The server tries to access a null Communicator frame and extract a value that does not exist (programmer error).
5	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	During data transfer, the program creates an audio communicator frame but does not wrap the audio data inside the frame (programmer error).
6	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	The server gets an inappropriate frame (programmer error).
7	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	Communicator frame that does not have any hub rule pertaining to it (programmer error).
8	Choose one of these tasks: 1) Use the Speech Analysis application to record audio data. 2) Use Dialog system to get a response to your query.	An error in setting a hub rule (programmer error).

Table 8 (continued)

Serial No.	Tasks	System Specifications
9	Speech Analysis Application: Recording task: time duration: 1 second	Normal operating conditions
10	Speech Analysis Application: Recording task: time duration: 5 second	Normal operating conditions
11	Speech Analysis Application: Recording task: 15 seconds time duration: 15 second	Normal operating conditions
12	Speech Analysis Application: Record and playback alternatively in the following sequence. record -> playback	Normal operating conditions
13	Speech Analysis Application: Record and playback alternatively in the following sequence. record -> playback-> record -> playback	Normal operating conditions
14	Speech Analysis Application: Record and playback alternatively in the following sequence. record -> record -> playback-> playback	Normal operating conditions
15	Dialog system Application: Try recording for different time durations (refer task 9, 10, 11).	Normal operating conditions
16	Dialog system Application: Try recording and playback alternatively (refer task 12, 13, 15).	Normal operating conditions
17	Dialog system Application: Try recording an utterance that is not part of the model (an utterance that does not relate to address-queries) and test how the recognition module handles it.	Normal operating conditions
18	Dialog system Application: Use the text mode, and try parsing a string that does not belong to an address query.	Normal operating conditions
19	Dialog system Application: Use “Text input” mode to query an address which uses the SQL database for retrieving a response.	Normal operating conditions
20	Dialog system Application: Use “Text input” mode to query an address which uses Mapquest for retrieving a response.	Normal operating conditions
21	Dialog system Application: Use “Text input” mode to query a direction which uses SQL database for retrieving a response.	Normal operating conditions
22	Dialog system Application: Use “Text input” mode to query a direction which uses Mapquest for retrieving a response.	Normal operating conditions

Table 8 (continued)

Serial No.	Tasks	System Specifications
23	Dialog system Application: Use “Text input” mode to query a direction which uses context information.	Normal operating conditions
24	Dialog system Application: Use “Text input” mode to query the distance which uses SQL database for retrieving a response.	Normal operating conditions
25	Dialog system Application: Use “Text input” mode to query a distance which uses Mapquest for retrieving a response.	Normal operating conditions
26	Dialog system Application: Use “Text input” mode to query a direction which uses context information.	Normal operating conditions
27	Dialog system Application: Use “Text input” mode to query a list of places which uses Mapquest for retrieving a response.	Normal operating conditions
28	Dialog system Application: Use “Text input” mode to query for building information which uses SQL database for retrieving a response.	Normal operating conditions
29	Dialog system Application: Use “Text input” mode to query an address which uses the SQL database for retrieving a response.	Normal operating conditions
30	Dialog system Application: Use “Text input” mode to query an address which uses Mapquest for retrieving a response.	Normal operating conditions
31	Dialog system Application: Use “Text input” mode to query a direction which uses SQL database for retrieving a response.	Normal operating conditions
32	Dialog system Application: Use “Text input” mode to query a direction which uses Mapquest for retrieving a response.	Normal operating conditions
33	Dialog system Application: Use “Text input” mode to query a direction which uses context information.	Normal operating conditions
34	Dialog system Application: Use “Text input” mode to query the distance which uses SQL database for retrieving a response.	Normal operating conditions
35	Dialog system Application: Use “Text input” mode to query a distance which uses Mapquest for retrieving a response.	Normal operating conditions
36	Dialog system Application: Use “Text input” mode to query a direction which uses context information.	Normal operating conditions

Table 8 (continued)

Serial No.	Tasks	System Specifications
37	Dialog system Application: Use “Text input” mode to query a list of places which uses Mapquest for retrieving a response.	Normal operating conditions
38	Dialog system Application: Use “Text input” mode to query for building information which uses SQL database for retrieving a response.	Normal operating conditions

APPENDIX B

LIST OF SCENARIOS

List of Scenarios

Speech Analysis Application Tasks:

Task 1:

Record your voice for varying time durations.

1) 1 second
2) 5 seconds
3) 15 seconds

Task 2:

Try recording your voice and play it back. Repeat this in different sequences.

1) record -> playback
2) record -> playback -> record -> playback
3) record -> record -> playback -> playback

Dialog Systems Application Tasks:

Task 1:

Record your voice for varying time durations (similar to Task 1 in the Speech Analysis application).

1) 1 second
2) 5 seconds

3) 15 seconds

Task 2:

Try recording your voice and play it back. Repeat this in different sequences (similar to Task 2 in the Speech Analysis application).

1) record -> playback
2) record -> playback -> record -> playback
3) record -> record -> playback -> playback

Task 3:

Imagine you are in a big city to attend a conference. Once the conference proceedings are over for the day, you want to visit some sites of interest. You don't have a map with you and have no idea about the layout of the city. Use the system to plan your trip.

Task 4:

1) Imagine you are working in a big city for quite a few years. You plan to make a visit to Starkville. So you start on a road trip from your city. You are almost near Starkville when you find that you are really low on gas. Use the system to make a decision on whether you can make it without filling gas.
2) You decide on filling gas. Use the system to locate a gas station.
3) You reach your hotel. You need to visit your friend's place. Use the system to get to his place.
4) You and your friend want to go to your favorite restaurant. Your friend is unsure whether the restaurant still exists. Use the system to verify this.

5) You are happy to find that the restaurant still exists. Use the system to reach the restaurant.
6) Once you had lunch, you want to return to your hotel. Use the system to get back to your hotel.
7) You are planning to eat your favorite cuisine for dinner. Use the system to help you in choosing a restaurant.
8) Tomorrow, you plan to visit your department. In a casual chat with your friend you learn that your department has been moved to a different location. Use the system to get the exact location.

All the tasks listed above were spoken by you through the voice interface. You need to use a text interface for the tasks mentioned below. Click the “show text input” option from the menu. You will get a text input box at the bottom of the interface. Use this input box to enter your text queries for the tasks mentioned below.

Task 5:

You want to watch your favorite TV show and cannot find it on any of the channels. Use the system to get the channel/timing information.

Task 6:

Repeat all the subdivisions in task 4. Remember to type in your queries this time instead of speaking the queries.

APPENDIX C

INSTRUCTIONS AND WARM UP EXCERCISES

Instructions

This experiment has a series of tasks that test the improvement on the robustness of our HLT system. The experiment has a collection of scenarios under which you will be asked to use the system to accomplish a given task. In order to measure the robustness improvement, you will have to repeat the tasks in the experiments twice 1) On the enhanced system and 2) On the original system. For both the experiments, you will be given maximum time duration of 30 minutes. During the experiment, you will be using the Speech Analysis and Dialog system application. The Speech Analysis application is a basic recording application. The Dialog system is an address querying system which will assist you in navigation. Please inform us immediately, if you feel the application is not responding to your queries.

Warm up exercises

1) Open the Speech Analysis application, which is the first item, listed on the Demo Selector. Try recording your utterance and playing it back. Repeat this if necessary.

2) Open the Dialog system application. You will be prompted for username and password. Please seek assistance in filling these fields. Once you have successfully logged in, you can try some of the queries listed below.

1) Where is Walmart?
2) Where is Simrall?
3) How can I go from Simrall to Butler?
4) How far is Walmart from ERC?

5) You can also try some queries on you own.

Once you are comfortable with the system, you can start the experiment. Please remember to ask for assistance if needed.