Incremental Speech Understanding in a Multimodal Web-Based Spoken Dialogue System

by

Gary M. Matthias

S.B., Massachusetts Institute of Technology (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author .....
Department of Electrical Engineering and Computer Science
May 28, 2009

Certified by .....
James R. Glass
Principal Research Scientist
Thesis Supervisor

Certified by .....
Stephanie Seneff
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Incremental Speech Understanding in a Multimodal Web-Based Spoken Dialogue System

by

Gary M. Matthias

Submitted to the Department of Electrical Engineering and Computer Science
on May 28, 2009, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In most spoken dialogue systems, the human speaker interacting with the system must wait until after finishing speaking to find out whether his or her speech has been accurately understood. The verbal and nonverbal indicators of understanding typical in human-to-human interaction are generally nonexistent in automated systems, resulting in an interaction that feels unnatural to the human user. However, as automatic speech recognition gets incorporated into web-based and portable interfaces, there are now graphical means in addition to the verbal means by which a spoken dialogue system can communicate to the user.

In this thesis, we present a multimodal web-based spoken dialogue system that incorporates incremental understanding of human speech. Through incremental understanding, the system can display to the user its current understanding of specific concepts in real-time while the user is still in the process of uttering a sentence. In addition, the user can interact with the system through nonverbal input modalities such as typing and mouse clicking. We evaluate the results of a comparative user study in which one group uses a configuration that receives incremental concept understanding, while another group uses a configuration that lacks this feature. We found that the group receiving incremental updates had a greater task completion rate and overall user satisfaction.

Thesis Supervisor: James R. Glass
Title: Principal Research Scientist

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Scientist

Acknowledgments

Before I begin my exhaustive list of acknowledgments, first I would like to say how deeply appreciative I am of the past research efforts that made this thesis possible. Conducting research with the Spoken Language Systems (SLS) Group has empowered me with a wide variety of technologies. The pre-assembled pieces for automatic speech recognition, natural language understanding and generation, speech synthesis, and dialogue management collectively form the spoken dialogue system, which I humbly inherited to produce the conversational web application presented in this thesis. Furthermore, a framework for developing graphical web applicatons greatly facilitated my endeavor of producing the multimodal conversational interface presented here.

My advisors Jim Glass and Stephanie Seneff were instrumental in the production of this thesis. I would like to thank Jim Glass for his patience and guidance, especially during times when my path felt uncertain. Also, I would like to Stephanie Seneff for the time she spent updating the previously telephone-based spoken dialogue system to accommodate an online audience, and additionally for her seemingly infinite knowledge about the GALAXY architecture and the MERCURY flight reservation system.

I cannot thank Alex Gruenstein and Ian McGraw enough for their development of the WAMI toolkit and their troubleshooting skills regarding WAMI-related issues. The *FlightBrowser* application is one of a growing list of applications that are modeled using the WAMI framework. Ibrahim Badr was the first person to introduce me to many of the SLS applications, and helped me to begin a brand new web application from scratch using the WAMI toolkit, which eventually became the *FlightBrowser* application presented in this thesis. In addition, Brandon Yoshimoto's utility for user group management made the task of conducting a comparative user study less painful.

Whenever I encountered a strange, obscure problem pertaining to the our UNIX-based operating system, I was usually convinced that it would be impossible to figure

out without spending hours pulling my hair directly out of my scalp. However, Scott Cyphers, who also served as my academic advisor throughout my MIT educational journey, always seemed to intuitively know exactly how to alleviate the problem, no matter how bizarre or arcane the issue turned out to be.

I would like to thank Marcia Davidson for handling the distribution of the Amazon gift certificates to test subjects before they degenerated into an angry email mob. Also, I would like to acknowledge Stephen Pueblo for his words of optimism and encouragement, even when none were necessary.

And, of course, I would like to acknowledge all the unnamed persons in my life that contributed to the production of this thesis, however indirect their contribution may have been. I would especially like to thank my parents for loving me unconditionally despite all of the phone calls that I neglected to make while preoccupied with my academic work.

This research was made possible by the support of Foxconn.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis explores the use of incremental speech understanding in a spoken dialogue system to display real-time understanding to a human user while he or she is speaking. Furthermore, once the speaker has completed an utterance, the system resolves its current understanding with contextual information stored in its discourse as context. We introduce *FlightBrowser*, a multimodal, spoken dialogue web application that provides information about airline flight schedules. In order to measure the effects of displaying the system's comprehension to the user, we perform a comparative study using two different configurations of the *FlightBrowser* system—an enhanced configuration that displays the information provided through incremental speech understanding, and a baseline configuration that does not display this information. We report the results of this study based on the completion rate of assigned tasks, system usage, and user satisfaction based on a post-study evaluation survey.

1.1 Incremental Understanding in Humans

A *spoken dialogue system* is a system that uses automatic speech recognition while retaining information about the entire verbal exchange internally as context. In many of these spoken dialogue systems, the most common scenario is that a human user utters a sentence to the system; then, once the user has finished speaking, the speech is processed and the system responds accordingly to the user input. During the few sec-

onds of processing and the subsequent system response, the user may wonder whether the system has accurately recognized the utterance. This back-and-forth style of interaction results in a dialogue that often seems unnatural to humans, especially those whose experience with human-machine speech interaction is limited. In a dialogue between two humans, it is typical for the listener to provide verbal and nonverbal indicators of understanding to the speaker. This can be achieved by using vocalizations such as "uh-huh," more meaningful words such as "okay" and "wow," or nonverbally through gestures, facial expressions, and head-nodding [44]. Providing these verbal or nonverbal cues to the speaker is commonly referred to as *back-channeling*, a term coined by Yngve in [42].

In a unimodal spoken dialogue system where speech is the only modality for sending and receiving information, the system is unable to convey its current understanding to the human user until after a completed spoken utterance. Thus, the typical turn-taking model of a unimodal system is vastly different from that of a human-to-human interaction, since back-channeling does not occur. A human expects to receive some feedback from the listener at certain points during each sentence—often when new, significant pieces of information have been added to the current context. This type of verbal comprehension is referred to as *incremental understanding*. Chater et al. describe incremental understanding as the process by which humans analyze speech (or written text) in a piecewise manner as it is received [6].

Neurolinguistic research provides evidence that the human mind interprets language incrementally as proposed by Chater et al. In [40], Tanenhaus et al. describe an eye-tracking experiment in which the subject is given an image to view while the researcher dictates a sentence about an object in the image, describing it by its physical characteristics and relative position within the image. This study indicates that eye movement occurs incrementally at specific places in the sentence where the subjects' overall understanding increases. In addition, this study shows that sentences containing ambiguous phrases result in additional incremental eye movement as extra steps of incremental understanding are needed to resolve contextual ambiguity.

To illustrate the piecewise language analysis that is characteristic of incremental

I would like a flight — from Boston — to San Francisco — departing on Friday — in the morning — at eight o'clock — on American Airlines.

Figure 1-1: Illustration of incremental understanding. The dashes (—) represent locations in the sentence where the discourse is incrementally updated.

understanding, take the following sentence in Figure 1-1. In this example, we are presented with a typical flight reservation query from a *FlightBrowser* dialogue: "I would like a flight from Boston to San Francisco departing on Friday in the morning at eight o'clock on American Airlines." This utterance can be divided into pieces, delimited here with a dash (—) such that each piece represents an incremental update to the context. In an automated system, these dashes would be analogous to the locations within the sentence where the real-time understanding would be incrementally updated.

1.2 Web-Based Interfaces and Multimodality

The Internet has gained popularity over the past decade as a medium for speech-enabled interfaces due to the advances in speech technology and the expanding availability of high-speed Internet access. An application that performs speech recognition over the Internet has several advantages compared to its telephone-based counterpart. One advantage is that the Internet provides a much larger potential audience for research subjects. Another potential advantage is that tasks such as speech transcription that require human effort can be "crowdsourced" to Internet users. For example, Paek et al. of Microsoft describe the use of the crowdsourcing web site, Amazon Mechanical Turk[1], as a way of aggregating human-transcribed data for a directory assistance corpus [31].

In addition to the benefits for data collection and analysis, the multimodal capability of the Internet adds a new dimension to the user experience. One example of multimodality is the use of graphics to present additional information to the user, or

[1]Amazon Mechanical Turk, http://www.mturk.com

information that would otherwise take an excessive amount of time to depict verbally through speech synthesis. Multimodality can be further exemplified by web-based applications of speech recognition technology in cell phones, cars, and global positioning systems (GPS). The advantages of a graphical, multimodal interface and the potential for future portable applications were the motivation behind implementing a graphical, web-based version of our pre-existing telephone-based flight schedule system.

FlightBrowser, our graphical web application centered around a previously-developed flight reservation dialogue system, implements incremental speech understanding similar to the phenomenon observed in human language understanding. In the context of automatic speech recognition, *incremental speech understanding*, or *incremental understanding*, involves taking partial recognition hypotheses of the incomplete sentence while a user is speaking a sentence, then using natural language processing techniques to parse the incomplete sentence into a set of keys and values. These keys represent important domain-specific information from the utterance, which will be termed as *concepts* in this thesis. In the flight reservation domain, examples of concepts would be the source and destination airports, airline, departure time, etc. The key-value representation produced through incremental speech understanding is then converted into graphical output that the user observes while uttering a sentence.

Furthermore, after the sentence has been processed by the dialogue manager, the system updates the values of those keys with context-resolved values by a process called *context resolution*, illustrated in Figure 1-2. In this example, we see the incremental updates made to the domain-specific concepts. Once the utterance is complete, the concepts are updated with the context-resolved values. For example, the airports (Portland, Miami) and airline (United) are converted to their respective IATA[2] airport codes (PDX, MIA) and airline code (UA). Also, the date Thursday is converted to a full date (Thursday May 21) after being processed by the dialogue management component, and the ambiguous time 8:00 a.m., which could be either a departure or arrival time, is chosen to be a departure time based on contextual system knowledge.

[2]International Air Transport Association

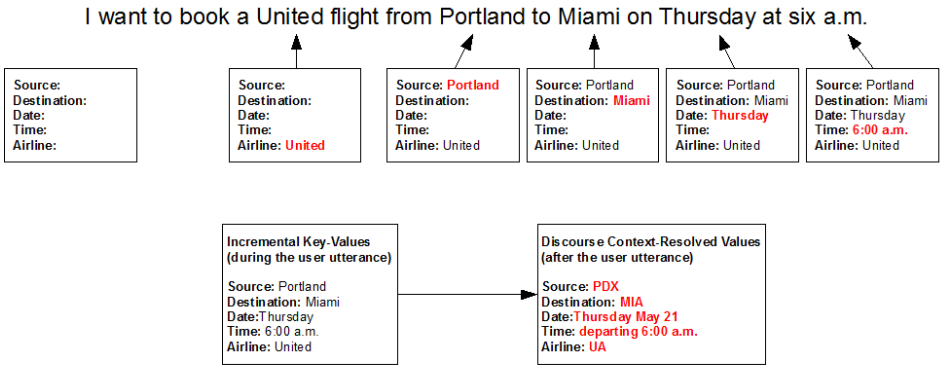


Figure 1-2: A key-value representation of real-time incremental understanding and post-utterance context resolution. The bold values represent updates to the key-value concept representation.

This web-based spoken dialogue system is *multimodal* in the way that it presents information to the user, and also in the way that it receives information from the user. That is to say, this system accepts speech input and produces speech output while also providing alternative *modalities* for sending and receiving data. The *FlightBrowser* interface can either present the data graphically through text and images, or audibly through speech synthesis. When interacting with the system, a user may communicate with the system using the modalities of speech, typing, and mouse clicking.

In addition to conveying real-time understanding, *FlightBrowser* includes several other multimodal features. One example of a multimodal feature is the graphical list of flights that meet a set of constraints provided by the user. The user may sort this list of flights by criteria such as price, departure time, or duration by clicking on their corresponding links in the document. Also, the user may click on a specific flight and refer to that flight through speech. Implementing this feature allows us to increase the number of flights of which the user may speak without lengthening the synthesized speech output, and without forcing the user to commit information to memory. Thus, incorporating automatic speech recognition into web-based and portable systems gives the user more versatility than a speech-only interface and enables the creation of more complex conversational interfaces.

1.3 Problem Statement

In this thesis, we describe the design of *FlightBrowser*, a graphical web-based interface that extends the functionality of our pre-existing telephone-based spoken dialogue system. We describe the process of modifying the infrastructure of our original system in the following ways:

- designing a graphical interface to supplement the information provided in the original speech-only system,
- enabling support for multimodal input from our graphical interface, and
- implementing incremental speech understanding as a means of providing visual feedback to the user.

Furthermore, we will study and quantify the effects of a specific multimodal feature in the *FlightBrowser* web application—namely, the text fields that provide real-time incremental feedback to the user and their corresponding recording buttons for concept-specific corrections. We describe a comparative user study consisting of two subject groups, each using a different configuration of the *FlightBrowser* system. One group interacts with an enhanced version of the system that includes incremental understanding and context resolution via text fields, while the second group interacts with a baseline configuration that is deficient of these recording buttons and text fields for incremental feedback. This thesis will explore differences between the groups in such categories as multimodal feature usage, task completion, and user satisfaction.

1.4 Overview

The remainder of the thesis is structured as follows: Chapter 2 will mention related research in telephone-based dialogue systems, multimodal and web-based dialogue systems, and interfaces that implement incremental understanding; Chapter 3 will describe the underlying web server framework and the various components that combine to create the conversational interface; Chapter 4 will elaborate specifically on

the *FlightBrowser* web application, describing its back-end design, graphical layout, and noteworthy features, while also describing our implementation of incremental understanding; Chapter 5 will discuss the setup of a user study comparing the two different configurations of the FlightBrowser system, with an analysis of the results; and finally, Chapter 6 will include a discussion of the results of this research, with concluding remarks and suggestions for future directions of research.

Chapter 2

Related Work

This chapter will describe work related to the research presented in this thesis. We begin by discussing previous work with telephone-based spoken dialogue systems that utilize a single modality for input and output. Next, we will discuss examples of multimodality in a wide range of spoken dialogue systems. Then, we will cover several implementations of incremental understanding in spoken dialogue systems.

2.1 Telephone-Based Systems

There have been previous studies with telephone-based spoken dialogue systems in which speech is the sole modality for both input and output. Several such unimodal telephone-based systems have been developed by the Spoken Language Systems Group (SLS) at the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL). One example of a telephone-based system is JUPITER, a spoken dialogue system that retrieves weather forecasts for specific cities [14, 47]. Another example is the MERCURY system [36, 38, 39] for scheduling airline flight reservations that serves as part of the *FlightBrowser* infrastructure. Both of these systems involve a goal-oriented, domain-specific interaction between a human and a machine. The human has a specific goal to fulfill and the machine must understand the user's speech to assist him or her in accomplishing this task. Therefore, the system must be equipped to handle a multitude of possible responses in the user's natural language,

while also possessing a mastery of the domain-specific vocabulary.

Another domain-specific telephone-based conversational system is the one described by Gorin et al. of AT&T Labs [15] for troubleshooting and providing assistance for telephone-related issues. In this system, the user is asked the open-ended question "How may I help you?" After the user phrases a question or request concerning a telephone-related problem, the system either provides the requested information to the user, or, for more complicated queries, redirects the user to the appropriate human agent.

Displayless conversational interfaces such as the aforementioned telephone-based systems have several drawbacks. For instance, Zue et al. explain that since speech synthesis is the only way to portray information to the user in a unimodal system, this information must be very descriptive; however, it must be presented in manageable amounts, as several consecutive sentences of synthesized audio output may overwhelm the user cognitively [47]. As a result, a graphical display as part of a conversational interface allows for more efficient communication with the user, while also improving the user experience by shortening the amount of time spent on synthesized speech.

As Internet bandwidth continues to increase, most notably for portable devices, the Internet is rapidly becoming a more desirable medium for spoken dialogue systems. One primary advantage of web-based multimodal dialogue systems over their telephone-based counterparts is the graphical component of the interface. Often it takes a user much less time to read information or view images on a screen than it does to listen to identical information presented by means of synthesized audio output. A significant advantage is that the information displayed remains on the screen, thus the user does not need to commit it to memory. Furthermore, tasks are often made easier by combining speech with another modality, i.e., using the second modality to provide additional context to the spoken utterance. Hence, the multimodal capability of a web-based system, when effectively exploited, saves time when carrying out certain tasks in a spoken dialogue system. The following section elaborates on some of the previous work with multimodal speech interfaces.

2.2 Multimodal Systems

Many attempts have been made to combine multimodality with automatic speech recognition technology. Microsoft's MiPad (Multimodal Interface Pad) [7] is a personal digital assistant (PDA) that has a multimodal speech interface. MiPad can receive input through speech or pen modalities. The graphical display shows the user how the system has interpreted the user's intentions, so that corrections can be carried out through the appropriate modality if necessary.

Another example of a multimodal system is MATCH (Multimodal Access to City Help), developed by Johnston, Bangalore, et al. of AT&T Labs [4, 24, 25]. MATCH, a system that provides information about subways and restaurants in New York City, has been implemented on a tablet computer and a kiosk. The MATCH system can receive speech and pen input (or touch-screen input for the kiosk version) simultaneously and can portray information graphically or through synthesized speech output. City Browser [20, 21] is a web application developed by Gruenstein et al. of SLS that is similar to MATCH in its functionality. The City Browser interface provides information for various metropolitan areas rather than a single city and can receive speech, pen, or mouse input. It has been incorporated as an application for a tablet computer and as an automotive human-machine interface [19].

Another application of multimodality in a spoken dialogue interface is a home entertainment system, also developed in SLS by various researchers [17]. This project takes spoken input from a smart cell phone[1] to retrieve information about television program schedules, displaying it on a remote graphical output device, e.g., a television or computer monitor. As illustrated by the wide variety in both the functional domains of the multimodal systems listed here and the devices on which they are implemented, multimodality serves as a way of enriching the conversational experience and expanding the reach of automatic speech recognition technology.

[1] We use the term "smart cell phone" to refer to a cell phone with enhanced capability.

2.3 Incremental Understanding

There have been previous implementations of incremental understanding in multimodal systems. Higashinaka et al. [23], Miyazaki et al. [29], and Nakano et al. [30] of NTT Laboratories in Japan describe a meeting scheduler that incrementally accepts sentence fragments in real-time, updating the information stored in the discourse after new information has been provided. Fink et al. of the University of Bielefeld in Germany describe a method of incremental understanding applied to a virtual reality environment [9]. In this system, incremental speech understanding is combined with sensor-tracked hand gestures, creating an interface in which a user assembles virtual parts such as tires, metal pipes, and engines to design a personal transport vehicle in the virtual environment.

Gómez Gallo [10, 11], Aist [1, 2, 3], et al. of the University of Rochester describe Fruit Carts, an incremental spoken dialogue system in which the user's objective is to match a configuration of fruits and geometric shapes. In a similar small-domain implementation of task-driven incremental understanding, Gruenstein of SLS has developed an incremental web-based interface, Shape Game, where the goal is to match a pattern of shapes by moving and modifying a separate set of shapes [16]. Both of these systems attempt to portray the user's intentions on the screen in real-time, assuring the user of the system's understanding, and enabling the user to make any necessary corrections before the end of a spoken utterance. As a follow-on to Shape Game, several related games are being developed by other students in the SLS group for language learning applications, for example, Word War [28] and Rainbow Rummy [43].

There are several possible motivations for implementing incremental speech understanding in a conversational interface. One motivation for incremental understanding is the ability to stream continuous speech input from the user, updating the discourse in real-time so that multimodal actions are contextually coherent. Another possible motivation of incremental speech understanding is to display the user's perceived intentions in real-time so that speech recognition errors can be pinpointed and cor-

rected before the end of a spoken utterance, if necessary. The motivation behind the *FlightBrowser* system is to show the system's internal representation of the user's speech content in real-time, and also to demonstrate the process of context resolution once the user has stopped speaking.

2.4 Chapter Summary

As shown in this chapter, there have been several telephone-based spoken dialogue systems that were developed when the telephone was the most effective medium for conversational interfaces. However, as the Internet becomes more capable of supporting the required bandwidth for audio streaming and playing, especially in portable devices such as smart cell phones and PDAs, many of these unimodal telephone-based systems are being replaced with web-based systems. This has catalyzed the development of multimodal systems that can present and receive information from a variety of non-speech modalities to supplement the speech input. In the next chapter, we will describe the system components that serve as the back end of the *FlightBrowser* system. Then, Chapter 4 will provide more detail about the specific multimodal capabilities of our system, including our implementation of incremental understanding, and Chapter 5 will describe a user study that aims to quantify the effects of displaying the system's internal understanding to the user.

Chapter 3

System Components

In this chapter, we describe the overall system architecture and components that make up the back end for the *FlightBrowser* interface. Figure 3-1 is a block diagram outlining the relationships among these components. We begin this chapter by describing the spoken dialogue component of the back end. First, we will describe our transition from the GALAXY hub-based client/server architecture for the spoken dialogue system to a version that does not use a centralized hub for communication. Then, there will be a description of the various modules that compose the spoken dialogue system, including automatic speech recognition, natural language understanding and generation, discourse context tracking, dialogue management, and speech synthesis. Finally, we describe the WAMI toolkit for developing web-accessible multimodal interfaces, the basic framework for connecting the GUI (graphical user interface) server and client via a web server.

3.1 Client/Server Architecture

Our flight reservation spoken dialogue system consists of several interconnected components as shown in Figure 3-1. In the "Spoken Dialogue System" block in the figure, the natural language understanding, discourse context tracking, dialogue management, and natural language generation are surrounded by a dashed box. These components are represented internally by a set of *declarative knowledge* that is known

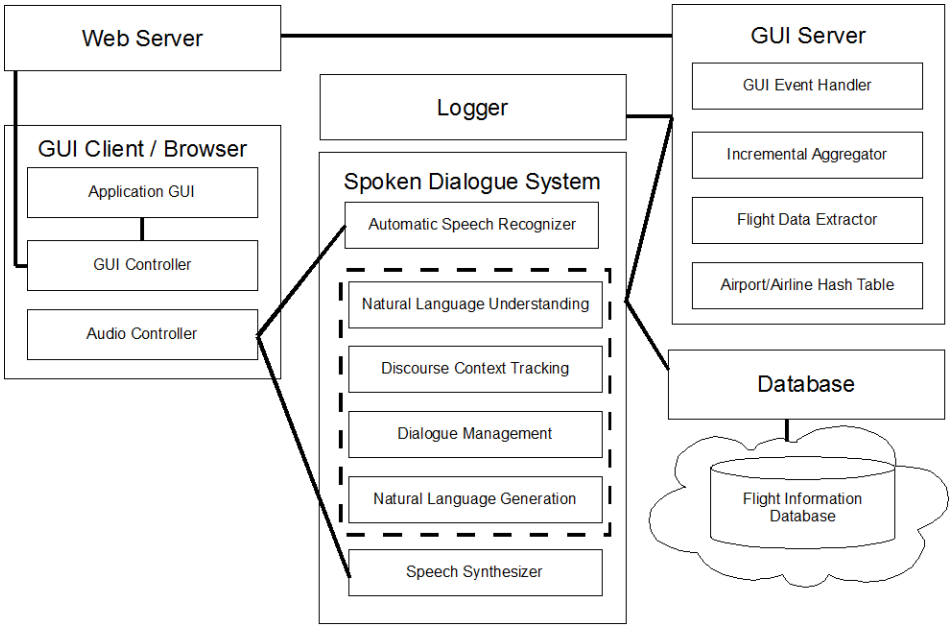


Figure 3-1: Block diagram of the *FlightBrowser* system.

at runtime and *procedural rules* that are executed based on information provided via conversation with a human user. In order to connect the various system modules, our flight reservation dialogue system had originally been using the GALAXY client/server architecture, based on a centralized programmable hub server [37]. However, during the course of this project we consolidated this system into a hubless client/server architecture for greater compatibility with mobile devices. In this section, we will describe the architecture of our spoken dialogue system, first describing the original hub-based GALAXY system, then explaining the changes implemented for our recently-developed hubless architecture.

3.1.1 GALAXY Hub-Based Architecture

The GALAXY architecture consists of a network of servers that intercommunicate via a single centralized hub server. Each task in the spoken dialogue system has a separate module in the architecture that connects to the hub. When the system requests a specific module to perform a task, the hub sends the module a message, or *frame*, specifying parameters for the requested task. Once the module has completed its task, it sends its results back to the hub, which then handles the results accord-

ingly. The separate modules that communicate with the hub are as follows: audio streaming and playing, automatic speech recognition, natural language understanding, discourse context tracking, dialogue management, natural language generation, speech synthesis, and database access. Refer to Figure 3-2 for an illustration of these modules and their interaction with the centralized hub.

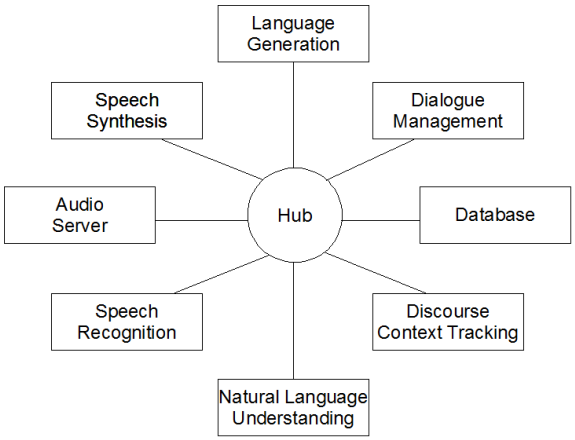


Figure 3-2: Diagram of the GALAXY architecture described in [37]. All modules communicate via a centralized hub server.

The hub executes a hub script that contains a list of the locations (i.e. host names and port numbers) of the various servers that will communicate with the hub. The hub script also imports program files that contain a set of hub rules. Each rule specifies conditions that must be satisfied in order to carry out a specific operation. These rules also consist of input and output variables for the rule (`INPUT`, `OUTPUT`), parameters that have pre-assigned values (`PARAM`), variables that are stored into or retrieved from the discourse history (`STORE`, `RETRIEVE`), and variables that are logged during input or output (`LOG_IN`, `LOG_OUT`).

Figure 3-3 is an example of a rule in the hub script. The conditions that must be satisfied for this rule to fire are that there must exist either an incremental key-value frame or string (`:kv_frame |:key_value`), however, no discourse key-value frame (`& !:discourse_kv_frame`). Then, a frame titled `handle_partial` containing an `:input_string` and a `:paraphrase_string` (decoded and generated from the user speech), and also a `:partial_frame` and `:partial_result` (renamed from `:kv_frame` and `:key_value`, respectively) are sent to the GUI server for further processing. Fur-

thermore, several input variables are stored in a log file of the current session.

```
RULE: :kv_frame | :key_value & !:discourse_kv_frame --> gui.handle_partial
IN: :input_string :paraphrase_string \
    (:partial_frame :kv_frame) (:partial_result :key_value)
LOG_IN: :paraphrase_string :partial_result :partial_frame
```

Figure 3-3: Example of a GALAXY hub rule. This rule is analogous to the hubless dialogue control rule in Figure 3-4.

3.1.2 Hubless Architecture

Although the hub-based architecture is equipped to handle multimodal input, modifying the code to the existing hub program is often a daunting task when dealing with simultaneous speech and multimodal input. This problem was mainly due to ambiguity with the history IDs, reference values that are incremented at the end of each computer generated response to a user query. These history IDs are not much of an obstacle when receiving input from one modality at a time. However, when the user utilizes two modalities concurrently, the manipulation of the history IDs causes ambiguity when attempting to store or retrieve variables from the session history.

This history ID ambiguity was one motivation behind developing a hubless system that consolidates the tasks of several GALAXY modules (natural language understanding and generation, discourse, and dialogue management) into a single server. Furthermore, combining these tasks into a single server makes our system more compatible with mobile devices for which a more simplified architecture is preferred. The hubless architecture varies from our original hub-based system because the modules are stored as libraries contained in a single Python wrapper rather than on separate TCP ports. Instead of communicating via ports, the various modules communicate with each other using XML-RPC calls.

In addition to completely remodeling the previous architecture to remove hub dependence, this transition also required refactoring the hub rules to the dialogue control rule format and combining these modified hub rules with the existing dialogue control rules. Figure 3-4 shows an example of such an adaptation. The GALAXY

hub rule shown in Figure 3-3 has been converted to a discourse control rule, removing the need for the hub program while retaining its previous functionality.

```
{c rule
   :conditions "(:kv_frame | :key_value) & !:discourse_kv_frame"
   :variables {c variables
                  :log_in ":paraphrase_string :partial_result :partial_frame"
                  :in ":input_string :paraphrase_string
                       (:partial_frame :kv_frame) (:partial_result :key_value)"
                  :program "handle_partial" }
   :operation "dispatch_token" }
```

Figure 3-4: Example of a hubless dialogue control rule. This rule is analogous to the GALAXY hub rule in Figure 3-3.

3.2 Speech Recognizer

This section will describe SUMMIT, an automatic speech recognizer developed by the Spoken Language System Group (SLS) [12, 13, 27, 45, 46].

3.2.1 Landmark-Based Segmentation

Many speech recognition systems use fixed-length time windows (frames[1]) to segment an acoustic waveform [34], evaluating short-term spectral information such as Mel-frequency cepstral coeffecients (MFCCs) over the window. Then, hidden Markov models (HMMs) are used to map out the observation space. The SUMMIT recognizer, in contrast, divides the speech into segments of variable length, representing each segment as a fixed-size feature vector, whereas a frame-based system would require an additional processing stage to convert the frame information into a feature vector. The choice of a segmentation site in SUMMIT is guided by a spectral change detection algorithm, which detects points in speech where the most constriction or opening in the vocal tract occurs.

[1]Not to be confused with the "frame" describe in section 3.1.1.

3.2.2 Modeling and Decoding

SUMMIT has a lexical model that maps words to their corresponding pronunciations. It is equipped to handle common variations in word pronunciation, which are instantiated via a set of phonological rules. The language model is based on a class n-gram model which assumes that a word, or class of words, depends statistically on the $n-1$ words or classes preceding it. Using these models, SUMMIT discovers the sequence of words with the best score based on a two-pass search. The forward pass uses a Viterbi search and the backward pass uses an A* search to decode the speech input.

3.2.3 Endpoint Detection

Automatic endpoint detection is used to detect the beginning and end of the spoken utterance. For the starting point, the SUMMIT system waits until the energy of the speech exceeds a certain energy threshold for a brief time window. Similarly, the ending point of the utterance is determined when the energy of the speech content falls below a threshold for a brief period of time. Individual frames are sent to the back end for each of these boundary markers. Endpoint detection is used in the web-based system to wait for hands-free speech input instead of forcing the user to hold down a mouse button while speaking. This allows the user to simultaneously utilize other modalities while speaking, if desired.

3.3 MERCURY Flight Reservation System

The *FlightBrowser* multimodal web interface makes use of the MERCURY flight reservation spoken dialogue system, developed by SLS [36, 38, 39]. This section will describe various aspects of the MERCURY system including its natural language understanding and generation, and its discourse and dialogue management.

3.3.1 Natural Language Understanding

The MERCURY system has a natural language (NL) understanding component that parses the decoded utterance from the speech recognition module into a semantic frame. This frame is used by the discourse module to make changes to the current context and to determine which conditions will be satisfied to carry out application-specific operations. This semantic frame parsing is performed by TINA [35], which carries out this parsing based on MERCURY-specific NL grammar rules. The lexicon of MERCURY is composed of approximately 1750 words and the context-free grammar includes around 1100 nonterminal categories.

Figure 3-5 shows the semantic parsing of the sentence, "I want a flight from Boston to Miami tomorrow morning" in a tree structure where each node represents a grammar category. The topmost nodes correspond to more general semantic and syntactic categories, whereas nodes closer to the bottom of the tree represent more specific semantic categories. Figure 3-6 shows a *linguistic frame* for the user-spoken query, dividing the query into predicates for the source, destination, time interval, and date. This frame is created from the parse tree, guided by a set of mapping rules.

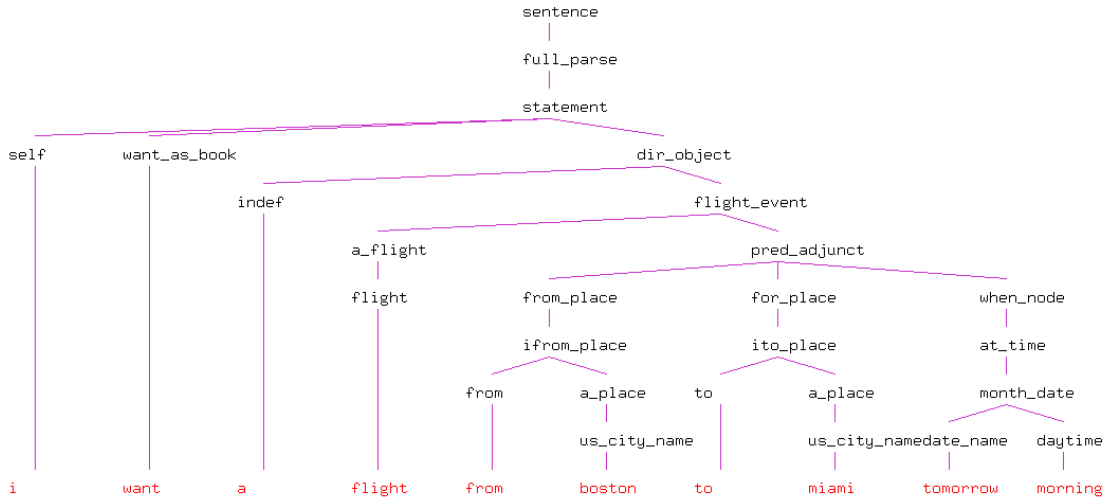


Figure 3-5: Parse tree for the utterance "I want a flight from Boston to Miami tomorrow."

```
{c maybe_book
   :domain "Mercury"
   :topic {q flight
             :quantifier "indef"
             :num_preds 4
             :pred {p source
                      :topic {q city
                                :name "boston" } }
             :pred {p destination
                      :topic {q city
                                :name "miami" } }
             :pred {p time_interval
                      :topic {q time_of_day
                                :name "morning" } }
             :pred {p month_date
                      :topic {q date
                                :name "tomorrow" } } } }
```

Figure 3-6: Linguistic frame for the utterance "I want a flight from Boston to Miami tomorrow."

3.3.2 Discourse and Dialogue Management

An essential component of any spoken dialogue system is the ability to store the current state of the dialogue, making decisions based on the stored history of the conversation. After the user's speech input has undergone automatic speech recognition and natural language understanding, the resulting linguistic frames are manipulated to acquire new information about the active conversation. Two modules in the spoken dialogue system are responsible for carrying out this task. The discourse module aims to understand the user's speech with respect to the context of the conversation, and the dialogue management module decides what actions to perform based on the context-resolved user speech.

The discourse is the source of context for the dialogue, consisting of user-presented information and world knowledge from the flight reservation domain. The discourse component carries out a context resolution algorithm that determines the meaning of the user's current utterance based on previous utterances. Included in this algorithm are stages for determining the meaning of ambiguous phrases such as pronouns (it), demonstrative noun phrases (this flight), and ordinal phrases (the first one) based on the context and system initiative. Then, inheritance rules decide which items of

context to remember or forget. Furthermore, references to relative dates like "next Thursday" are resolved into absolute dates. Finally, the updated history is stored in a frame that stores all relevant contextual variables from which the system can make decisions. When the user provides new information to the system with a subsequent utterance, the discourse begins with an initial state, then a series of rules are executed in a precise order, firing based on context-dependent conditions. The context resolution algorithm of the discourse module is described with more detail in [8].

The dialogue management module [38] has built-in strategies for requesting information based on the current context. One example of its strategies is that it asks questions regarding the flight details until there is enough information to carry out a query. The system requires all of this information before it can retrieve results from an external online database of flights. This information can be provided by the user all in one utterance or in any particular order over several utterances. If the system has not yet contacted the database and the discourse is still missing one of the required pieces of information, the system requests the missing piece(s) of information from the user. Since the dialogue control consists of an ordered set of rules, the system requests the destination, source, and date exactly in that order, although the user may choose to supply the information in any arbitrary order. Figure 3-7 has an example of the dialogue rules associated with filling in flight details. Once there is enough information to perform a database query, another set of rules are processed to control how the system will present the database results in a verbal summary.

```
...
{c rule
   :conditions ":num_found <0 & !:destination"
   :operation "need_destination" }
...
{c rule
   :conditions ":num_found <0 & !:source"
   :operation "need_source" }
...
{c rule
   :conditions ":num_found <0 & !:date"
   :operation "need_date" }
...
```

Figure 3-7: Example of a set of dialogue control rules.

3.3.3 Natural Language Generation

At the end of the context resolution and dialogue management stage, a *reply frame* is generated containing the information that will be used to generate a verbal response. In the natural language generation step, this reply frame is converted into a reply string and a synthesis string using GENESIS [5].

3.4 Speech Synthesis

In order to produce a "conversational" experience for the user, we supply natural language speech output to the user by means of a speech synthesizer. Based on the runtime configuration, this system uses one of two speech synthesizers. We incorporate the DECtalk [22] and ENVOICE [41] systems for speech synthesis, though we can potentially support a wide-range of text-to-speech (TTS) synthesizers. For DECtalk, a TTS synthesizer, we provide a synthesis string that is identical to NL string produced by GENESIS. However, for the ENVOICE synthesizer, the synthesis string is composed of a series of parameters that represent the audio files that will be concatenated for synthesis. See Figure 3-8 for an illustration of reply string and synthesis string generation.

During the course of this research, we changed the speech synthesizer from ENVOICE to DECtalk, which results in an obvious trade-off. Although DECtalk is more versatile in producing any arbitrary speech output, most people would state that DECtalk sounds uncomfortably robotic. The ENVOICE synthesizer has a more pleasant voice, however, it is only trained to pronounce a limited number of phrases. This limitation of ENVOICE hinders future expansion of the MERCURY system, therefore a more versatile synthesizer is preferred for future development.

3.5 WAMI Toolkit

The WAMI toolkit for web-accessible multimodal interfaces, developed in SLS, is a publicly available framework that allows software developers to create interfaces

```
{c need_date
   :continuant {c empty }
   :domain "Mercury"
   :topic {q iflight
             :pred {p source
                      :topic {q city
                                :name "SEA" } }
             :pred {p destination
                      :topic {q city
                                :name "NYC" } } } }
```

Generated text string (for conversation box and DECtalk speech synthesizer):
Okay, from Seattle to New York City. What date will you be traveling?

Figure 3-8: Example of natural language generation from a reply frame for the utterance "I want to book a flight from Seattle to New York City."

utilizing automatic speech recognition technology [18]. Through this framework, it is relatively simple to develop a prototype for a speech-enabled, multimodal web interface. Users may access these interfaces over the Internet with any standard web browser, which allows for systems that are usable on alternative computing devices that access the Web. The WAMI toolkit uses AJAX (Asynchronous JavaScript and XML) technology, which has been popularized recently due to the rich, dynamic web pages it is capable of producing. WAMI has been the foundation for several other SLS speech-enabled web applications, such as City Browser for geographical knowledge of metropolitan areas [19, 20, 21], the Word War game for second language acquisition in Mandarin [28], and a home entertainment system [17]. Refer back to 3-1 to view the relationship between separate components in our WAMI-based system design.

3.5.1 GUI Client/Server Communication

A WAMI-based system consists of a GUI server and client. The GUI client serves as the front end of the application, part of which composes the graphical interface that the user interacts with. The GUI client is composed of the audio controller (described in the following section), a GUI controller, and the application GUI itself. The application GUI is what visually appears to the user, built on a foundation of HTML, CSS (Cascading Style Sheets), and JavaScript. The GUI client and server communicate with each other by means of a Java EE (Java Enterprise Edition) HTTP

web server. The GUI controller module, which has a JavaScript code base, serves as the pathway to the back-end GUI server through the HTTP web server. The GUI server handles the back-end application logic, receiving messages from the spoken dialogue system in the form of frames. The GUI server and client communicate with each other through XML messages sent over the web server.

3.5.2 Audio Controller

The WAMI framework includes an audio controller as part of the GUI client. The audio controller usually consists of a Java applet that connects to the sound card or audio device of the client computer. This sound card or audio device is used for both input and output. For input, the audio device records as the user speaks into the microphone and streams this data directly to the speech recognizer in the back end. Furthermore, once the user utterance is complete, the system streams output to the audio device directly from the speech synthesizer.

3.5.3 Logging and Annotation

The WAMI system also has built-in capability on the server side for logging the current session to a database. Included among these database entries are the waveforms for the user utterances, N-best hypotheses for these utterances, and logged occurrences of application-specific events. In addition, the WAMI toolkit provides an interface by which a researcher can annotate a user session and evaluate the system's ability to provide a correct response to a spoken query.

3.5.4 User Group Management

In addition to the logging and annotation capability, a utility has been developed to manage groups of users participating a user study [43]. Using a database for storage, user information is recorded, classifying users into groups that have distinct sets of tasks or application parameters. Users log into the system using an email address to

receive their personalized or group-specified settings. We will discuss the user group management more in Chapter 5, where we discuss our *FlightBrowser* user study.

3.6 Chapter Summary

Over the past few years, advances in web technology have expanded the capabilities of online interfaces. Whereas in the past, the use of third-party applications was required to create dynamic interfaces, now AJAX techniques allow for dynamic page manipulation without having to reload the page [32]. Although a special application (Java, Adobe Flash, etc.) may still be needed for tasks such as audio streaming and playing, the dynamic page manipulation using AJAX provides an ideal situation for a multimodal spoken dialogue system. XML messaging between the HTML/JavaScript-based GUI client and the Java GUI server enables the real-time use of multimodal input and dynamic page changes necessary in the *FlightBrowser* system.

The greater ease, in addition to an increase in connection speeds, have allowed us to implement an interactive interface for the pre-existing telephone-based MERCURY flight reservation system, thus broadening the potential audience of users. This required some modification to the architecture of the original system, replacing our hub-based system with a hubless system to better support multimodal input. In the next chapter, we will describe the functionality and layout of the graphical user interface from both the server and client side, and also explain how we dynamically update the interface.

Chapter 4

Layout and Functionality

One of the accomplishments of the *FlightBrowser* project is the development of a graphical interface for the MERCURY flight reservation system. Although the Web-GALAXY [26] interface implements a graphical version of the MERCURY system, the multimodality of this system mainly serves to mimic the speech output in a graphical format. The *FlightBrowser* system extends this multimodal capability by providing additional content not presented in the speech output, and allowing the user to speak about items that appear on the screen. In addition, *FlightBrowser* gives the user the ability to provide additional context to the speech input by using a second modality such as typing and mouse clicking. Thus, the system presented in this thesis incorporates multimodal functionality comparable with the multimodal conversational interfaces listed in Chapter 2.

FlightBrowser goes a step further by portraying real-time incremental speech understanding as a way of back-channeling to the human speaker. The graphical component is utilized as a way of providing graphical feedback that portrays the user's intentions. Incremental understanding has been applied to several domains, such as task-driven games ([3], [16], [28], and [43]), a meeting scheduler [23, 29, 30], and a virtual reality environement [9]. However, no attempts have been made to accomplish incremental understanding in the flight reservation domain, a useful application of automatic speech recognition technology and research.

In this chapter, we will describe the framework of *FlightBrowser* and how it in-

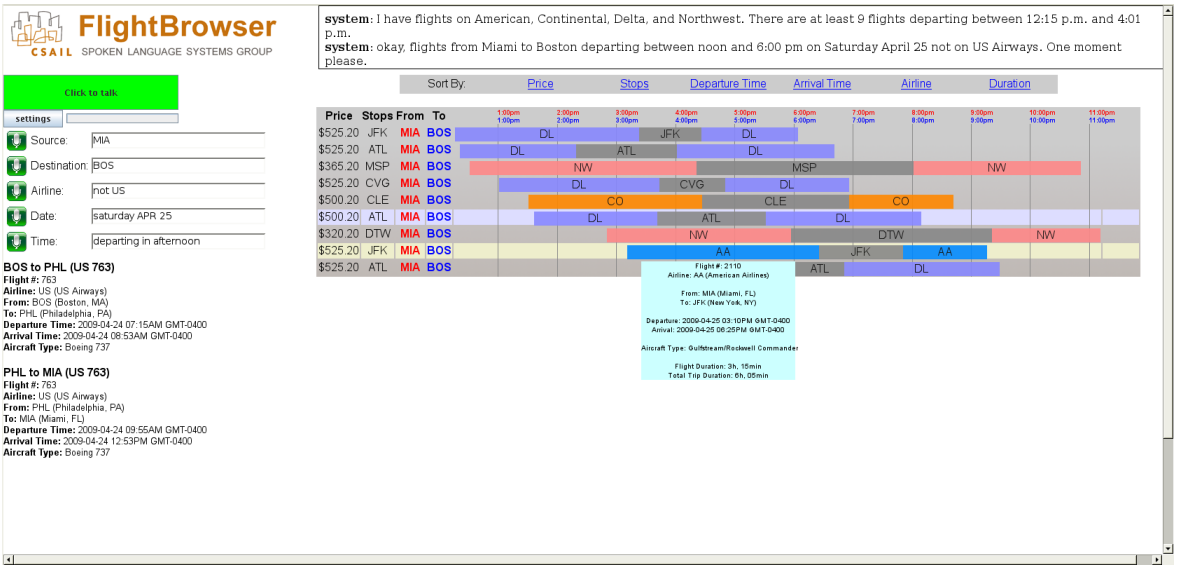


Figure 4-1: Screenshot of the *FlightBrowser* graphical interface, which consists of the speech input button (top-left), conversation box (top), concept text fields (left), concept recording buttons (far-left), partial itinerary (bottom-left), and flight information region (bottom-right).

teracts with the components described in the previous chapter. We will describe *FlightBrowser* starting from the outside, then continuing inward. That is to say, we will begin by describing the graphical layout of the GUI client, then continue by describing its multimodal interaction with the GUI server, and finally describe the interaction between the GUI server and other back-end components such as the database and spoken dialogue system. Furthermore, we will explain the process of implementing incremental understanding as a means of visual feedback to the human user.

4.1 Graphical Layout

In this section, we will describe specific features of the *FlightBrowser* graphical interface. A screenshot of the interface appears in Figure 4-1.

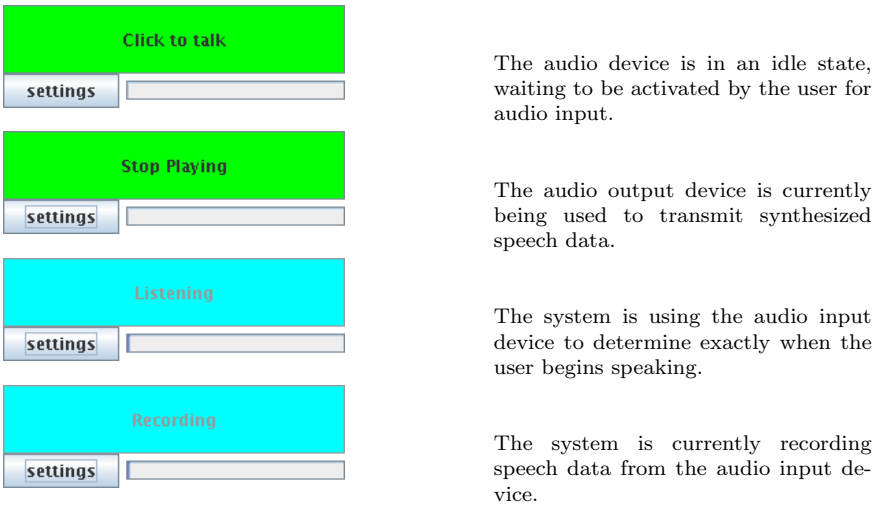


Figure 4-2: Speech input button states.

4.1.1 Speech Input Button

The speech input button is an embedded Java applet, positioned in the upper-left corner of the window, through which the user can instruct the system to begin recording speech input or to stop producing synthesized speech output. The button displays a text label and color coding for the current status of the audio device. The four possible states of the audio device—Click to Talk (Waiting), Stop Playing, Listening, and Recording—are outlined in Figure 4-2. When the audio device is in the Listening or Recording state, the amplitude meter monitors the level of the user's speech. This amplitude meter is a way to ensure that the audio device is working properly. If necessary, the settings can be adjusted to toggle between different audio input and output devices. Also, the settings can be adjusted to increase or decrease the amplitude of the audio input, or tweak the voicing threshold used for endpoint detection.

4.1.2 Conversation Box

The conversation box is a text area located at the top of the window that provides a text transcript of the dialogue between the system and the user. This text area is automatically updated at the end of a spoken utterance or a text input entry, or when

Figure 4-3: Screenshot of the concept recording button and text field for the "destination" concept.

the system responds with synthesized text output. Also, *intermediate replies*, which are not vocalized using the speech synthesizer, are displayed in the conversation box while the system accesses the third-party flight database.

4.1.3 Concept Text Fields

The concept text fields, located in the left side of the window, store information about concepts extracted from the user's speech. The domain-specific concepts displayed to the user are: source, destination, airline, date, and time. These text fields are automatically updated in real-time as MERCURY sends incremental key-value pairings for each concept. Once the user has finished speaking, these text fields are updated with context-resolved values. Another functionality of the concept text fields is the ability to type information about a specific concept rather than speaking. The system allows full names (Los Angeles) and abbreviations (LAX).

4.1.4 Concept Recording Buttons

In addition to the concept text fields, each concept has its own recording button that the user may press to begin recording. This is slightly different from merely pressing the speech input button because the additional context of the selected concept is sent alongside the speech data. For example, a user may click the recording button for the destination concept field and speak the one-word phrase "Atlanta." This feature allows phrases that would be ambiguous without additional context (Atlanta could have been either a source or destination) to be correctly interpreted with regard to the context. A concept recording button along with its corresponding text field is shown in Figure 4-3.

4.1.5 Partial Itinerary

The partial itinerary frame, located on the left panel of the window, provides a chronological list of all the flights that have been booked by the user in the active session. Each itinerary entry in the list contains important flight information including the source and destination airports, departure and arrival times in their respective local time zones, flight number, airline, and aircraft type. The entries are collapsible such that when the user clicks on the information for a specific flight, the information from that flight collapses into a more concise format.

4.1.6 Flight Information Region

The flight information region, occupying the largest window area of all graphical components, serves primarily to portray two different types of data to the user. The flight information region is used to display the current set of flights that meet the user's constraints, or the final itinerary when the user has finished selecting all flights for an entire trip.

First, the flight information region displays information about the current set of flights that the user may choose to add to the itinerary. When the system has a set of flights to present to the user, it is shown graphically in a table similar to Figure 4-4. The price, source and destination airports, and connecting airports of each flight are displayed in columns. Each flight is displayed as a horizontal bar whose length is proportional to the duration of the flight. The horizontal bar is divided into segments to indicate layovers and connecting flights. Furthermore, these segments are color-coded to distinguish layovers and distinct airlines. The user may scroll the mouse over a particular segment to retrieve more information about that segment, both for flight segments and layover segments. Also, the user may sort the graphical list of flights by price, number of stops, departure or arrival time, airline, or duration.

Secondly, the flight information frame is used to display the final itinerary once the user is done booking flights. Similarly to the partial itinerary displayed on the left, the final itinerary has specific information about each individual flight in the

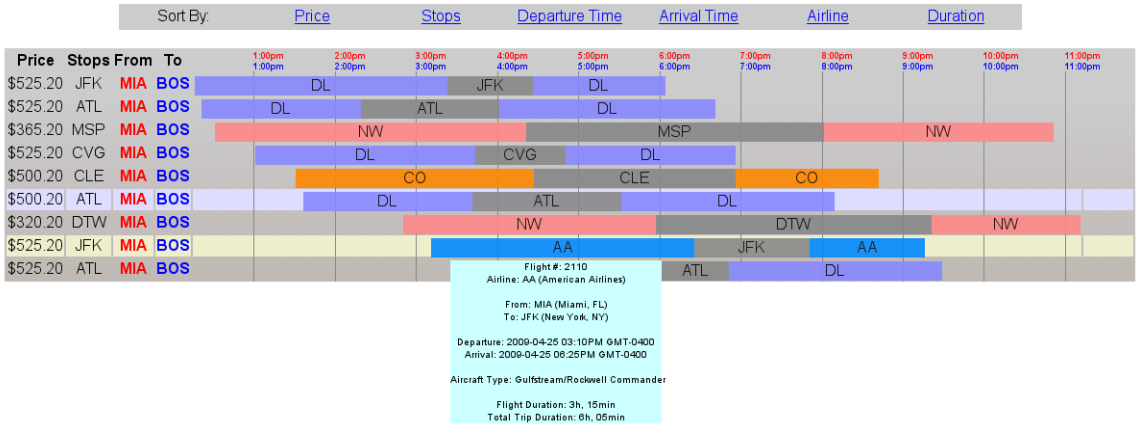


Figure 4-4: Example of a list of flights shown in the flight information region. Each flight is displayed as a horizontal bar whose length is proportional to the duration of the flight. The horizontal bar is divided into color-coded segments to indicate layovers and connecting flights. Users may select a flight by clicking, find out more information about a flight by hovering, or sort flights by clicking on a certain sorting hyperlink.

entire trip. See Figure 4-5 for an example of a final itinerary.

4.2 Event Handling and Messaging

In this section, we will describe the process by which a message is sent from the client side to the server side of the application, and vice versa, describing the steps that occur along the way and specific examples of such events. Figure 4-6 displays the chain of modules that pass along messages. This bidirectional chain of modules allows us to define two specific types of events in the *FlightBrowser* system. User-initiated events originate from the client-side of the module, usually involving the use of a specific modality; system-initiated events originate from the server-side, usually involving some change to the graphical layout.

4.2.1 User-Initiated Events

In order to carry out application-specific events, a message must be sent via the web server to communicate with the server side of the GUI. Oftentimes, these events are

Figure 4-5: Example of a final itinerary shown in the flight information region.

Figure 4-6: Messaging chain of the *FlightBrowser* system. User-initiated events propagate from client to server, whereas system-initiated events propagate from server to client.

driven by a user's interaction with the system, for example, the use of a multimodal feature in the interface. This user event sets off a chain of communication that extends from the browser and eventually propagates back to the spoken dialogue system.

The client side of the application consists of the graphical user interface and functions that handle specific events in the web application. This part of the application consists mostly of HTML and CSS to create the visual layout and JavaScript to handle dynamic changes to the layout and other behind-the-scenes functions. When the user initiates an event—for instance, clicking on a specific flight in the list of flights—parameters are sent to an XML message generator in the GUI controller, which in turn transmits the message over the web server to the server side of the GUI. Once the message is received by the GUI server, the message is then converted into the frame format and sent as input to the spoken dialogue system. The spoken dialogue system uses the content of this frame as additional context, or to execute an action.

In the *FlightBrowser* system, situations in which messages are passed from the front end GUI to the back end dialogue system include the multimodal actions of typing into a text field, clicking on a recording button, clicking on a flight, and sorting a list of flights.

4.2.2 System-Initiated Events

User-initiated events transmit information to the back end; however, the back end also has information to convey in the reverse direction. Various pieces of information must travel from the server side to the client side. System-initiated events generally originate from the spoken dialogue system as a frame. The information contained in this frame propagates along the chain by first parsing the information from the frame in the GUI server, then executing actions in the GUI server based on the content of the frame, and then packaging the information as an XML message, which is sent over the HTTP web server to the GUI controller on the client side. Finally, the GUI controller converts the contents of this message into graphical content by parsing the XML message.

Examples of system-initiated events include displaying a transcription string in the

conversation box, displaying a list of flights, showing a partial or final itinerary, and conveying visual feedback via incremental understanding and discourse resolution.

4.2.3 Audio Streaming Events

During audio streaming events, data is passed directly to the speech recognizer or synthesizer when streaming audio input or output, respectively, rather than through the messaging chain. Meanwhile, additional messages are passed via the messaging chain to the spoken dialogue system whenever the state of the audio device changes.

4.3 GUI Server Functionality

There are several functions that occur on the server side of the *FlightBrowser* interface. This section describes some of the server-side computation that takes place.

4.3.1 Database Access

There are primarily two situations in which we would like to connect to a third-party database for flight-related information. Database queries are performed when the system has obtained at least the source, destination, and date of a flight from user input. In addition, when the user is ready to receive the final itinerary for a complete trip, we contact the external database to determine the total price of the user's selected flights. When the system has enough information to perform a query, the dialogue system transmits a message to the GUI server, specifying the details of the current flight query (or, list of flights for the itinerary pricing case) as parameters. The database module uses these parameters to create an XML message that connects to the external database. This database returns with an XML message containing information about flights that meet the given constraints, or the price of a set of flights for the final itinerary. This message is repackaged into a frame that is sent to the GUI server for further processing.

4.3.2 Flight Information Extractor

After the frame containing flight information is sent from the database to the GUI server, some processing occurs that stores the frame of flights into a list of *trip objects*. Since a trip may require multiple flights to get from the source to the destination, each trip object stores an ordered list of *leg objects*, where each leg represents a piece of the trip separated by a stopover or connection. In addition to this list of flights, each trip object stores the total price, the number of legs, and a *trip index* designated by the spoken dialogue system to distinguish it from other trips. Leg objects contain information about the source and destination, departure and arrival times, airline, flight number, aircraft type, and duration. Trip objects can be sorted by a specific criterion. By default, trip objects are sorted by price, however, the user has the option to sort flights (trip objects) by a different criterion, if he or she so chooses.

4.3.3 Flight Code Hash Tables

In order to provide more meaningful output to the human user, we must substitute the IATA abbreviations we receive from the external flight information database with their lengthened names. The GUI server includes several hash tables for mapping an abbreviation to a full name. We use these hash tables to display full names for airlines, airports, and aircraft types. In the graphical display, the more commonly known full names are usually expressed in parentheses next to their abbreviated counterparts in partial and final itineraries. This allows us to convey both the abbreviations and full names of destinations and airlines, data that the MERCURY system does not provide to the user on its own (i.e. without graphics). Additionally, if the user types an abbreviation into a text field, these hash tables are used to convert the abbreviation into a lengthened name to be sent to the spoken dialogue system.

4.3.4 Incremental Aggregator

The *FlightBrowser* GUI server records the system's current understanding of domain-specific concepts through the process of incremental understanding and discourse

resolution, which we will discuss in the next section. Several incremental key-value frames are sent from the spoken language system to the GUI server during the user's spoken utterance, and a final context-resolved frame is sent at the end of the utterance. The GUI server stores a hash table that maps a concept to its value. The source, destination, airline, and date key-values map directly to stored concepts in the GUI server. However, there are several key-value pairings from the spoken dialogue system that represent the time. The time can be exact or inexact; and it can be a departure time, arrival time, or neither. The GUI server stores the key accordingly based on the type of pairing used to represent the time. Furthermore, phrases like "in the afternoon" are stored in the incremental aggregator as "afternoon," but represented in the spoken dialogue system as a fixed time window (in this case, afternoon is converted to "between noon and 6:00 p.m.").

4.4 Incremental Understanding and Context Resolution

In this section, we will discuss the process by which incremental understanding and discourse resolution occur and how this information gets sent to our incremental aggregator.

4.4.1 Incremental Recognizer Updates

In a system that does not implement incremental understanding, the N-best hypotheses for the processed speech are not determined until the end of the utterance. The difference between the incremental capability in SUMMIT and speech recognizers without this capability is that in SUMMIT, the N-best hypotheses are evaluated while the user is still speaking. For a complete utterance, SUMMIT performs a Viterbi backtrace. However, when performing a Viterbi backtrace on an incomplete utterance, we relax the constraint that the ending state is a terminal state [16]. Relaxing this constraint allows the recognizer to send partial updates after each audio

chunk instead of waiting for the utterance to be completed. There is one issue with sending these incremental updates from the recognizer. The final word of the incomplete sentence may be truncated, which often leads to recognition errors in the incremental updates, which spill over into the graphical display. However, despite the likelihood of error on the final word, the error is usually self-corrected by the end of the utterance, once more audio has been streamed to the recognizer and the truncated word becomes complete.

4.4.2 Incremental NL Understanding

After the partial hypothesis is decoded, the partial sentence gets sent to TINA, our natural language understanding module [35]. The process of extracting keys and values from a decoded user utterance required us to create a special key-value extraction language in the MERCURY domain. TINA incorporates this extraction language to parse an incomplete utterance into a set of keys and values, which correspond to the concepts that our system conveys graphically. Finally, these keys and values are packaged into an incremental key-value frame, and sent to the incremental aggregator in the GUI server for storage and manipulation. See Figure 4-7 for a series of incremental key-value frames.

4.4.3 Context Resolution

After the user utterance is fully decoded by the recognizer, the NL system processes the resulting N-best list and produces a key-value representation of the hypothesis it selects. The final key-value frame of the completed utterance is sent to the discourse module for context tracking. After a series of context resolution rules are carried out, the discourse is updated, forgetting or remembering previous context items as deemed necessary. The cities and airlines are replaced with their corresponding IATA abbreviations, the date is replaced with an exact date, and the time is converted to a precise interpretation. The resulting frame is sent to the GUI server and is handled similarly to the key-value frame supplied via incremental understanding. See Figure

"⓿I would like a flight❶ from Houston❷ to Las Vegas❸ on Friday❹ morning❺."

```
{c handle_partial
   ...
   :partial_frame {c eform
                    :clause "no_parse"
}}⓿

{c handle_partial
   ...
   :partial_frame {c eform
                    :clause "maybe_book"          ❶❷❸❹❺
                    :flight_quant "indef"         ❶❷❸❹❺
                    :source "Houston"             ❷❸❹❺
                    :destination "Las Vegas"      ❸❹❺
                    :date "friday"                ❹❺
                    :when "morning"               ❺
}}
```

Figure 4-7: A series of incremental (partial) key-value frames for the sentence, "I would like a flight from Houston to Las Vegas on Friday morning." The first key-value frame, labeled 0, is a *no-parse*, since the sentence does not contain enough information for parsing. Subsequent incremental key-value frames (1-5) are sent while the sentence is spoken and more information becomes available for parsing. Later frames contain more information than previous frames.

4-7 for an example of post-utterance context resolution.

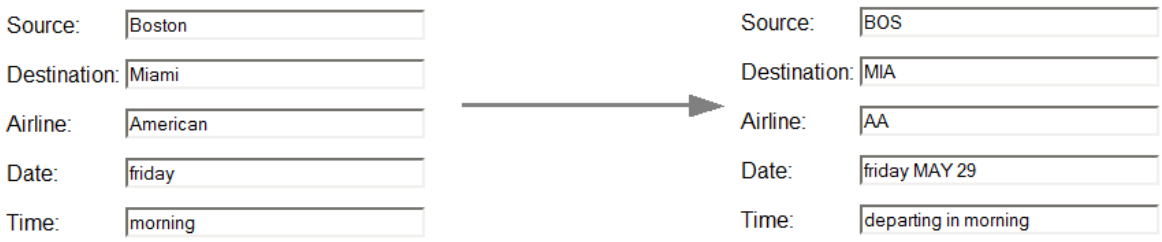


Figure 4-8: Example of post-utterance context resolution based on incremental key-value pairings.

4.5 Chapter Summary

In this chapter, we have presented details about the *FlightBrowser* system from both the client and server side. This system has several advantages over the speech-only MERCURY system that it extends. One advantage is that the user does not have to commit a large quantity of information to memory, as in a unimodal system. On the telephone-based MERCURY system, a user would most likely need to write down information about the active selection of flights in order to make an informed decision. Similarly, a user would have to retain information about flights that have already been added to the itinerary. The graphical layout lightens the user's cognitive load through its visual output.

FlightBrowser is an information-rich display that lessens the amount of time spent streaming synthesized audio. The system remains conversational; however, the system does not need to "ramble" in order to recite large amounts of information since that information can be conveyed visually rather than verbally. The most notable example of an information-rich feature is the graphical list of flights, which uses horizontal bars proportional to the duration of the flight and color-coding to produce a *glanceable* environment, taking advantage of the human ability to quickly decode visual stimuli. Incremental understanding is an additional method for exploiting the human capacity for visual stimuli, allowing the user to confirm that the system has satisfactorily

understood his or her speech before listening to the system's response.

In the next chapter, we will describe the results of a comparative user study that tests two different configurations of our system; the following chapter will include a discussion of these findings.

Chapter 5

User Study

We would like to explore the effects of displaying the system's current understanding to the user while he or she attempts to perform flight reservation tasks in our system. This user study is similar to the experiments in the Fruit Carts domain presented by Aist et al. in [2]. In their experiments, the dialogues of 22 subjects were collected—11 from a nonincremental system and 11 from an incremental system. Further analysis indicated that subjects using the incremental system completed tasks more quickly, and also that there is a positive correlation between incremental system use and user satisfaction.

In this user study of the *FlightBrowser* system, we would like to compare a baseline configuration of the system with an incremental configuration. We analyze the results based on rate at which tasks are completed and ratings given to the system by subjects after using the system. Also, we elaborate on key differences in system feature usage between the two groups.

5.1 User Study Setup

This section describes the setup of our user study, going into detail about the user groups, different configurations (baseline and incremental), and flight reservation assignments. We also describe the user group management utility by which a user signs up for a study and performs a set of tasks.

5.1.1 Subjects and User Groups

To set up our experiment, we gathered 20 subjects—composed of students and faculty from the MIT community—and randomly divided these subjects into two user groups containing 10 subjects each. The groups were divided as such to compare two different configurations of the *FlightBrowser* system. One group makes use of a baseline configuration that lacks the text fields that provide incremental visual feedback of specific concepts and subsequently, their context-resolved values. The second group uses an incremental configuration that includes the text fields that provide incremental feedback and context resolution. Figures 5-1 and 5-2 provide examples of the two layouts tested in our user study. All users were brought to the Spoken Language Systems (SLS) laboratory for the experiments. After completing the user study, each subject was compensated with a $10 Amazon gift certificate.

5.1.2 User Group Management

This user study involved using a utility, developed by Yoshimoto of SLS, that allows us to assign users to groups using the baseline or incremental configuration to keep track of each user's progress using the *FlightBrowser* system [43]. This management tool allowed us to design our user study by creating a specific sequence of variables that get entered into the query string in the *FlightBrowser* URL; one of these variables determines whether our system is configured with a baseline or incremental layout.

A user logs into the system with his or her email address as shown in Figure 5-3. Each user group has its own login screen. Once the user logs in, we assign a unique user ID and the specific group ID based on the particular login screen; we use these IDs rather than email addresses to identify users during data analysis. The user sees a set of five tasks to be completed: four of which are flight reservation assignments, and a final task that is a user evaluation survey.

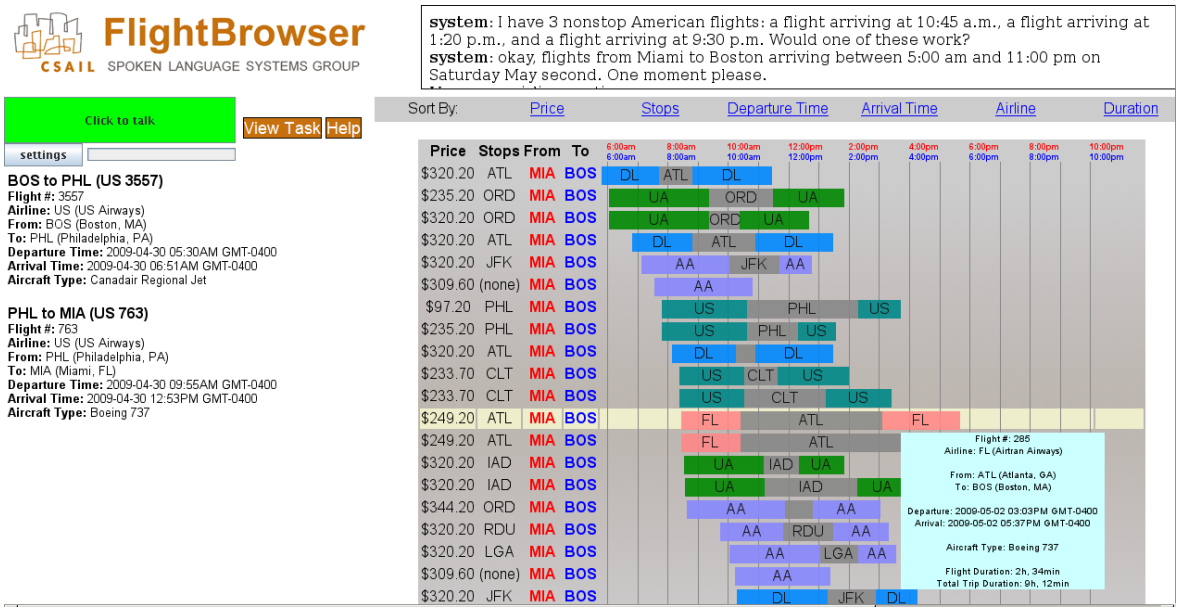


Figure 5-1: Baseline configuration.

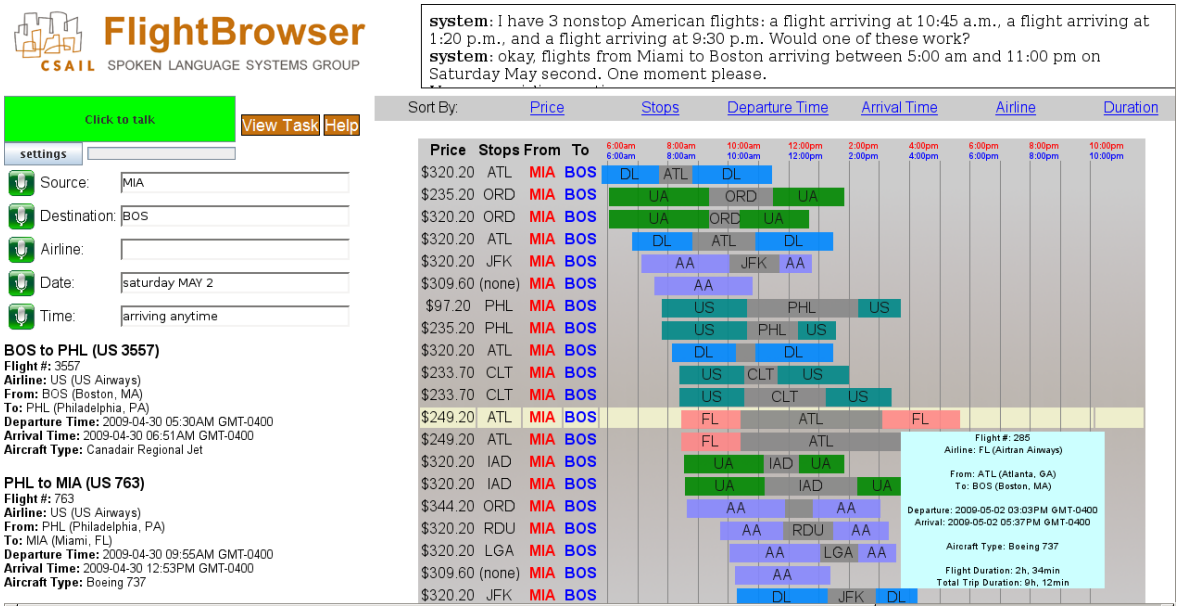


Figure 5-2: Incremental configuration.

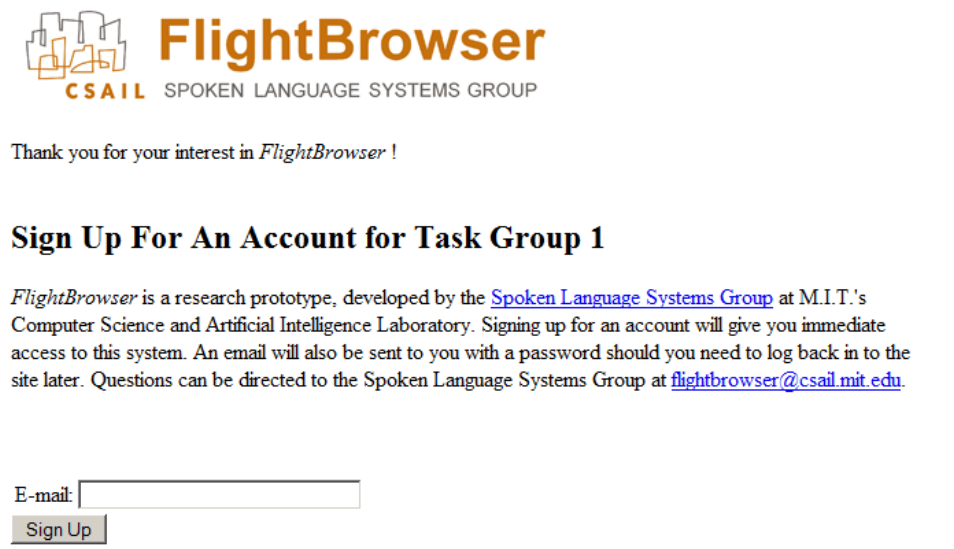


Figure 5-3: Login screen for user study participants.

Figure 5-4: Instruction box displayed at the beginning of a user session.

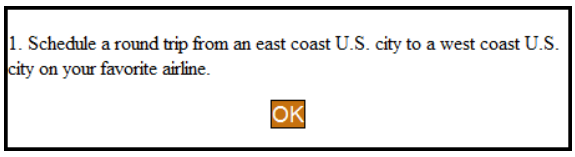


Figure 5-5: Example of a task box displayed at the beginning of a user session.

5.1.3 Instructions

Once the user has logged into the system, and has received the set of assigned tasks, the user clicks on a specific task, and elects to "Start Assignment." This loads the *FlightBrowser* graphical interface along with two pop-up boxes that appear at the beginning of a task. Users are naïve in the sense that they know nothing about the system prior to using the system, other than a brief overview of its main functionality, i.e., it is a graphical flight scheduling interface. The first pop-up box provides a set of general instructions for using the system, as shown in Figure 5-4. The second pop-up box includes the specific task to be performed for this session, which we will enumerate in the next section (also see Figure 5-5). Furthermore, we include two buttons that allow the user to view the general instructions or assigned task at any point during the current session.

5.1.4 Assigned Tasks

Both the baseline and incremental groups are given an identical set of four tasks, and asked to perform as many tasks as possible in a span of 25 minutes. A task is considered to be completed when the user has obtained a final itinerary from the system that satisfies the given task. These tasks are designed such that the difficulty increases as the user becomes more familiar with the system. These tasks must be performed in order unless the subject absolutely gives up on completing a specific task, and elects to skip it. The tasks, presented in their proper order, are as follows:

1. Schedule a round trip from an east coast U.S. city to a west coast U.S. city on your favorite airline.

2. You are looking for the cheapest round trip from a northern U.S. city of your choice to a southern U.S. city of your choice. You would prefer non-stop flights for both the forward flight and the return flight.

3. You would like to take a vacation in Europe for an indefinite amount of time. Set up a one-way flight from a U.S. city to a city in Europe.

4. You have a busy weekend ahead of you. You have a business conference to attend on Friday afternoon in Chicago. Then, you must attend your best friend's wedding in Baltimore on the following day. Finally, you must return home so that you can get to work on Monday morning. Schedule a trip that takes you from home to the business conference, then to the wedding, then finally back home.

5.2 Significance Testing

For the remainder of this chapter, we will analyze the results of this user study. In order to quantify the statistical significance of some of the comparative results presented here, we will conduct a two-tailed t-test with 18 degrees of freedom, denoted as ν. This particular test of statistical significance was chosen due to the relatively small sample size and also due to the fact that the mean and standard deviation of our two samples is unknown *a priori*. The value of ν is calculated by taking the total sample size of 20 across both user groups, then subtracting 2 from that number since the test is two-tailed. Using the t-distribution with ν=18, we evaluate the t-statistic and its resulting p-value to determine statistical significance for each user group comparison. A lower p-value is an indication of higher statistical significance.

5.3 System Usage (Baseline vs. Incremental)

In this section, we will compare system feature usage of subjects using the baseline versus those who used the incremental system. Overall, we found that those using the baseline system were more likely to use the features of history revision, flight clicking, and flight sorting, which we will discuss below. Table 5.1 and the left side of Figure 5-6 provide data on the usage of system features shared between the two user groups. A user is given a value of 1 if the feature was used at least once during the user session, and given a value of 0 otherwise.

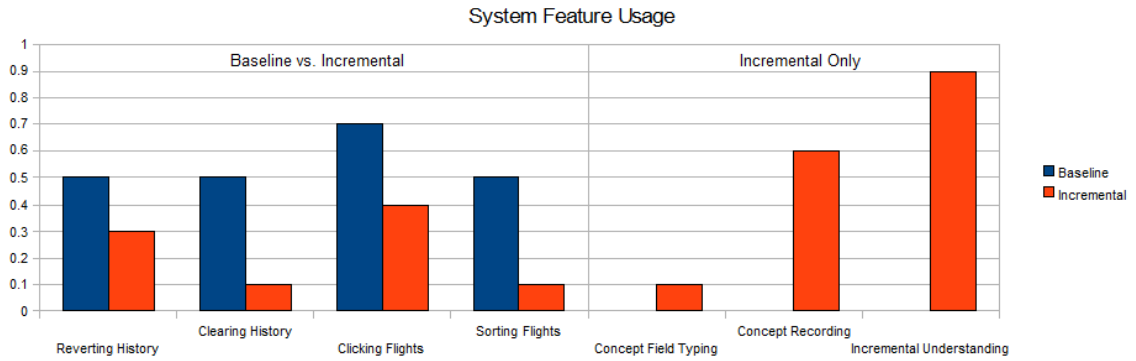


Figure 5-6: System feature usage results from user study. This graph shows the percentage of users that used a specific feature in the FlightBrowser system. The left side of the graph shows the usage of features that were available in both the baseline and incremental configurations, whereas the right side of the graph shows features that were exclusively available in the incremental configuration.

Feature	Baseline Mean	Incremental Mean	Baseline Std. Dev.	Incremental Std. Dev.	t-statistic	Two-tailed p-value
Reverting History	0.5	0.3	0.53	0.48	0.88	0.39
Clearing History	0.5	0.1	0.53	0.32	2.06	0.05
Clicking Flights	0.7	0.4	0.48	0.52	1.34	0.20
Sorting Flights	0.5	0.1	0.53	0.32	2.06	0.05

Table 5.1: System feature usage (baseline vs. incremental).

5.3.1 Reverting and Clearing History

In this user study, we found that those using the baseline system were more likely than incremental users to utilize history revision. There are two types of history revision events: reverting history (i.e., "going back") or clearing history (i.e., "starting over"). These events are usually used when the system makes a mistake and attempts to act on this misunderstood speech. The more frequent usage of history revision events suggests that those using the baseline configuration were more likely to encounter errors that they do not feel comfortable trying to correct with a follow-up utterance.

5.3.2 Clicking and Sorting Flights

Furthermore, we found that subjects using the baseline configuration were more likely to use the flight information table in the display to click on flights to refer to them through speech. Also, those using the baseline system were much more inclined to use the sorting feature for the flight information table. This indicates that those using

the incremental system were more likely to allow the system to suggest flights since it could visualize what the system currently understands. Those using the baseline system usually carried out less specific queries, then narrowed their search down using sorting and clicking.

5.4 System Usage (Incremental Only)

In this section, we discuss the frequency of usage for several features that were exclusively available in the incremental configuration of the *FlightBrowser* system. This includes concept field typing and recording, and also the usage of visual feedback provided through incremental understanding and context resolution. Table 5.2 and the right side of Figure 5-6 provide data on the usage of system features that are exclusive to the incremental configuration. Like in the previous section, a user is given a value of 1 if the feature was used at least once during the user session, and given a value of 0 otherwise.

Feature	Mean	Std. Dev.
Concept Field Typing	0.1	0.32
Concept Recording	0.6	0.52
Incremental Understanding	0.9	0.32

Table 5.2: System feature usage (incremental only).

5.4.1 Concept Field Typing

Out of the 10 participants who interacted with the incremental version of the system, only one participant used the typing feature. Most users were not aware of this feature, though it was exemplified in the initial instructions provided when the user loads the page. One reason could be that since these text fields are shared with output from *FlightBrowser*'s incremental understanding and discourse resolution, it is not intuitively obvious that these text fields can also be used for input.

5.4.2 Concept Recording

Five out of the ten subjects using the incremental system used the small microphone icons for concept recording. There were two different styles of using the concept recording buttons. Some used the concept recording buttons to make a single correction when a misrecognition occurs, whereas others used the concept recording buttons nearly exclusively, interacting using one or two word utterances for each concept rather than using the main speech input button and speaking a much longer sentence.

5.4.3 Incremental Understanding

Ninety percent of the users stated that they noticed the text fields being filled in real-time as they were speaking or post-utterance, using this information to check the system's current understanding. Based on the higher usage of history reversion and clearing shown in the baseline group, we can hypothesize that being able to perceive the system's visual understanding prevented incremental configuration users from having to go backward in history during a session. The large percentage of users who noticed the visual output of the system's understanding may explain why less users in the incremental group necessitated the use of history revision. Furthermore, it may explain why the baseline users who lacked incremental understanding more frequently used some of the multimodal features such as clicking and sorting flights.

5.5 Task Completion

Subjects who interacted with the baseline system had less information from which to verify the system's understanding. As a result, several baseline users ran into errors which forced them to start over or go backward in history. Two baseline users were only able to complete one task, possibly because it is harder to assuage an error when it occurs in the baseline system. Overall, users of the incremental system were able to complete more tasks in the 25 minute period than the baseline group. One can

argue that being able to see the system's understanding prevents having to revert to a prior state in the system, and makes it easier for users to complete tasks. See Table 5.3 and Figure 5-7(a) for data on task completion.

	Baseline Mean	Incremental Mean	Baseline Std. Dev.	Incremental Std. Dev.	t-statistic	Two-tailed p-value
Task Completion	3.3	3.8	1.25	0.42	1.2	0.25

Table 5.3: Task completion (out of 4 tasks) between baseline and incremental groups.

5.6 Post-Study Evaluation Survey

In this section, we will discuss the results of a post-study evaluation survey. The survey is given immediately after the user has completed all tasks or has exceeded the time allotment. The user rates a certain aspect of our system on a scale of one to five, where five represents the most favorable score. We present these user-evaluated aspects of the system in order of increasing statistical significance. However, due to the small number of users, none of this data is *statistically significant*, since $p >> 0.05$. See Table 5.4 and Figure 5-7(b)-(f) for user evaluation data.

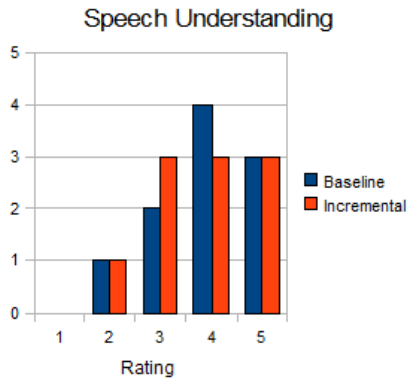
Feature	Baseline Mean	Incremental Mean	Baseline Std. Dev.	Incremental Std. Dev.	t-statistic	Two-tailed p-value
Speech Understanding	3.9	3.8	0.99	1.03	0.22	0.83
Usability	3.4	3.3	0.97	0.82	0.25	0.81
Graphical Appeal	3.8	3.5	0.92	1.51	0.54	0.60
Task Difficulty	4.3	4.5	0.82	0.53	0.65	0.53
Overall Rating	3.0	3.5	1.41	0.85	0.96	0.35

Table 5.4: System ratings given by baseline and incremental groups during post-usage evaluation.

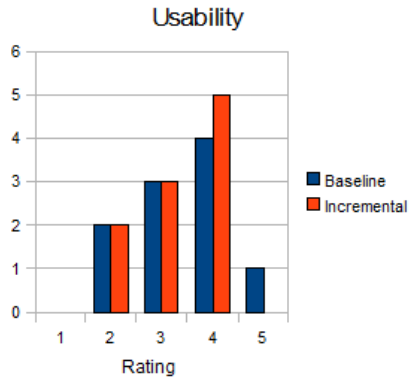
5.6.1 Speech Understanding

Each user was asked to indicate how well our speech understanding performed during the user study. Both groups rated the speech understanding of our system similarly. In this case, it seems that the incremental configuration did not significantly impact a user's opinion about the speech understanding ability of our system.

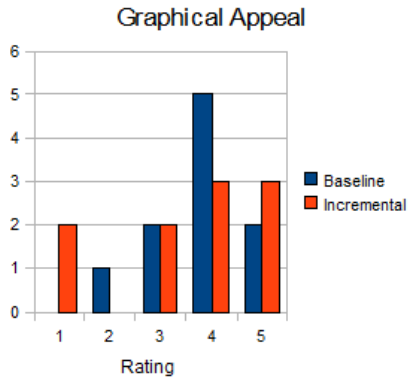
(a) Task Completion



(b) Speech Understanding

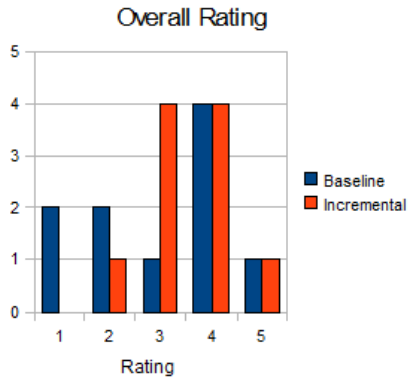


(c) Usability



(d) Graphical Appeal

(e) Task Difficulty



(f) Overall Rating

Figure 5-7: Task completion and user evaluation graphs. Graph (a) shows the number of users that completed a certain tasks from each user group. Graphs (b)-(f) show the number of users that provided a certain rating in the post-usage survey for speech understanding, usability, graphical appeal, task difficulty, and overall rating.

5.6.2 Usability

Each user was also asked to rate the usability of the *FlightBrowser* system. Like the speech understanding responses, both groups provided similar answers for usability. This suggests that the incremental configuration did not significantly impact the user's opinion about the usability of the system. However, this could also be a result of incremental users not fully utilizing the multimodal features. Maybe if an incremental user exploited more the multimodal features, he or she would have given the system a higher usability rating.

5.6.3 Graphical Appeal

For graphical appeal, baseline users were more inclined to give a high rating. This may suggest that even though our incremental system is more functional and leads to a higher task completion rate, its layout may be less appealing visually. Also, as noted by the data in Tables 5.3 and 5.4, many incremental users did not fully utilize *FlightBrowser*'s multimodal capability. Baseline users seemed more likely to explore its multimodal features, although the incremental group had access to the same features, plus additional features. Perhaps the extra cognitive load of having more input choices for audio recording led to the lower score in the incremental group.

5.6.4 Task Difficulty

Users of the incremental configuration gave a generally higher rating for tasks. That is to say, incremental users found tasks to be slightly easier than those in the baseline group. This data seems consistent, considering that the incremental group were able to perform more tasks in the allotted 25-minute time period.

5.6.5 Overall Rating

Overall, users that interacted with the incremental system were more likely to give the system a high rating. This was the most statistically significant result of the

user evaluation, and thus our most profound distinction between the two user groups. Incremental users may have provided a higher overall rating as a result of the higher completion rate and lesser perception of task difficulty. We will explore the correlation between these statistics in the next section.

5.7 Correlation

For further illustration, we have calculated the correlation between some of our most significant distinctions in the two user groups. We have selected the task completion rate, task difficulty, overall rating, and usage of history clearing for cross-correlative comparisons, depicted in Table 5.5.

	Task Completion	Task Difficulty	Overall Rating	Clear History Usage
Task Completion	–	0.62	0.44	-0.14
Task Difficulty	–	–	0.40	-0.27
Overall Rating	–	–	–	-0.07
Clear History Usage	–	–	–	–

Table 5.5: Correlation between several statistics collected from user study data.

In our results, there appears to be a (weak) negative correlation between using the history clearing (start-over) feature and the values for task completion, task difficulty, and overall rating. This is expected since having to start over indicates a negative performance of our system. The most outstanding correlation involving history clearing is the negative correlation between history clearing and task difficulty (where a high rating represents easier tasks). Therefore, users that carried out a history clearing command found their tasks to be more difficult.

Furthermore, we found a moderate positive correlation between the overall rating and task completion, meaning that those who completed more tasks were more inclined to rate the system highly. There is also a moderate positive correlation between overall rating and task difficulty; users who found the tasks to be easy also gave the system a higher overall rating. However, the strongest correlation that we found in

our results was the positive correlation between task completion and task difficulty—those who completed fewer tasks also expressed that these tasks were rather difficult.

5.8 Chapter Summary

In this chapter, we have documented the results between the baseline and incremental configurations of the *FlightBrowser* system. When taking system usage into consideration, baseline users were more likely to revert or clear the history when confronted with recognition errors in the system, which may indicate that incremental feedback makes the user more confident with the system's speech recognition and understanding. Furthermore, baseline users were more likely to use the features of flight clicking and sorting than incremental users. The clear distinction between two groups in the usage of flight sorting and clicking show that incremental users, though presented with more multimodal features, were more comfortable with using speech exclusively to communicate with the system.

Additionally, an incremental user, completed more tasks on average than a baseline user, possibly due to the visual understanding conveyed on the screen. Perhaps as a result of this higher rate of task completion, incremental users also rated the system more highly overall, and found tasks to be easier to complete. In the next chapter we will discuss the implications of the work presented in this thesis and our user study findings, with suggestions for future directions of research.

Chapter 6

Discussion

In this thesis, we have described the transformation of a telephone-based spoken dialogue system into a more functional web-based version. The MERCURY flight reservation system in its telephone-based form was limited in its capability, mainly because of its absolute dependence on spoken input and output, which incurs restrictions on the type of information that can be understood or conveyed. One example of a restriction in its telephone-only version is the number of flights that the system provides as for the user to choose from. In the unimodal system, we could only reasonably describe four flights at a time to the user. Presenting more than this amount would require excessive memorization from the user, which is undesirable for any conversational interface. Additionally, it would take extensive speech synthesis to convey this information to the user. However, now that we have developed a graphical display upon which to store information, we have expanded the number of flights from four to fifty, and eliminated the need for the user to memorize flights.

Another example of the enhanced capability provided by our graphical interface is the process of incremental understanding, by which the user receives real-time visual feedback of the system's comprehension. Encountering similar results to the Aist et al. experiment [2], our user study reveals that there is an added benefit to making the user aware of what the system is "thinking." The advantage of sending visual feedback of the system's understanding manifests itself in the rate of task completion and overall user satisfaction.

Based on our results, we can hypothesize that conveying the system's understanding on the screen gives the user more clarity and confidence regarding the accuracy of our speech recognition and natural language understanding, as shown by the less frequent usage of history revision commands among incremental users. Another distinction between the two groups was the more pervasive usage of multimodal features in the baseline group that did not receive incremental understanding. This shows that perhaps incremental users were comfortable interacting with the system exclusively with speech when the system's understanding was visible.

6.1 System Improvements

As a result of text-based responses in the user survey, there are several changes that we would like to implement in the near future to further enhance the capability of the *FlightBrowser* system. One major change that we would like to implement is to replace our current speech synthesizer. During the user study, many users commented on the robotic-sounding voice produced by the DECtalk [22] synthesizer. Thus, we would like to utilize a more user-friendly speech synthesizer, which may arguably improve the user experience.

In addition, our spoken dialogue system has several discrepancies as a result of the more extensive data provided graphically. For instance, although the system provides prices for all flights listed, this information is known only by the GUI server, but not the dialogue system. Therefore, if a user desires to add the cheapest flight to the itinerary, he or she cannot verbally ask for the cheapest available flight. This problem is an artifact because the original system was unable to look up the price until the entire itinerary was finalized. Therefore, the user must instead select that flight by sorting the graphical list, selecting the cheapest flight, then referring to it by speech. This duality is certainly nonintuitive for a naïve system user, so appropriate changes must be made to accommodate queries that involve price.

We would also like to implement several new multimodal features as an improvement to the graphical interface. One implementation is allowing the user to double-

Figure 6-1: *FlightBrowser* on an Apple iPhone

click on a specific flight to add it to the itinerary, rather than single-clicking on a flight and referring to it through speech. Additionally, adding buttons to the interface for reverting and clearing the history would be beneficial. In both of these cases, we would still interact with the spoken dialogue system in the back end using natural language (e.g. "book this flight," "start over," "go back"), but this information will be hidden from the user. This relieves the user from having to provide speech input for such commands.

6.2 Portable Devices

However, this raises doubts to the importance of speech input. If input can be provided effectively through other modalities such as typing and mouse clicking, how necessary is the speech input? Although it can be argued that alternative modalities may work more effectively on a standard computer, on a small portable device such as a smart cell phone, speaking is often the quickest way to send information to the system. On these devices, the additional features for typing, pressing buttons, and

touching the screen serve the purpose of supplementing the more essential speech information. Since smart phone interfaces are more speech-centric, an integral part of future development in the *FlightBrowser* system should be to increase its compatibility with mobile devices. Figure 6-1 shows a picture of the *FlightBrowser* interface, viewed on an Apple iPhone.

6.3 Incremental Understanding and Back-Channeling

In addition to improving its compatibility with portable devices, there are several possible directions for the *FlightBrowser* project with regards to incremental understanding. An alternative way of implementing incremental understanding would be to portray its understanding using symbols rather than text. Perhaps, a user may respond more favorably to a symbolic representation of its understanding rather than a textual one. Futhermore, a future implementation of incremental understanding may involve adding a videorealistic speech-guided facial animation [33] to the graphical layout, which may use facial expressions, and even vocalizations, as a way of incrementally back-channeling as the user is speaking.

Appendix A

User Study Data

In this appendix, we provide the complete numerical data from the user study described in Chapter 5, documenting the task completion, system feature usage, and user evaluation results. For convenience, the users have been sorted by group ID, for which a 0 represents a baseline system user, and a 1 represents an incremental system user.

User ID	**Group ID**	Reverting History Used	Clearing History Used	Clicking Flights Used	Sorting Flights Used	Concept Typing Used	Concept Recording Used	Incremental Understanding Noticed	Task Completion (0-4)	Speech Understanding (1-5)	Usability (1-5)	Graphical Appeal (1-5)	Task Difficulty (1-5)	Overall Rating (1-5)
1	**0**	1	1	1	0	N/A	N/A	N/A	1	4	2	3	3	1
2	**0**	1	1	0	0	N/A	N/A	N/A	4	4	3	5	5	3
3	**0**	0	0	1	0	N/A	N/A	N/A	4	3	4	4	5	4
4	**0**	0	0	0	0	N/A	N/A	N/A	1	3	3	2	3	2
5	**0**	0	0	1	1	N/A	N/A	N/A	4	5	4	4	4	4
6	**0**	1	1	1	1	N/A	N/A	N/A	4	4	3	4	4	4
7	**0**	1	1	1	1	N/A	N/A	N/A	3	2	2	3	4	2
8	**0**	0	0	0	0	N/A	N/A	N/A	4	5	4	4	5	1
9	**0**	1	1	1	1	N/A	N/A	N/A	4	4	4	4	5	4
10	**0**	0	0	1	1	N/A	N/A	N/A	4	5	5	5	5	5
11	**1**	0	0	0	0	0	1	1	4	5	4	4	4	3
12	**1**	0	0	0	0	0	1	1	4	3	4	3	4	3
13	**1**	1	1	0	1	1	1	1	3	4	4	3	5	4
14	**1**	1	0	1	0	0	1	1	4	3	3	4	4	4
15	**1**	1	0	0	0	0	0	1	4	4	2	1	4	3
16	**1**	0	0	1	0	0	1	1	4	3	3	5	5	4
17	**1**	0	0	0	0	0	0	1	4	5	2	1	5	2
18	**1**	0	0	0	0	0	0	1	4	4	3	4	5	3
19	**1**	0	0	1	0	0	1	1	4	2	4	5	4	4
20	**1**	0	0	1	0	0	0	0	3	5	4	5	5	5

Table A.1: Complete user study data. A group ID of 0 represents a baseline system user, and a Group ID of 1 represents an incremental system user.

Bibliography

[1] Gregory Aist, James Allen, Ellen Campana, Lucian Galescu, Carlos A. Gómez Gallo, Scott C. Stoness, Mary Swift, and Michael Tanenhaus. Software architectures for incremental understanding of human speech. In *Proceedings of the Ninth International Conference on Spoken Language Processing (INTERSPEECH 2006 - ICSLP)*, Pittsburgh, Pennsylvania, September 2006.

[2] Gregory Aist, James Allen, Ellen Campana, Carlos Gómez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus. Incremental dialogue system faster than and preferred to its nonincremental counterpart. In *Proceedings of the 29th Annual Meeting of the Cognitive Science Society (CogSci-07)*, Nashville, Tennessee, August 2007.

[3] Gregory Aist, James Allen, Ellen Campana, Carlos Gómez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus. Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods. In *Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue (SEMDIAL)*, pages 149–154, Trento, Italy, May 2007.

[4] Srinivas Bangalore and Michael Johnston. Robust multimodal understanding. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Montreal, Canada, May 2004.

[5] Lauren Baptist and Stephanie Seneff. GENESIS-II: A versatile system for language generation in conversational system applications. In *Proceedings of International Conference on Spoken Language Processing (ICSLP 2000)*, Beijing, China, October 2000.

[6] Nick Chater, Martin Pickering, and David Milward. What is incremental interpretation? *Edinburgh Working Papers in Cognitive Science*, 11:1–22, 1995.

[7] Li Deng, Kuansan Wang, Alex Acero, Hsiao-Wuen Hon, Jasha Droppo, Constantinos Boulis, Ye-Yi Wang, Derek Jacoby, Miland Mahajan, Ciprian Chelba, and Xuedong D. Huang. Distributed speech processing in MiPad's multimodal user interface. In *IEEE Transactions on Speech and Audio*, volume 10, pages 605–619. 2002.

[8] Edward Filisko and Stephanie Seneff. A context resolution server for the galaxy conversational systems. In *Proceedings of EUROSPEECH-2003*, pages 197–200. Geneva, Switzerland, September 2003.

[9] Gernot A. Fink, Christoph Schillo, Franz Kummert, and Gerhard Sagerer. Incremental speech recognition for multimodal interfaces. In *Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society*, Aachen, Germany, September 1998.

[10] Carlos Gómez Gallo, T. Florian Jaeger, James Allen, and Mary Swift. Production in a multimodal corpus: How speakers communicate complex actions. In *Proceedings of Language Resources and Evaluation Conference (LREC-08)*, Marrakech, Morocco, May 2008.

[11] Carlos Gómez Gallo, T. Florian Jaeger, and Ron Smyth. Incremental syntactic planning across clauses. In *Proceedings of the 30th Annual Meeting of the Cognitive Science Society (CogSci-08)*, Washington, D.C., 2008.

[12] James Glass, Jane Chang, and Michael McCandless. A probabilistic framework for feature-based speech recognition. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, pages 2277–2280, Philadelphia, Pennsylvania, October 1996.

[13] James R. Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech and Language*, 17:137–152, 2003.

[14] James R. Glass, Timothy J. Hazen, and I. Lee Hetherington. Real-time telephone-based speech recognition in the JUPITER domain. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Phoenix, Arizona, 1999.

[15] A.L. Gorin, G. Riccardi, and J.H. Wright. How may I help you? *Speech Communication*, 23:113–127, 1997.

[16] Alexander Gruenstein. Shape Game: A multimodal game featuring incremental understanding. Term project, 6.870: Intelligent Multimodal Interfaces, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 2007. http://people.csail.mit.edu/alexgru/shapegame/.

[17] Alexander Gruenstein, Bo-June (Paul) Hsu, James Glass, Stephanie Seneff, Lee Hetherington, Scott Cyphers, Ibrahim Badr, Chao Wang, and Sean Liu. A multimodal home entertainment interface via a mobile device. In *Proceedings of the Association for Computational Linguistics (ACL) Workshop on Mobile Language Processing*, Columbus, Ohio, United States, June 2008.

[18] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the International Conference on Multimodal Interfaces (ICMI)*, Chania, Crete, Greece, October 2008.

[19] Alexander Gruenstein, Jarrod Orszulak, Sean Liu, Shannon Roberts, Jeff Zabel, Bryan Reimer, Bruce Mehler, Stephanie Seneff, James Glass, and Joseph Coughlin. City Browser: Developing a conversational automotive HMI. In *Proc. CHI*, Boston, Massachusetts, April 2009.

[20] Alexander Gruenstein and Stephanie Seneff. Releasing a multimodal dialogue system into the wild: User support mechanisms. In *Proceedings of the 8th SIGdial Workshop on Discourse and Dialogue*, pages 111–119, Antwerp, Belgium, September 2007.

[21] Alexander Gruenstein, Stephanie Seneff, and Chao Wang. Scalable and portable web-based multimodal dialogue interaction with geographical databases. In *Proceedings of the Ninth International Conference on Spoken Language Processing (INTERSPEECH 2006 - ICSLP)*, Pittsburgh, Pennsylvania, September 2006.

[22] William I. Hallahan. DECtalk software: Text-to-speech technology and implementation. *Digital Technical Journal*, 7(4):5–19, 1995.

[23] Ryuichiro Higashinaka, Noboru Miyazaki, Mikio Nakano, and Kiyoaki Aikawa. A method for evalutating incremental utterance understanding in spoken dialogue systems. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, Colorado, September 2002.

[24] Michael Johnston and Srinivas Bangalore. MATCHKiosk: A multimodal interactive city guide. In *Proceedings of the Association for Computational Linguistics (ACL-2004) Interactive Posters/Demonstrations Session*, pages 222–225, Barcelona, Spain, July 2004.

[25] Michael Johnston, Srinivas Bangalore, Gunaranjan Vasireddy, Amanda Stent, Patrick Ehlen, Marilyn Walker, Steve Whittaker, and Preetam Maloor. MATCH: An architecture for multimodal dialogue systems. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 376–383, Philadelphia, Pennsylvania, July 2002.

[26] Raymond Lau, Giovanni Flammia, Christine Pao, and Victor Zue. WebGALAXY – Integrating spoken language and hypertext navigation. In *Proceedings of EUROSPEECH-1997*, pages 883–886, Rhodes, Greece, September 1997.

[27] Steven C. Lee and James R. Glass. Real-time probabilistic segmentation for segment-based speech recognition. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, Sydney, Australia, November 1998.

[28] Ian McGraw and Stephanie Seneff. Speech-enabled card games for language learners. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, Chicago, Illinois, July 2008.

[29] Noboru Miyazaki, Mikio Nakano, and Kiyoaki Aikawa. Spoken dialogue understanding using an incremental speech understanding method. In *Systems and Computers in Japan*, volume 36. 2005.

[30] Mikio Nakano, Noboru Miyazaki, Jun-ichi Hirasawa, Kohji Dohsaka, and Takeshi Kawabata. Understanding unsegmented user utterances in real-time spoken dialogue systems. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 200–207, College Park, Maryland, June 1999.

[31] Tim Paek, Yun-Cheng Ju, and Christopher Meek. People Watcher: a game for eliciting human-transcribed data for automated directory assistance. In *Proc. INTERSPEECH*, Antwerp, Belgium, September 2007.

[32] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, October 2005.

[33] Stephen J. Pueblo. Videorealistic facial animation for speech-based interfaces. M.Eng. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2009.

[34] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

[35] Stephanie Seneff. TINA: A natural language system for spoken language applications. In *Computational Linguistics*, volume 18, pages 61–86. 1992.

[36] Stephanie Seneff. Response planning and generation in the MERCURY flight reservation system. In *Computer Speech and Language*, volume 16, pages 283–312. 2002.

[37] Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. GALAXY-II: A reference architecture for conversational system development. In *Proceedings of International Conference on Spoken Language Processing (ICSLP 1998)*, Sydney, Australia, November 1998.

[38] Stephanie Seneff and Joseph Polifroni. Dialogue managment in the MERCURY flight reservation system. In *Proceedings of the Applied Natural Language Processing / North American Association for Computational Linguistics Conference (ANLP-NAACL)*, Seattle, Washington, United States, April 2000.

[39] Stephanie Seneff and Joseph Polifroni. Formal and natural language generation in the MERCURY conversational system. In *Proceedings of International Conference on Spoken Language Processing (ICSLP 2000)*, Beijing, China, October 2000.

[40] Michael K. Tanenhaus, Michael J. Spivey-Knowlton, Kathleen M. Eberhard, and Julie C. Sedivy. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268(5217):1632–1634, 1995.

[41] Jon R. Yi. Natural-sounding speech synthesis using variable-length units. M.Eng. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1998.

[42] Victor Yngve. On getting a word in edgewise. In *Papers from the Sixth Regional Meeting of the Chichago Linguistic Society*, pages 567–577, 1970.

[43] Brandon Yoshimoto. Rainbow Rummy: A web-based game for vocabulary acquisition using computer-directed speech. M.Eng. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2009.

[44] Richard F. Young and Jina Lee. Identifying units in interaction: Reactive tokens in Korean and English conversations. *Journal of Sociolinguistics*, 2004.

[45] Victor Zue, James Glass, David Goodine, Hong Leung, Michael Phillips, Joseph Polifroni, and Stephanie Seneff. Recent progress on the SUMMIT system. In *Proceedings of Third DARPA Speech and Natural Language Workshop*, pages 24–27, Hidden Valley, Pennsylvania, June 1990.

[46] Victor Zue, James Glass, Michael Phillips, and Stephanie Seneff. The MIT SUMMIT speech recognition system: A progress report. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 21–23, Philadelphia, Pennsylvania, 1989.

[47] Victor Zue, Stephanie Seneff, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and Lee Hetherington. Jupiter: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1), January 2000.