A Framework for Multi-Modal Input in a Pervasive Computing Environment

by

Shalini Agarwal

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

May 2002

Author . . . .
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by . . . .
Larry Rudolph
Principle Research Scientist
Thesis Supervisor

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

A Framework for Multi-Modal Input in a Pervasive Computing Environment

by

Shalini Agarwal

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we propose a framework that uses multiple-domains and multi-modal techniques to disambiguate a variety of natural human input modes. This system is based on the input needs of pervasive computing users. The work extends the Galaxy architecture developed by the Spoken Language Systems group at MIT. Just as speech recognition disambiguates an input wave form by using a grammar to find the best matching phrase, we use the same mechanism to disambiguate other input forms, T9 in particular. A skeleton version of the framework was implemented to show this framework is possible and to explore some of the issues that might arise. The system currently works for both T9 and Speech modes. The framework also includes potential for any other type of input for which a recognizer can be built such as graffiti input.

Thesis Supervisor: Larry Rudolph
Title: Principle Research Scientist

Acknowledgments

There are many people who made this thesis possible. This section can only try to do them justice.

Foremost, are the people who made this thesis happen. Larry Rudolph, my thesis advisor was always available and ready with creative ideas and encouragement. He somehow manages to care for both his students' academic well-being as well as their personal well-being. The Spoken Language Systems group (SLS) was instrumental in helping me understand the architecture and work out any problems that arose over the course of the year. I would particularly like to thank Scott Cyphers as well as the rest of the SLS team, Jim Glass, TJ Hazen, Lee Hetherington, and Eugene Weinstein, for their help throughout my design and implementation process. Also, Greg Shomo provided the technical support for this project.

Throughout the past year, I received inspiration and guidance from my officemates and co-workers as well. They are the ones who kept me going this year. My lab companions, Todd Amicon, Amay Champaneria, Josh Jacobs, and Jorge Ortiz, made all the difference and helped me to keep sane through the long days (and nights). I'd also like to thank Jason Dettbarn, Vincent Lee, Orlando Leon, Leon Liu, Brendon Kao, and Tom Kotwal. Ed Suh and Prabhat Jain added the PhD influence, always ready to answer yet another latex or academic world related question.

My friends outside of lab have also been a huge part of helping me through my years at MIT. My study-buddies, b4party, the soccer and lacrosse teams, and all the rest of you have made this an amazing place. I will not mention all of you here, but you know who you are.

Last but certainly not least, I could not have done any of this if without my family. I would like to thank them for all their support over the years. If it were not for them, I would not have made it to or through MIT.

This research was supported and funded by the Oxygen affiliates: Acer, Delta Electronics, Hewlett-Packard, Nippon Telegraph and Telephone, Nokia Research Center, and Philips Research as well as by DARPA through the Office of Naval Research contract number N66001-99-2-891702. Much of the iPAQ related applications were developed and supported by Compaq.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As ubiquitous computing becomes increasingly popular, natural input methods compatible with mobile devices become even more important. As devices become smaller, an accurate input disambiguation and understanding system for a large variety of input modes is ideal. In this paper, we describe a single-platform framework that disambiguates input by introducing a multiple domain, multi-modal input system for pervasive computing systems.

1.1 Oxygen

Project Oxygen [7] is an attempt to address the need for human-centric computing and connectivity. It is an effort that we have been working on over the past year and a half to create a system which connects all controllable devices seamlessly and dynamically. It allows people to use and control them via a hand-held computer, e.g. a Handy21. In time, Oxygen will allow people to overcome the need for desktop computers; creating a web of connected electronic devices accessible from anywhere via the Handy21. Users will not have to remember cryptic commands. They will be able to speak normally in everyday language into their Handy21 and make other simple commands by writing on their Handy21 screen. Their speech and script will be processed and understood. Their commands will be executed no matter how they choose to input them.

While the idea of Oxygen is simple and futuristic, the design and development of the system reveals the need for careful crafting and piecing together of different components. These components include speech analysis and understanding, a secure universal file system, an intentional naming system used to locate different devices (INS), a communications oriented routing environment (CORE) to maintain connections between devices, and many other pieces. We have been working closely with many other groups throughout the Laboratory of Computer Science (LCS) here at MIT to begin to integrate some of the major projects to create one larger system. Over the course of the year, the design of the input system, CORE, and the use of other components has changed as we implement and try different approaches. The driving force has always been ease of use and flexibility of the system.

1.2 Motivation

When humans communicate with each other, they have multiple sources of input and are aware of a context in which to place input they receive. For example, if a pedestrian is explaining directions to a driver in Cambridge, the pedestrian normally uses a combination of speech and gestures to tell the driver where to go. Also, both dialog participants know the context is driving directions and geographical landmarks in Cambridge. Therefore, when the pedestrian uses a street name such as "Main Street", both participants are aware that the "Main Street" being referred to is the one in Cambridge, not another "Main Street" in a different city such as Seattle. This background information reduces the chance that the driver will misunderstand the directions. Human-computer communication can also be improved with this context and multiple-input-mode information.

Since speech is an example of natural communication, we use speech input as an example in this section to motivate this thesis.

1.2.1 The Problem: False-Positives

Speech is a comfortable communication method for humans. Unfortunately, even with all of the speech technology we have today, speech recognition is not perfect. There are many speaker-dependent, unrestricted-domain and speaker-independent, restricted-domain speech recognition systems out there. However, it is very difficult to build a speaker-independent, unrestricted-domain speech recognition system that will recognize and understand everything a person might say. The combination of variation involved in the human voice and the sheer magnitude of sentences and phrases in any given language forms an enormous challenge for flexible speech input [11]. That is why most speech input systems in use today tend to be more focused and domain-dependent.

In order to implement our system for other input disambiguation, we decided to extend the Galaxy system developed by the Spoken Language Systems (SLS) group at MIT [22]. To use the Galaxy system, a user creates a context-based domain of phrases and concepts that he wants the system to recognize using the SpeechBuilder application [8]. Although SpeechBuilder is originally intended for speech, this domain is a language model on which to base all input. It is similar to the directions context in the pedestrian-driver example above. Another example is the creation of a domain for controlling the slides during a presentation. In this scenario, a user would enter phrases and concepts for jumping between slides, playing animations, starting/stopping, direction of control, et cetera.

The idea of a domain works well when the user has a manageable number of well-defined commands. However, as the scope grows, the number of commands to be recognized also increases. By induction, one can imagine an infinitely large domain that is essentially the world of all possible phrases, a very difficult problem. Even with sophisticated models for confidence scoring, recognition is not perfect[10]. As our presentation manager domain began to include more than a few simple commands, we started to experience many more false positives using speech input. Sometimes even simple things like “go to the previous slide” was mistaken for “go to the next slide.”

1.2.2 Coming Up With a Solution

Check Input Before Execution

One simple solution to this problem of false positives is a check before executing a command. For example, the system could return a list of three commands it thinks the user might have said and have the user confirm one of these commands. There are many instances in which a human-computer dialogue with these error checks would be perfectly acceptable. An automated help system over the phone is a good example. It is a one-on-one conversation between the user and the computer over the phone in which the user can easily press a button to select a choice. However, we are creating an oxygen system and our main focus is creating a system that is both accurate and natural for a human to use. When a presenter gives a command, during a presentation, it should be understood correctly and executed without the computer coming back and asking questions.

We have established the need for a “smarter” system that is better equipped to correctly understand input and execute a command without additional user input. Our first idea for implementing this “smarter” system was slide-tracking.

Slide-Tracking

The main idea behind slide-tracking is to track a presenter’s progress along a slide. Such information would help us to better understand what kinds of commands are more likely to be requested next and thereby reduce the number of false-positives experienced. For example, suppose a speaker is introducing background material about the next slide. He might say “On the next slide..” not intending to change to the next slide. With slide tracking, we could disable the “next slide” command until some specified percentage of the slide has been covered. Other types of commands may also be more probable at certain stages of a presentation than others.

As we began to think about implementing slide-tracking, we came up with some interesting issues. The main issue was the prevalence of figures and pictures in presentations. Sometimes, the best way to illustrate a point is to show a picture of it and then talk about it. However, at first glance, it seems difficult to track progress along a picture. Also, there is no guarantee that the presenter will choose to say the same words written on the slide. Therefore, in order to get accurate progress reported from a slide tracking mechanism, one must be very careful. Disabling or even decreasing the probability of certain commands based on slide-tracking information alone could be detrimental.

As we thought about the problem of false-positives in relation to ambiguous inputs such as speech, we began to think of slide-tracking as one powerful input to a larger system. The solution we present in this paper is multiple context domains with multi-modal input.

1.3 Multiple Domains

As we discussed earlier, the number of false-positives increases as the domain size increases. Following from this logic, if we can create a network of small domains that together function as a larger domain, we should be able to keep the number of false-positives to a minimum. Our claim is that this network of small domains can be in the form of multiple domains all running in parallel with a "smart" selection process to determine which domain the input was meant for. We discuss this idea further in Chapter 3.

1.4 Multi-Modal Input

Another way to reduce the number of false-positives of one input mode such as speech is to combine it with other modes of input. Since speech is not always the most convenient input mode, other modes of input are necessary for a human-centric pervasive system. It turns out that our multiple domain structure lends itself to multiple mode extensions. In this paper we concentrate on introducing T9 input both for ease of use and false-positive reduction. We develop a system in which speech is simply one of the possible input modes, a system that can even be used predominantly for an input mode such as graffiti. We discuss the multi-modal aspect of our input framework in Chapter 4.

1.5 Running Example: The Presentation Manager

The input structure we develop in this paper has an infinite number of applications. For the purpose of illustration, we will use the Presentation Manager as a running example throughout this paper. In this section, we outline some of the specifics of the Presentation Manager for background information.

One of the example scenarios the Oxygen Research Group (ORG) has been working on this past year is a Presentation Manager. The manager allows a person to give a presentation via a Handy21, thereby controlling anything he might need. This includes environment controls such as lights and speakers, computer controls such as laptops and projectors, application controls such as Powerpoint and slide managers, and anything else connected to an electronic device. With this presentation manager in mind, we can begin to develop the input framework of this thesis.

1.6 Guide to this Thesis

This thesis will focus on the input side of oxygen systems, using the Presentation Manager as an example. We will first discuss related research and background material necessary for understanding the building blocks for our system architecture in Chapter 2. We will then delve into the heart of this thesis in Chapter 3 and discuss the multiple domain structure. Chapter 4 will continue the second part of this thesis and introduce the implementation of other types of input to create a multi-modal system, focusing on T9.

The contributions of this thesis include proposing an overall framework for a pervasive computing input disambiguation system, exploring the ideas of multiple domain input and multi-modal input, and extending existing architectures to begin implementing this system. The basic functionality of this system has been implemented and some preliminary results have been investigated. These results and their analysis are discussed in Chapter 5.

Chapter 2

Related Work and Background

There has been a great deal of research interest in the reduction of speech recognition error through frameworks utilizing multiple modes of input. Most of these frameworks are focused on a smaller context domain set just as we break down our context into smaller domains. The majority of them were also predominantly concerned with the mutual disambiguation aspect of multi-modal structures. In this section we discuss work done in multiple domains and multi-modal systems applicable to this thesis. We also introduce some of the building blocks necessary to create this system.

2.1 Multiple Domains

Splitting context into multiple domains is a common approach to improving recognition accuracy. Hsin-min Wang and Berlin Chen researched the area of spoken document retrieval[16]. They found that for their specialized task, recognition errors were greatly reduced with a content-based language model. Although their study was based on the Mandarin language that has a different set of language issues, the findings on a constrained language model are still helpful for this research. They found that whether the language model was constrained using actual transcriptions of the spoken language or the baseline language model, the recognition accuracy still improved.

Although speech recognition is improved through the breaking up of context into multiple domains, ease of use is still most important. It is essential that the user does not have to concern himself with switching between domains. Reginia Braga's research involves retrieving domain information from multiple domains when a user might not even be aware that he is looking for information from a different domain [4]. She suggests an architecture designed to access these domains and retrieve information via the Internet and through Java modules. Although this thesis is more focused on user-defined domains being run simultaneously using a consistent architecture and processing algorithm for all inputs, the aspect of seamless switching and integration of multiple domains is the same.

2.2 Multi-Modal Input

2.2.1 Choosing the Most Effective Input

It is a commonly perceived that using multi-modal input to reduce recognition error will actually increase the recognition error by compounding the errors from the different types of input. However, this is not true. With multiple input choices, the user will be able to choose the input mode that best suits the message being given [18]. For example, consider a flight information system similar to the Mercury system developed by SLS [23]. A user might choose to give the type of flight information he wants via speech and specify the city codes via a pen or touch-pad based input. This way, recognition error will be minimized since the speech input only needs to match one of a few models and and there is less error involved with the pen-based specific city code information. If the user had to speak the city code, the probability of recognition error might have been higher, and inputting the whole phrase using pen or touch-pad based input might have been too tedious.

Multi-modal input simplifies language complexity since input can be expressed in its most natural form [18]. For example, one gesture can replace a whole phrase of explanatory words. Pen based input can replace the need to repeat a misunderstood word and can allow for corrections to be made more easily.

Research done by Bernhard Suhm, Brad Myers, and Alex Waibel indicates that multi-modal error correction is faster and more accurate than attempting error correction through a unimodal speech input architecture [25]. This could be due both to the user choosing the most effective mode for correction and the correction capabilities of the speech interface used. In this thesis, we stress the importance of different input modes being better suited for different tasks. We also hope to utilize the different correction capabilities of different modes of input. For example, if a user attempts to speak a word that is misunderstood, picking a word out of the top five recognized choices might be more efficient than attempting to re-speak the word and have it misunderstood again.

2.2.2 Mutual Disambiguation

Humans use complementary visual and audio inputs simultaneously to understand human communication almost flawlessly. Simultaneously combining multiple modes of input such as speech and lip reading or speech and gestures drastically reduces the recognition error [3]. As Karen Mills and James Alty from Loughborough University point out, part of this reduction in error is due to redundancy of information captured through multiple input sources [15]. And, as Sharon Oviatt points out, part of this reduction is due to eliminating conflicting interpretations [20]. The latter type of error reduction might lead to unknown or ambiguous results, but these results might still be more accurate.

Quickset

A multi-modal input myth proposed by Oviatt is that multi-modal inputs always overlap temporally. She points out that in the case of speech and gesture inputs, only about 25% of speech input had references to things that had to be disambiguated by temporally overlapping gestures [18]. In her research, Oviatt cites Quickset [6], a study done by the Oregon Graduate Institute, to examine overall results for their multi-modal speech and pen-based architecture and for native versus non-native speakers. She mentions that the spoken language error rate was reduced overall by an impressive 41% with the multi-modal architecture [17, 20]. Although this seems higher than one might expect and could be due to the high number of non-native speakers, it does suggest that multi-modal architectures are useful and can be designed to help alleviate the frustration that speech recognition errors bring about.

The results were even more dramatic in other environments. Quickset research on portable speech and pen-based multi-modal architectures in mobile environments found that the spoken language errors were reduced by 19-35% through multi-modal disambiguation [19]. Some of this error reduction was probably due to the noisiness factor of the environment being eliminated through other silent modes of input. This is an important phenomenon for our research as well. Since our input structure is meant to be used through hand-held mobile devices, specifically the iPAQ, it is comforting to know others have reduced the speech recognition errors in mobile environments through multi-modal frameworks. The mutual disambiguation statistical techniques Quickset uses would be applicable to this research as well since Quickset also makes use of n-best lists of processed output.

Quickset and this thesis have a similar goal: to decrease input recognition error. The difference with our system lies in the extension of the Galaxy speech architecture, to create a single platform to run multiple domains in addition to multiple input modes.

The Use of Finite State Machines

Another study done at AT&T Labs by Michael Johnston and Srinivas Bangalore focuses on the integration of speech and gesture mode information and its processing [12]. They suggest altering the speech language model based on information obtained from the gesture recognizer and describe a finite-state device to process both speech and gesture inputs. They have come up with a specification for translating gestures into gesture symbols that can be understood by a finite-state machine [1].

Their work is similar to our research in that we also use finite-state machine processing, but for individual inputs. Their gesture symbol specification might prove useful in our implementation of disambiguating input, though that is ongoing work at this time. They discuss the use a finite state machine to encompass the entire structure. We will use finite-state machines for more isolated functions. This may allow for added flexibility in utilizing other types of state information to select the correct input.

2.3 A Simple Structure: One Domain

In order to develop a multiple-domain system, we must first understand how a single-domain system works. In this section we describe the basic building block for our overall structure, the Galaxy system.

A domain can be created using the SpeechBuilder tool as mentioned in Section 1.2.1. In order to use the domain, it must be compiled and built in conjunction with the Galaxy system built by SLS. The Galaxy system consists of a hub connecting many different components as shown in Figure 2-1. This figure was taken from an SLS group presentation.

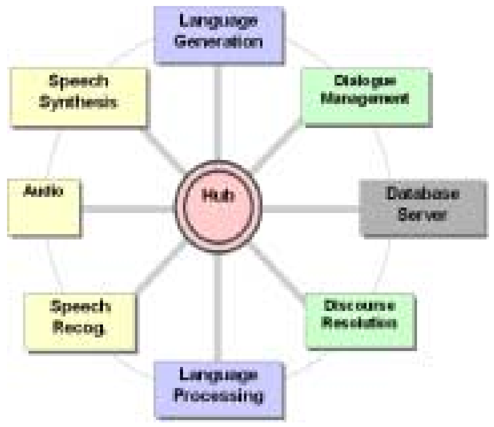
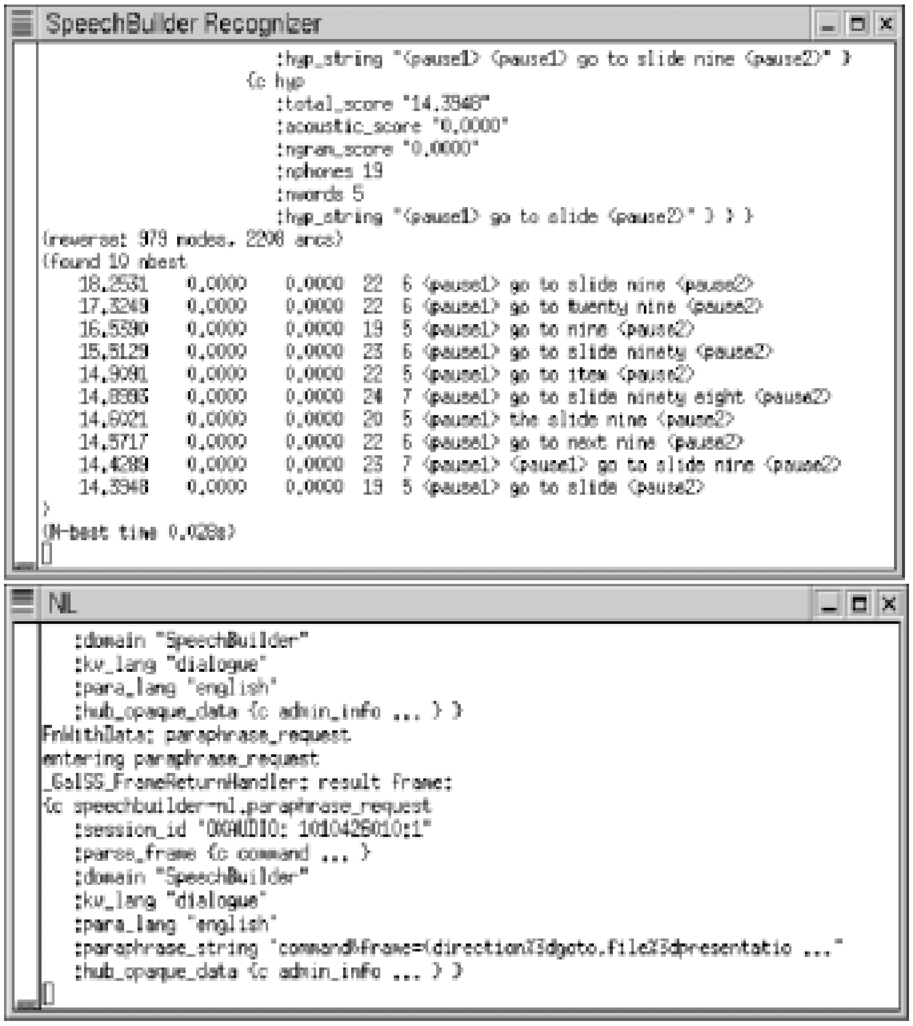


Figure 2-1: Galaxy architecture

The two Galaxy components that we are most concerned with are the Speech Recognizer and the Language Processing unit. When a user runs a domain and issues a command, the command is captured by the Audio Server and saved as a wave file. The wave file is sent from the Audio Server to the Speech Recognizer.



```
                    :hyp_string "<pause1> <pause1> go to slide nine <pause2>" }
                  {c hyp
                    :total_score "14.3948"
                    :acoustic_score "0.0000"
                    :ngram_score "0.0000"
                    :nphones 19
                    :nwords 5
                    :hyp_string "<pause1> go to slide <pause2>" } } }
(reverse: 979 nodes, 2208 arcs)
(found 10 nbest
    18.2531    0.0000    0.0000  22  6 <pause1> go to slide nine <pause2>
    17.3249    0.0000    0.0000  22  6 <pause1> go to twenty nine <pause2>
    16.5390    0.0000    0.0000  19  5 <pause1> go to nine <pause2>
    15.5129    0.0000    0.0000  23  6 <pause1> go to slide ninety <pause2>
    14.9091    0.0000    0.0000  22  5 <pause1> go to item <pause2>
    14.8985    0.0000    0.0000  24  7 <pause1> go to slide ninety eight <pause2>
    14.6021    0.0000    0.0000  20  5 <pause1> the slide nine <pause2>
    14.5717    0.0000    0.0000  22  6 <pause1> go to next nine <pause2>
    14.4289    0.0000    0.0000  23  7 <pause1> <pause1> go to slide nine <pause2>
    14.3948    0.0000    0.0000  19  5 <pause1> go to slide <pause2>
)
(N-best time 0.028s)
```

```
   :domain "SpeechBuilder"
   :kv_lang "dialogue"
   :para_lang "english"
   :hub_opaque_data {c admin_info ... } }
FnWithData: paraphrase_request
entering paraphrase_request
_GalSS_FrameReturnHandler: result frame:
{c speechbuilder-nl.paraphrase_request
   :session_id "OKAUDIO: 1010426010:1"
   :parse_frame {c command ... }
   :domain "SpeechBuilder"
   :kv_lang "dialogue"
   :para_lang "english"
   :paraphrase_string "command%frame=(direction%3dgoto.file%3dpresentatio ..."
   :hub_opaque_data {c admin_info ... } }
```

Figure 2-2: Sample output from the Speech Recognizer and the Language Processing unit of Galaxy.

The Speech Recognizer comes up with an n-best list of phrases. These phrases best match the speech input based the words in the user-defined domain. This n-best input interpretation list is then sent to the Language Processing unit. The language processor selects one input based on information derived from the user-defined domain and language models. Eventually, Galaxy packages up the results in the form of a frame, and sends the frame to a user-defined back-end server. The server is responsible for parsing the frame and executing the command. Figure 2-2 shows sample output of the n-best list and the domain output from the Speech Recognizer and Language

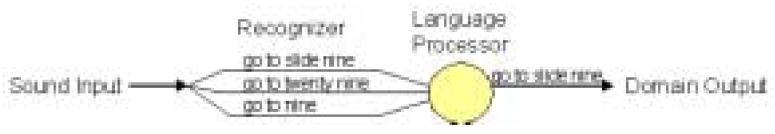


Figure 2-3: Basic structure for implementing one domain.

Processing unit. A stripped-down, simplified view of data flow from speech input to domain output is illustrated in Figure 2-3.

Chapter 3

Multiple Domains

Multiple domains is a common approach to the problem of running numerous jobs simultaneously. SLS has also created a system in which you can switch between different domains to obtain information on weather, flights, and directory information. However, they are not running these domains for simultaneous use.

As mentioned in the previous chapter, our decision to create multiple context domains stems from the need to reduce the number of false-positives as scope increases. By limiting the size of each domain to a single layer of commands or smaller concepts and running these domains simultaneously, we hope to improve the accuracy of each domain and thus the accuracy of the overall system.

3.1 Breakdown Into Domains

Let us begin with an everyday example of multiple domains. Imagine Bob, the President of a firm, walking into a conference room to run a morning meeting with the Vice-Presidents. The first thing Bob says is "Let's start out with some coffee." Tom, the secretary, steps out of the room to get it. Next, Bob asks "What were the sales figures for this year so far?" Immediately, Lisa, the VP of sales, responds with the figures.

The key aspect of this interaction is that Bob did not have to say "Tom, please get the coffee" or "Lisa, what are this year's sales figures?" The person being addressed was implicit in the context of the question. Each person in the meeting has a different specialty and understands different commands and questions. Similarly, by running multiple domains simultaneously, each with a different context, the domain being invoked should be implicit in the context of the command.

In the Presentation Manager scenario, each domain would represent a different layer of commands that a user might give. Our current breakdown consists of three different layers:

- **Slide Layer Domain:** This domain contains commands to navigate around a slide and is intended for a user in the middle of presenting a slide. Example commands include "play sound" and "show animation".

- **Presentation Layer Domain:** This domain contains commands to navigate through a presentation. Example commands include "skip to conclusion", "next slide", and "go back".

- **Application Layer Domain:** This domain contains commands to control different applications and a computer in general. Example commands include "start the oxygen presentation", "switch to email", "open Netscape", and "kill the browser".

It is important to note that the layers and commands described above are only suggestive. A powerful aspect of this system is that every domain concept and command description is completely user defined. A user can choose to map any word to mean any other word and can choose to build a domain for any context, not just the Presentation Manager ones we are describing here.

By running these three domains simultaneously, we can allow each domain to listen to the commands being issued, decide whether it understands what was said, and send its interpretation to an output server. The output server can then decide what to do with the results. If speech recognition were perfect and the domains did not have any overlapping commands, only one domain would return an answer. The others would return "unknown". However, the motivation for multiple domains is the inaccuracy of speech recognition as the number of commands increases. Although the command will not match all the domains, it is likely to match more than one domain. The server then selects which domain it thinks the user intended to invoke based on scoring and state information. An example of such state information is slide tracking, a potentially useful input to the selection process as discussed further in Section 3.2.2.

Another advantage to multiple domains is the modular design. Good software engineering practices stress the importance of flexibility of code via modular designs. Suppose a user is running the three domains described above, but then decides to end his presentation and move his laptop to his office. When he moves to his office, he would like to start the office control environment domain. This domain would define commands to allow him to control the lights, the windows, the air conditioning, and so on. However, he still has his laptop running and needs to be able to access the Application Domain outlined above. He no longer needs the Slide Layer and Presentation Layer domains since he is done with his presentation. With one large Presentation Manager domain, it would not be possible to turn off a set of previously-useful commands. However, with this modular approach of multiple domains, he can easily kill the Slide and Presentation Layers and start up his Office Environment domain.

Instead of creating many overlapping domains, a user will be able to create domains based on a simple concept and choose any subset of these domains to run simultaneously at any given time. Depending on the domains running, the command “play sound” can mean different things. If the Slide Layer domain is running, “play sound” could mean “play the next sound clip on the current slide.” If an office environment is running, “play sound” could mean “start up Winamp on my computer and play my favorite song.” With one large domain, it would be much more difficult to implement these different contextual meanings.

Let us now consider the implementation of multiple domains. In the previous chapter, we laid the framework for a single domain. Now that we understand how a single domain works and the need for multiple domains, we can extend the system.

3.2 Extending the Structure: Multiple Domains

The structure for a single domain as discussed in Section 2.3, can be extended to multiple domains relatively easily.

3.2.1 Architecture

There are two methods of extension that we considered. The first involves running all domains simultaneously with a single Galaxy system. The second is a more modular system with one Galaxy system for each domain.

One Galaxy

The SLS group has been developing different methods of facilitating multiple domains within a single Galaxy architecture. One of these methods uses a single, domain-independent recognizer [14]. However, this approach requires regularized language models and results in slightly degraded recognition accuracy. Another multiple domain method they have been working on involves a two stage recognition model: a domain-independent recognition engine and a domain tailored knowledge constraint back-end [5].

With these setups, we could probably also achieve simultaneous domains with only one Galaxy system. However, we are concerned with flexibility and fault isolation in the system. For example, a user might want to completely switch modes from presentation to office environment control. In this case, we would want the office domain to start in parallel without disturbing the presentation manager related domains already running. Although this might be possible with the setup SLS is developing, we would prefer a more modular system.

Multiple Galaxy's

The method of extension we have decided upon involves running multiple instances of the Galaxy system in parallel, each running a single domain. In this way, we can create a system in which instances of domains can be introduced and destroyed without affecting other domains already running. We hope this type of setup will result in better reliability and allow for the sharing of computing responsibility among many processors, not just one.

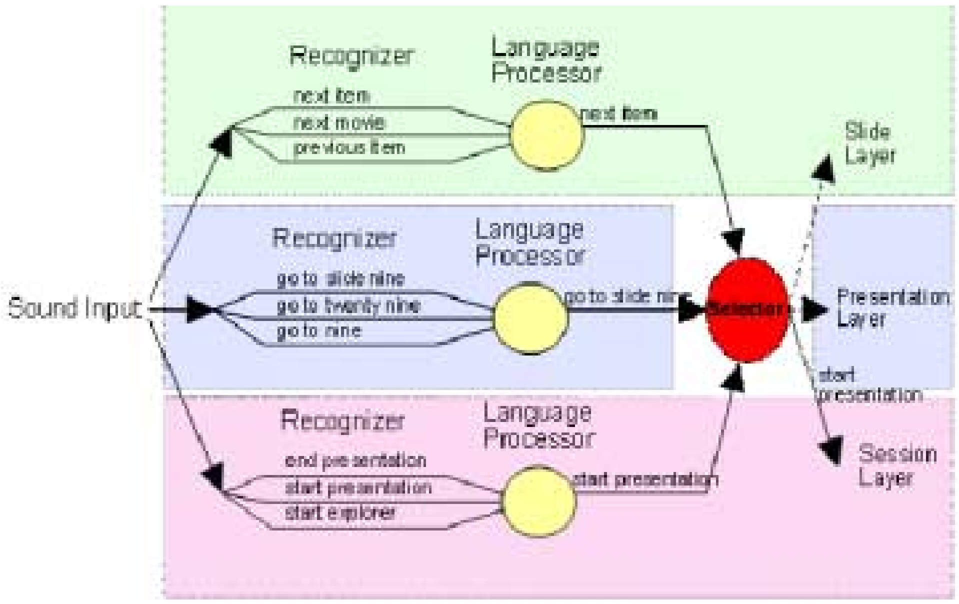


Figure 3-1: Overall system structure for facilitating multiple domains.

The structural details of multiple galaxy's are very similar to those of the single domain setup in Figure 2-3. By modifying the audio server or creating an additional server, it is theoretically possible to broadcast the speech input wave files to multiple Galaxy instances. Each Galaxy instance uses its own domain-dependent Speech Recognizer and Language Processing unit to process the wave file and come up with an input interpretation for its domain. After each Galaxy system has come up with its own interpretation, a Selector selects the domain for which we think the input was intended to invoke. The Selector collects the interpretations from all the domains and chooses which one it thinks the user intended. As mentioned in Section 1.2.1, the original motivation for building this system was to reduce the number of false-positives for a given input mode. The first step we are taking to achieve this goal is to break up commands by context into smaller domains. The second step will be the use of the Selector. We must be careful in the implementation of the Selector to avoid introducing additional false-positives into the system. If the wrong domain's output is chosen even if the speech was understood correctly, we will be back where we began. Figure 3-1 shows the overall design of the system structure.

3.2.2 The Selector

Currently, we think the Selector should have at least two inputs. The first, and hopefully most useful, will be the score attributed to the input interpretation by each Galaxy system. When the Speech Recognizer comes up with the n-best list of interpretations, each possibility on the list has attached to it a score of how well the interpretation matched the input [21]. When the Language Processing unit outputs the final interpretation, it also outputs a scored list. Unfortunately, these scores are not currently definitive scores, they are ordinal in nature. The scores are therefore useful for ranking possible interpretations for each input within a domain, but perhaps not useful for comparing final interpretations from different domains. Fortunately, though this feature is not currently available, it is currently possible to generate other more confident scores using Galaxy. When these definitive scores do become available for public domains, they will be a useful tool for the Selector.

Another input to the Selector could be slide tracking information. As mentioned in Section 1.2.1, slide tracking alone is not enough to select between all commands. However, it may be helpful in determining which domain the user intended to invoke at a particular time. With this structure, slide tracking could have a influence in the Selector without having the system depend on it heavily. Other state information could also prove useful input to the Selector as the complexity of the system grows and more ideas are incorporated. There has been some research done on expectations in spoken dialog by Ronnie Smith and Howard Hipp [24]. They propose that the dialog structure mirrors that of the task at hand. Therefore, if the Selector was trained on certain type of tasks and could recognize dialog pertaining to those common tasks, it could keep track of state information pertaining to the task and be able to expect certain follow-up input commands. This information might also have to be encoded into the back-end application that actually parses and executes the commands. With proper and careful input, the Selector can be a powerful step in decreasing the number of false-positives experienced.

Chapter 4

Multi-Modal Input

The most exciting aspect of the multiple domain system is the flexible and simple structure. This simple structure is extendable to different types of input; the main focus of this thesis.

4.1 The Need for Multiple Input Modes

As mentioned when we started out discussing this system in Section 1.1, the main focus is on human-centric computing, making the system easy and natural to use. Since speech is one of the most natural human-to-human communication methods, we began our discussion with the facilitation of speech input. However, not only is speech recognition not always accurate, but also speech is not always the preferred method of communication with a computer. Speech is transient in nature. When a user is speaking, he cannot see his command as he can with pen-based inputs. Speech is not yet globally accepted as a natural input to a computer. People are simply not used to it. Also, speech is not private [9]. The usefulness of speech input really depends at the task at hand. Suppose a person wants to send a command to an oxygen system during a lecture. Even though it is easier to make his command via speech, it is not acceptable to speak during a lecture. Therefore, he must find an alternative way of executing his command.

There are many natural modes of input. When humans communicate with each other, they often simultaneously use speech and spatial communication methods, such as gestures, to illustrate a point [13]. These input modes complement each other to help reduce the number of recognition errors. There are also other types of communication modes, both human-human and human-computer. These include pen-based modes such as graffiti and handwriting and touch-pad modes such as telephone keypads and palm pilot based navigation. Eventually, all of these input modes would be useful to the oxygen system we are developing since the user should be able to input information in the most effective and comfortable mode. However, for this framework, we chose to focus on the T9 mode.

4.2 Additional Input: T9

Until recently, the most common way to send a command to a computer was via a keyboard. With the advent of mobile devices, people are getting accustomed to entering data into their palm pilots via graffiti and character recognition systems. When using a cell phone, the only way to enter text data is via the number keypad, a technology known as T9. T9 is a text input method defined by the number-to-letter mapping found on each key of a telephone pad. For example, the number "2" maps to "a", "b", or "c". Basically, T9 enables faster text input with a fewer number of available input buttons.

Although T9 is somewhat less convenient than entering text data via a keyboard in stationary situations, it is more convenient with small mobile devices. Therefore, it is important that we be able to integrate this increasingly popular mode of input into our oxygen system, allowing the user flexibility to choose his preferred input mode at any given time.

For this thesis, we chose to implement T9 first to prove that it is possible to utilize the Galaxy components for different inputs. From the T9 implementation, it is obvious that graffiti and other input modes can easily be implemented.

4.3 Facilitating Multiple Inputs

As we add this multiple-input feature to the system structure described in Section 3.2, it is important that we remember the human-centric focus of this project. Switching between multiple input modes should be seamless; a user should be able to use any mode of input he desires without specifying the mode first. Also, given the same input, regardless of input mode, the output should be the same. By using the same framework for all input modes, the latter requirement is easier to meet.

4.3.1 Building on the Speech Implementation

First, we consider the implementation of a single domain with T9 input. It is very similar to the structure of a single domain with speech input illustrated in Figure 2-3. With speech input, the waveform is sent from the Audio Server to the Speech Recognizer for translation into an n-best list of guesses. These guesses are sent to the Language Processing unit for final interpretation based on the language model of the domain. Finally, the interpretation is sent to an output server for execution. The only speech dependent parts of this process are the Audio Server and the Speech Recognizer. Therefore, by building a new server and recognizer for T9, we can introduce this input mode using the same natural language processors developed by the speech group and Selector output method described in Section 3.2.2. Figure 4-1 illustrates a new Galaxy hub for T9 and graffiti input modes. We show graffiti here as well to illustrate that the modifications to implement T9 are the same as for implementing any other input mode.

For T9, the Audio Server must be replaced with a T9 server connected to the Galaxy hub. The T9 numerical data sequence is then sent via the hub to the T9 Recognizer for processing. Similarly, for graffiti input, the Graffiti Server would be connected to the Galaxy Hub to allow for graffiti input text to be sent to the Graffiti Recognizer.

The T9 server implementation we have built, allows a user to enter text on an iPAQ via buttons representing a telephone keypad. Figure 4-2 shows a picture of the

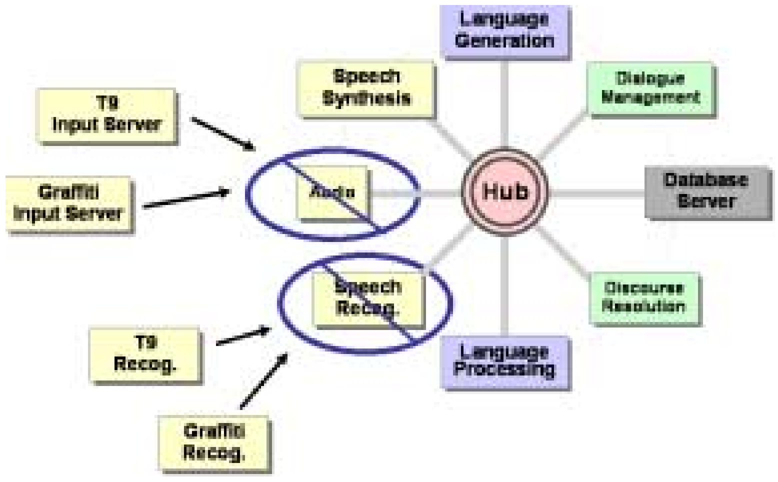
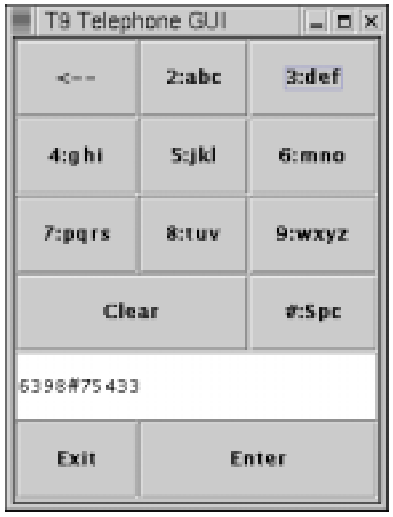


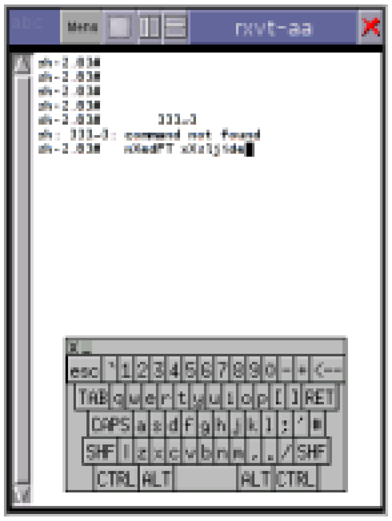
Figure 4-1: Modifying the Galaxy Hub system for new input modes.

T9 GUI on the iPAQ. This GUI shows that T9 can be used for not only input via a telephone interface, but also a virtual telephone keypad on an iPAQ. This T9 virtual keypad is faster and easier to use than a full keyboard picture on the screen because of the larger size of the buttons. Figure 4-2 shows the comparison between the two interfaces. The T9 GUI sends the numerical sequence to the T9 Server, which is connected to the Galaxy hub, to allow the numerical sequences to be sent to the T9 Recognizer.

4.3.2 The T9 Recognizer

Building the recognizers for each different input mode is probably the most involved part of this project. Through discussions with the speech group about implementing a T9 recognizer, it became clear that a generic programmable symbol recognizer would be most useful to both groups. Therefore, they implemented this symbol recognizer combining many of the tools that already existed within the Galaxy infrastructure. This symbol recognizer is programmable through building a series of finite-state transducers.




Menu
rxvt-aa
esc ` 1 2 3 4 5 6 7 8 9 0 - = <--
TAB q w e r t y u i o p [ ] RET
CAPS a s d f g h j k l ; ' \
SHF z x c v b n m , . / SHF
CTRL ALT ALT CTRL

```
sh-2.03#
sh-2.03#
sh-2.03#
sh-2.03#
sh-2.03#        333-3
sh: 333-3: command not found
sh-2.03#    [illegible]
```

(a) T9 GUI (b) Virtual Keyboard

Figure 4-2: T9 GUI on the iPAQ compared with a regular virtual keyboard on the iPAQ.

Programming the T9 Recognizer Through Finite State Transducers

Finite state transducers (FST's) are essentially a web of nodes representing states with directed transition arcs. These transition arcs have an associated input-output mapping and an optional weight. FST's can be used to implement probabilistic input recognition techniques. Commonly used for speech recognition [2], the constraint based path modeling capabilities of FST's make them appealing for building other types of recognizers as well. We describe how we make use of FST's in this section.

The T9 Recognizer uses FST's to map a series of inputs to outputs with associated penalties and scores. We worked with the speech group to understand FST methods to translate T9 number input to text input in ways similar to how they use them for speech.

To create the T9 FST's, we needed the words from the domain, the number to letter mapping, and any other mappings we wanted to implement such as mistype mappings. Once the FST's were defined, we were able to use a series of FST composition and FST conversion tools already built by SLS for FST manipulation to create a single, complete T9 FST. The T9 FST first corrects for mistyped numbers, insertions, and deletions by creating a mapping from numbers to different numbers, numbers to blanks, and blanks to numbers with different associated penalties. It then maps letters to words based on the domain lexicon.

Once the FST was done, we were able to run the T9 Recognizer. The recognizer is able to convert numerical sequences into an n-best list of phrases that match best given the possibility of mistypes and the domain-defined words.

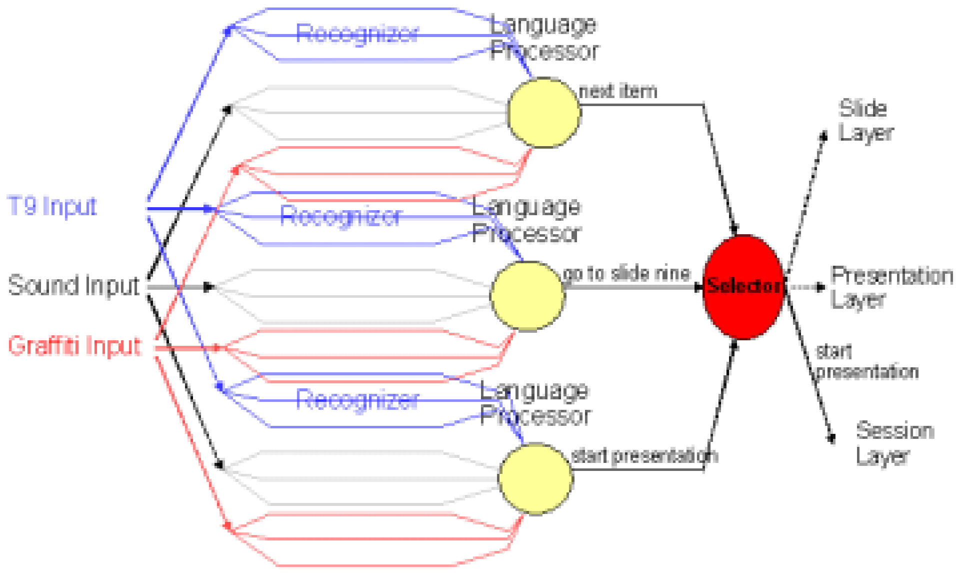


Figure 4-3: Input structure architecture with multiple domains and multiple input modes, highlighting T9.

Integrating the T9 Recognizer with the Galaxy System

Once the T9 Recognizer was built, we were able to integrate it with the Galaxy system via a modified hubscript [22]. Normally, when a user compiles a domain in SpeechBuilder, SpeechBuilder generates a hubscript. This hubscript essentially programs Galaxy with a path to route the input data for processing and output for distribution. In order to use the domain-defined language model and other pieces of Galaxy in a manner consistent with speech, we had to modify the hubscript. This hubscript takes the input in from the T9 Recognizer and sends it through the Language Processor and all other Galaxy channels similarly to how speech is processed. It was important to keep both speech and T9 functionalities embedded in the hubscript. This way, when audio can be streamed through the Frame Relay Server, another Galaxy component, the same hubscript can be used for both types of input. With this modified hubscript, we are able to understand T9 input using the same domain information and language models as with speech. Since this hubscript is consistent across different domains, any user can now create a domain, download it, copy over the new hubscript, and run the domain using T9 input.

4.3.3 Extensibility

With this generic symbol recognizer and the T9 Recognizer as an example, it would be simple to implement a Graffiti Recognizer. We would only have to create a series of FST's for mistypes with mappings and scoring based on graffiti intricacies. The hubscript could be also modified to watch for input from the Graffiti Recognizer. With training, the accuracy of detecting mistypes can also be increased.

By building the structure for understanding T9 input for a single domain, we can run multiple domains using the same structure as shown in Figure 3-1. We would simply need to broadcast the T9 input to all the T9 servers for each of the domain-dependent Galaxy instances running. Figure 4-3 shows the combined structure for running multiple domains for speech, graffiti, and T9 simultaneously.

With the introduction of multiple input modes, the implementation of the Selector becomes even more important. As mentioned in Section 2.2.2, one of the great advantages to multi-modal input is the use of mutual disambiguation. By processing the n-best lists from multiple simultaneous inputs together, mutual disambiguation techniques can be used to improve recognition accuracy.

Chapter 5

Conclusion

In this paper, we have proposed a framework for disambiguating different modes of input. The work of this thesis included the design of this framework, initial implementation of the system, and preliminary results on the input recognition accuracy of the system. In this chapter we describe the results of this thesis and give some suggestions for extensions and improvements.

5.1 Results

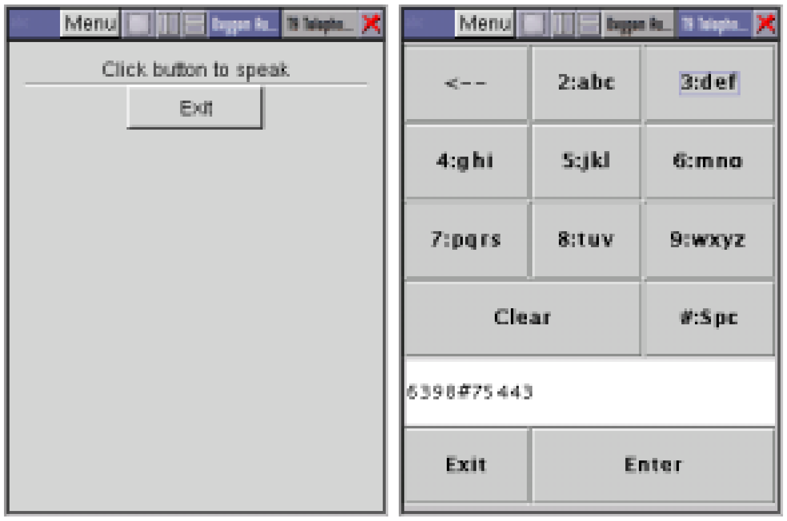
The result of this thesis is a single platform that can be used to disambiguate a large variety of input modes. One such platform is Galaxy. In this section we first explain the status of the current implementation of our system. Then, we explore statistics of established, finely-tuned domains. Finally, we show preliminary findings from an informal study run on our system and analyze the results.

5.1.1 Overview of Framework Implementation

The functionalities already in place include T9 recognition implemented to use the Galaxy infrastructure and speech recognition capabilities. Just as with speech domains, the T9 language models are created using the regular SpeechBuilder and Galaxy structure. A user can now build a domain in SpeechBuilder, download the domain, copy over the modified hubscript, and run both a T9 input version of the domain and a speech input version of the domain. The output of the T9 Recognizer and the Speech Recognizer will be processed using the same exact Natural Language Processing components and will be sent to the same back-end application. The input source is transparent to an outsider seeing only the actions resulting from execution of commands coming from the user.

The T9 Recognizer utilizes the generic symbol recognizer built by SLS. Any recognizer can be built in a symmetric fashion for any type of symbol based input. The additional input we are especially interested in at the moment is graffiti.

Figure 5-1 shows an iPAQ accepting both T9 commands and speech commands. Figure 5-2 shows the n-best results from both the T9 Recognizer and the Speech Recognizer for a Presentation Layer subdomain input "next slide". With both modes, the output from the Recognizers is sent to the Natural Language processing unit and so on through Galaxy and eventually the command "next slide" is sent on to the back-end application which controls a presentation and skips to the next slide.

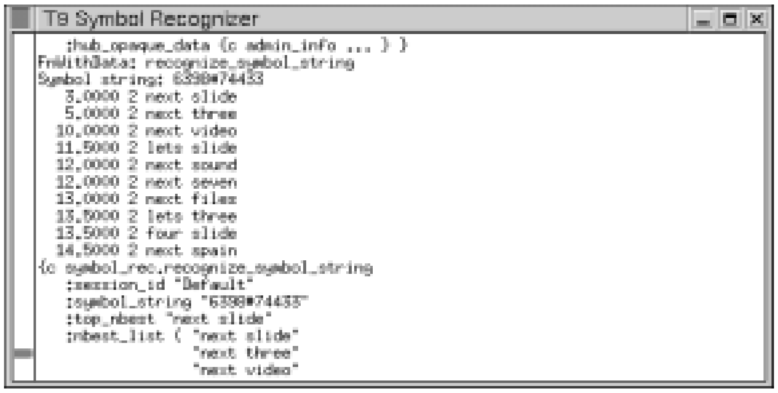


6398#75443

(a) Speech (b) T9

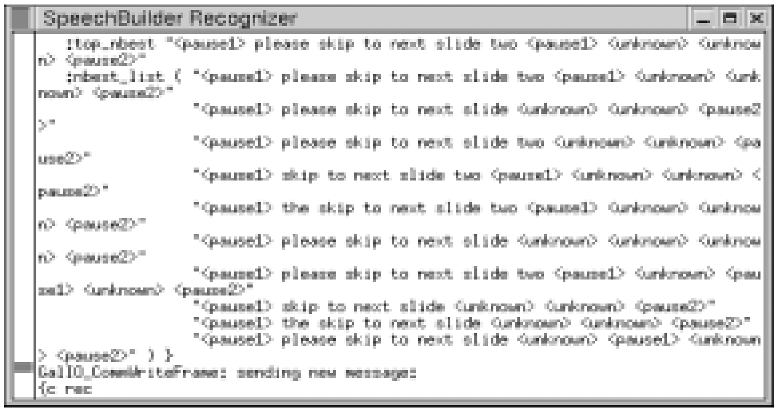
Figure 5-1: The T9 input GUI and the Speech input GUI running on the iPAQ. These are screen shots from the same iPAQ running both modes simultaneously.

The n-best list shown in Figure 5-2 for the T9 input contains the list of valid words



```
   :hub_opaque_data {c admin_info ... } }
FnWithData: recognize_symbol_string
Symbol string: 6398#74433
   3.0000 2 next slide
   5.0000 2 next three
  10.0000 2 next video
  11.5000 2 lets slide
  12.0000 2 next sound
  12.0000 2 next seven
  13.0000 2 next files
  13.5000 2 lets three
  13.5000 2 four slide
  14.5000 2 next spain
{c symbol_rec.recognize_symbol_string
   :session_id "Default"
   :symbol_string "6398#74433"
   :top_nbest "next slide"
   :nbest_list ( "next slide"
                 "next three"
                 "next video"
```

(a) T9 Recognizer



```
   :top_nbest "<pause1> please skip to next slide two <pause1> <unknown> <unknow
n> <pause2>"
   :nbest_list ( "<pause1> please skip to next slide two <pause1> <unknown> <unk
nown> <pause2>"
                 "<pause1> please skip to next slide <unknown> <unknown> <pause2
>"
                 "<pause1> please skip to next slide two <unknown> <unknown> <pa
use2>"
                 "<pause1> skip to next slide two <pause1> <unknown> <unknown> <
pause2>"
                 "<pause1> the skip to next slide two <pause1> <unknown> <unknow
n> <pause2>"
                 "<pause1> please skip to next slide <unknown> <unknown> <unknow
n> <pause2>"
                 "<pause1> please skip to next slide two <pause1> <unknown> <pau
se1> <unknown> <pause2>"
                 "<pause1> skip to next slide <unknown> <unknown> <pause2>"
                 "<pause1> the skip to next slide <unknown> <unknown> <pause2>"
                 "<pause1> please skip to next slide <unknown> <pause1> <unknown
> <pause2>" ) }
GalIO_CommWriteFrame: sending new message:
{c rec
```

(b) Speech Recognizer

Figure 5-2: T9 Recognizer output and Speech Recognizer output illustrating n-best lists generated for the input "next slide".

that best matched the number sequence "6398#75433" that was entered. Similarly, the n-best list for the Speech input contains the list of valid words that best matched one instance of the spoken input "next slide". It is the Natural Language Processing unit that is responsible for picking a valid phrase from the n-best list. Notice that the speech input was understood to be "please skip to the next slide two".

In testing the system, there were many cases where the selected output was not first on the n-best list. These cases are examples of the Recognizer suggesting valid words that best fit the input, and the Language Processing Unit selecting what it thought to be the best input that matched the language model of valid phrases. This case could turn out to be extremely useful when the Selector is implemented. It could help to disambiguate partial input from multiple modes even though no one input is complete.

5.1.2 Degradation Due to iPAQ Microphone

To establish a comparison for our speech accuracy, we must examine the accuracy of a large domain built and maintained by SLS themselves. For this comparative purpose, we chose to look at the Jupiter, a weather information system. Jupiter uses weather and geographic databases to answer queries such as "What cities do you know about in California?" and "What will the temperature be in Boston tomorrow?" SLS has been using Jupiter for a few years and has made it publicly accessible via a toll-free telephone number. As a result, they have logged over 10,000 calls which they have used for training speech recognition and language models. Jupiter makes use of a well-crafted domain designed by speech experts and has been finely tuned by training data. They indicate average word accuracies of 90% and correct understanding rates of 85% for queries made from novice users on commands in the domain. They also indicate that these accuracies go up to 98% and 95% respectively for experienced users.

	Recognition Accuracy of Jupiter
Telephone	100%
iPAQ	70%

Table 5.1: Our informal results from speech input to SLS's Jupiter weather system via telephone and iPAQ for 10 commands. SLS's more in-depth tests claim 98% accuracy over the telephone.

For comparative purposes, we also did our own brief test using the Jupiter system in the same environment as our own study. We used a list of ten suggested Jupiter queries to present to Jupiter via both the telephone and the iPAQ. Table 5.1 shows the result of our rough test by an experienced user.

As Table 5.1 indicates, we found the telephone speech interface recognition to be

100% accurate using a small set of commands. Interestingly, when we switched over to the iPAQ speech interface using the same exact domain and commands, the recognition accuracy dropped to 70%. This indicates a strictly worse rate of recognition accuracy on the iPAQ. This degradation in accuracy could be a result of many factors. First, the microphone on the iPAQ may not be tuned perfectly. We informally fiddled with the audio mixer equalizer and line-in settings. However, the microphone accuracy could potentially be improved by even more sophisticated configuration. Also, SLS's models have been trained for telephone-based dialogs through methods such as the Jupiter system. Since the microphones on the iPAQ and telephone are configured differently and the telephone is designed to get speech input at close range while the iPAQ is designed for slightly farther input range, it is possible that this is a contribution to the accuracy degradation. Since our tests use the iPAQ, this simple test is a clear indication that we should not necessarily expect recognition rates higher than that of 70%. After all, Jupiter has been tuned over many years and is designed for use over the telephone while our uses user-defined, rough domains over the iPAQ.

One important point about the data obtained from the Jupiter system is that the "correctness" of the output obtained is based on the output frame, not necessarily the exact output phrase. This means that the output phrase could be slightly wrong, but as long as the output frame was close enough, the correct command was executed. We discuss this difference more in-depth in the following sections.

5.1.3 Preliminary Findings

In this section we describe the preliminary results that we obtained from running a small informal study. It is important to note that the actual output that should be processed by the back-end application built by the user is the output frame, not necessarily the output phrase. When a user builds a domain in SpeechBuilder, the actions and keys are specified. When input is processed, the values of the actions and keys are set if an input word/phrase matches the specified values in the SpeechBuilder domain. Therefore, the output can be parsed more robustly even when Galaxy recognized only a subset of the entire phrase inputted. We utilize the output frame in our back-end application for command execution. However, since we are testing the recognition accuracy, for our results, we examine only the entire output phrase and the n-best list. This is because it is possible for Galaxy to misunderstand a word but still get the correct action/key frame output if two words are mapped to the same value. We did not want misunderstood words that by chance mapped to the same value to be counted as correctly understood words. Also, it is possible for Galaxy to have the correct understanding of the words said but also have additional words in the output phrase that could conflict. As we will mention later in the analysis, had we chosen to use the output frame instead, these chance misunderstandings and frame parsings would have improved our accuracy rates substantially.

The Jupiter-based comparative statistics above most likely make use of the output frame and other more sophisticated processing techniques that we did not use for analyzing our output in this study. Therefore, it is not surprising that our results are considerably worse than the comparative results. Still, the results that we obtained are interesting initial findings.

The Setup

In order to get some preliminary results, we created a sample Presentation Manager Scenario and ran a small informal study on four subjects. The scenario comprised of the three subdomains outlined in Section 3.1 (the Slide Layer domain, the Presentation Layer domain, and the Application Layer domain) and a composite domain of all the commands from the three subdomains. For each subdomain, we included the basic commands that a user might need to control a presentation on a laptop appropriate for the particular subdomain. The domains are all runnable via speech and T9.

We chose to run the experiment on the iPAQ since we are focusing on pervasive computing controlled via an iPAQ. Both speech and T9 could also be controlled from a desktop computer using the same commands and implementation.

To run the experiment we selected a set of eleven commands from each domain that covered the basic functionality and word set. We had each subject input these commands via speech and T9 for both the corresponding subdomain and the full domain, the domain that contains all the commands from each subdomain. We logged the output phrase, the n-best list, and the time elapsed for each set of commands inputted.

It is important to note the level of consistency of our data entry. For T9, our data entry is very consistent. Each subject performed the T9 input process only once for each subdomain. We then automatically inputted the same exact input string for the full domain. That way, we could be sure to preserve any typos and test the accuracy of the two domains consistently. However, with speech, we found it less accurate to first record the spoken commands and play them back due to the duplication of noise introduction. We therefore decided to have each subject speak the command twice, once for the subdomain and once for the full domain. Even though this method is not completely consistent since the subject may vary the way he speaks the command slightly and introduce additional error in our results, we decided to go with this method of testing.

As all of the setup descriptions indicate so far, this test is by no means a rigorous test. It is simply a rough indication of the potential of our system and what areas we need to concentrate on even more.

Raw Data

The tables in this section show the raw data from the log files we collected. For each set of commands inputted, the tables indicate how many commands were parsed 100% successfully (the output phrase matched the input command), how many commands were listed in the n-best list successfully, and how many T9 input sequences contained at least one typo or insertion/deletion.

Let us examine the first two rows of Table 5.2. The first row pertains to the Slide Layer commands processed by a Galaxy instance running the Slide Layer subdomain and the second row pertains to the Slide Layer commands processed by a Galaxy instance running the full domain containing all the commands. The second and fourth columns, labelled "Right", indicate how many commands were success-

Domain	**Speech Accuracy**		**T9 Accuracy**		
	Right	n-best	Right	n-best	typos
SL - Subdomain (11)	2	7	10	10	6
SL - Full domain (11)	2	4	10	10	6
PL - Subdomain (11)	2	5	10	11	6
PL - Full Domain (11)	0	1	7	8	6
AL - Subdomain (11)	4	7	10	10	7
AL - Full Domain (11)	1	4	10	10	7
Total Subdomain (33)	8	19	30	31	19
Total Full Domain (33)	3	9	27	28	19

Table 5.2: Subject 1 – Raw Data. This table shows the number of correct disambiguations for commands from the three layers: Slide, Presentation, and Application. Each domain was run on Galaxy with both the subdomain and full domain language models.

Domain	**Speech Accuracy**		**T9 Accuracy**		
	Right	n-best	Right	n-best	typos
SL - Subdomain (11)	5	7	11	11	1
SL - Full domain (11)	1	3	11	11	1
PL - Subdomain (11)	5	7	8	11	5
PL - Full Domain (11)	2	2	9	11	5
AL - Subdomain (11)	7	8	10	10	1
AL - Full Domain (11)	4	6	10	10	1
Total Subdomain (33)	12	22	29	32	7
Total Full Domain (33)	7	11	30	32	7

Table 5.3: Subject 2 – Raw Data.

fully disambiguated out of the eleven commands in the set. Similarly, the third and fifth columns, labelled "n-best", indicate for how many commands the correct disambiguation appeared in the n-best list out of the eleven commands in the set. The last column indicates how many commands contained at least one mistype, insertion, or deletion typo for the T9 version of the input.

The last two rows are a summation of the results for each subject. They indicate the total number of correct disambiguations appearing in the output and n-best list and the total number of typos that occurred for all thirty-three commands inputted by the subject for the corresponding type of Galaxy instance.

Domain	**Speech Accuracy**		**T9 Accuracy**		
	Right	n-best	Right	n-best	typos
SL - Subdomain (11)	3	6	10	10	3
SL - Full domain (11)	5	5	10	10	3
PL - Subdomain (11)	4	5	9	11	5
PL - Full Domain (11)	0	3	7	7	5
AL - Subdomain (11)	6	7	9	10	2
AL - Full Domain (11)	5	6	9	9	2
Total Subdomain (33)	13	18	28	31	10
Total Full Domain (33)	10	14	26	26	10

Table 5.4: Subject 3 – Raw Data.

Domain	**Speech Accuracy**		**T9 Accuracy**		
	Right	n-best	Right	n-best	typos
SL - Subdomain (11)	5	7	10	10	3
SL - Full domain (11)	3	4	10	10	3
PL - Subdomain (11)	5	6	9	10	1
PL - Full Domain (11)	1	2	8	10	1
AL - Subdomain (11)	6	6	10	10	1
AL - Full Domain (11)	4	6	10	10	1
Total Subdomain (33)	16	19	28	30	5
Total Full Domain (33)	8	12	26	30	5

Table 5.5: Subject 4 – Raw Data.

We have included these raw data tables to illustrate the variation between accuracy and speaker that we found. This does not imply that this speech recognition system is user-dependent, the accuracies could be different on another run-through. However, it does indicate a variation in accuracy rate.

Analysis of Data

Now we look at the overall results of the study. Figure 5.6 shows the correct recognition percentage rates over all four subjects for the set of 132 commands. These findings are more clearly illustrated by the graph in Figure 5-3. Here we examine the results on three bases: how they compare for a subdomain versus the full domain, how useful the n-best list is, and how speech and T9 compare.

	Speech		T9	
	Correct	**In n-best list**	**Correct**	**In n-best list**
Subdomain (132)	37%	59%	87%	94%
Full domain (132)	21%	35%	83%	88%

Table 5.6: Average disambiguation rates for Speech and T9 inputs over all study subjects.

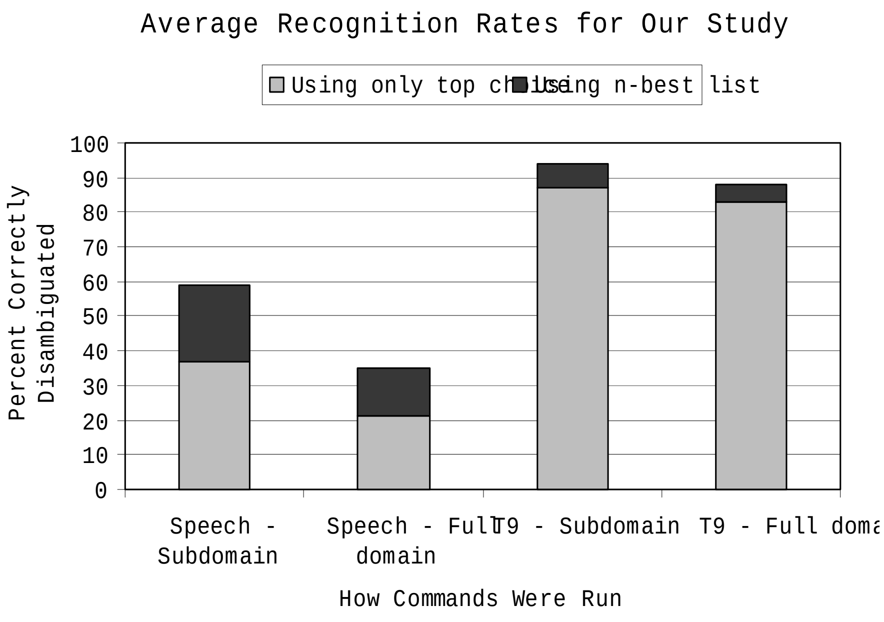


Figure 5-3: Graphical illustration of average disambiguation rates stated in Table 5.6. This graph clearly shows the disambiguation rates of Speech versus T9 and the potential improvement by using n-best list information.

It is not surprising that the recognition rates we obtained for speech are much higher for the subdomains than for the full domain. The full domain has roughly three times the number of words to select from and is therefore more prone to misunderstand more words given the natural error and variation in human speech. By creating smaller domains, our results indicate we can improve recognition rates by approximately 16% for speech and 4% for T9. It is important to remember that our test domains are fairly small and these numbers could look extremely different for

	Average Entry Time
Speech	7.4 seconds
T9	13.1 seconds

Table 5.7: Average entry time for each command over all study subjects. These times include entry, processing, and output feedback.

larger, more complex domains. A well-tuned iPAQ could also change these results significantly.

As expected, the rate of appearance of the correct disambiguation in the n-best list was consistently higher than the rate of correct disambiguation in the output. This implies that the n-best list can in fact be used in conjunction with other modes of input to further disambiguate the input and select the intended command. It is a matter of building a "smart" Selector to perform the mutual disambiguation. As mentioned earlier, by correct disambiguation, we mean the parsed output exactly matched the input. It is possible for the output frame to indicate the correct meaning and therefore result in the correct command execution, but for the output to be slightly different than the actual input. Our results include these situations as an error since the phrases did not exactly match. Table 5.8 shows the speech recognition rates based on the output frame. Notice they are strictly higher.

From this study, it is clear that the recognition rates were much higher for T9 than for speech. This is because the disambiguation necessary for T9 is much less than that for speech since the expected range of error is smaller. There are no errors introduced by accent, noise, or language differences. T9 disambiguation involves disambiguating some typos and the innate error involved with having 3-4 letters associated with each number. Even though the same input took on average 13.1 seconds to enter, process, and retrieve results via T9 as compared to 7.4 seconds via speech as shown in Table 5.7, the recognition accuracy seems to be astoundingly better. This by no means implies that T9 should be a replacement for speech. However, it does indicate T9 could be a useful tool for disambiguating error-prone speech input in certain situations. It is important to note that the average input time for speech was closer to 3.7 seconds since the 7.4 seconds recorded in the table includes the playback of the disambiguated output. This playback option is implementation specific and is not necessary for execution of the command. The T9 entry time does not include synthesized speech output playback.

These results are informal estimates. The next section helps to put them into perspective in the overall accuracy picture.

Putting the Results into Perspective

Some of the errors experienced in our study could be due to new user errors. Most of the subjects for this study had not had experience using the iPAQ. Although we gave them a brief explanation and allowed a practice run to become more comfortable with both the speech and T9 input methods, the results could have been biased by first time users. This is an indication that the user interface can be improved depending on the target audience.

Many more tests need to be run in order to come up with rigorous results. The speech recognition numbers here are surprisingly low. However, when you consider the comparative results from SLS's Jupiter weather system in Table 5.1 and the rough comparative results for the correct output frame in Table 5.8 versus the exact output parsing, it is clear that the low speech recognition rates are partly a function of test definitions.

Let us start from the top and work our way down. First, we start with the well-tuned Jupiter domain used over the telephone. Our rough test found the recognition to be perfect. When we estimated the accuracy over the iPAQ, the commands run with the same domain dropped down from a 100% to a 70% accuracy rate. These accuracy rates were measured using the output frame. When we took the same data from our study and examined the output frame with our domains, we found a 52% accuracy rate for speech subdomains. This accuracy rate dropped to 37% for the subdomain with the constraint of a correct output disambiguation. Similarly, we found a 31% accuracy rate for speech full domains using the output frame as opposed to the 21% accuracy rate for the correct output phrase constraint. These output frame rates are described in Table 5.8.

	Output Frame Accuracy for Speech
Subdomain	52%
Full Domain	31%

Table 5.8: Average accuracy rates of the output frame over all study subjects for a total of 132 commands.

These output frame rates are the right rates to compare to the comparative results in Table 5.1 since they use the output frame results also. The output frame results are also a good transition to understand the correct output phrase results from our study in Table 5.6. It indicates that our recognition accuracy of 37% for subdomains is not bad considering we are using the iPAQ and a domain built by speech novices. It is quite an improvement over the 21% accuracy using the full domain. And an even larger improvement can be made using the n-best list results in combination with multi-modal inputs such as T9.

5.2 Challenges and Future Work

It should be clear from this paper that the Galaxy system is trained to perform very accurate speech recognition and its architecture theoretically allows for utilizing individual components. That is why we chose to use it. However, there are always challenges to using ongoing and constantly improving research. Galaxy is an old system that has been developed over many years. It was originally meant only for speech. In order to use it without audio required some configuring, constant communication with and querying of SLS members, and lots of help from Galaxy experts.

As this research has been evolving over the past year, so has Galaxy. The increased functionality introduced by a new Frame Relay Server component allowed for an easier back-end connection and also an easier method of piecing together components with the Hub. As Galaxy and other input technologies continue to evolve, so do the possibilities for the topic of multi-modal input associated with this project.

This thesis thus far has developed a framework for multiple domains and disambiguating multi-modal input by extending the Galaxy architecture and focusing on the needs of pervasive oxygen systems. While we have built some of the groundwork for this framework and proved that it is possible, there is still much work left to be done before we will be using the system illustrated in Figure 4-3. The longer aim of the overall project is to build this entire structure for all useful inputs including pen-based inputs such as graffiti, touch-pad inputs, and spatial inputs such as gestures. In order to complete this structure, the broadcast functionality has to be implemented for parallel multi-domain processing and an intelligent Selector must be built to take advantage of multiple input information.

The results seem to indicate that the n-best list might be of use for disambiguating commands. However, this paper has not explored the specifications of what n should equal. Our study was run using very simple domains and tested only eleven commands for each domain. For our results, $n = 10$ made for a useful n-best list. Other larger studies might indicate different results for the usefulness of different length n-best lists. This is a topic that should be explored. Too large an n might result in many false-positives but too few might result in losing important information. The optimum value of n is an interesting extension.

Another major extension to this system is the implementation of a conversational command listener. Currently, in order to give commands via speech, a user must press a button to indicate the start of a command. A truly "smart" system would listen to all spoken dialog, distribute the parsing of sets of spoken words to many different processors, and have the capability of distinguishing a command from conversational words.

In this thesis, the framework and the steps for the system's basic functionality have been determined. In order for this project to become truly complete and usable, the process must be automated. Hopefully, having read this thesis, you will want to continue it.

Bibliography

[1] Michael Johnston Att. Finite-state Methods for Multimodal Parsing and Integration.

[2] S. Bangalore and G. Riccardi. Stochastic Finite-State Models for Spoken Language Machine Translation, 2000.

[3] C. Benoit, J. Martin, C. Pelachaud, L. Schomaker, and B. Suhm. Audio-Visual and Multimodal Speech Systems.

[4] Reginia M. M. Braga, Claudia M. L. Werner, and Marta Mattoso. Using Ontologies for Domain Information Retrieval.

[5] G. Chung and S. Seneff. Towards Multi-Domain Speech Understanding Using a Two-Stage Recognizer, 1999.

[6] Philip R. Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow. QuickSet: Multimodal Interaction for Distributed Applications. In *Proceedings of the fifth ACM international conference on Multimedia*, pages 31–40. ACM Press, 1997.

[7] Michael L. Dertouzos. The Future of Computing. *Scientific American*, July 1999.

[8] J. Glass and E. Weinstein. SpeechBuilder: Facilitating Spoken Dialogue System Development, 2001.

[9] Michael A. Grasso, David S. Ebert, and Timothy W. Finin. The Integrality of Speech in Multimodal Interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(4):303–325, 1998.

[10] Timothy Hazen, Stephanie Seneff, and Joseph Polifroni. Recognition Confidence Scoring and its Use in Speech Understanding Systems. *Academic Press*, 2002.

[11] G. James. Challenges for Spoken Dialogue Systems, 1999.

[12] M. Johnston and S. Bangalore. Finite-state Multimodal Parsing and Understanding, 2000.

[13] Michael Johnston. Deixis and Conjunction in Multimodal Systems.

[14] Timothy Hazen Lee. FST-Based Recognition Techniques for Multi-Lingual and Multi-Domain Spontaneous Speech.

[15] Karen McKenzie Mills and James L. Alty. Integrating Speech and Two-Dimensional Gesture Input - A Study of Redundancy between Modes. Computer Human Interaction Conference, pages 6–13, 1998.

[16] Hsin min Wang and Berlin Chen. Content-based Language Models for Spoken Document Retrieval. In *Proceedings of the fifth international workshop on on Information retrieval with Asian languages*, pages 149–155. ACM Press, 2000.

[17] Sharon Oviatt. Mutual Disambiguation of Recognition Errors in a Multimodel Architecture. In *Proceeding of the CHI 99 conference on Human factors in computing systems : the CHI is the limit*, pages 576–583. ACM Press, 1999.

[18] Sharon Oviatt. Ten Myths of Multimodal Interaction. *Communications of the ACM*, 42(11):74–81, 1999.

[19] Sharon Oviatt. Multimodal System Processing in Mobile Environments. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 21–30. ACM Press, 2000.

[20] Sharon Oviatt. Taming Recognition Errors with a Multimodal Interface. *Communications of the ACM*, 43(9):45–51, 2000.

[21] C. Pao, P. Schmid, and J. Glass. Confidence Scoring for Speech Understanding Systems, 1998.

[22] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-II: A Reference Architecture for Conversational System Development, 1998.

[23] Stephanie Seneff and Joseph Polifroni. Formal and Natural Language Generation in the Mercury Conversational System. 2000.

[24] Ronnie W. Smith and D. Richard Hipp. Using Expectation to Enable Spoken Variable Initiative Dialog. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, pages 123–130. ACM Press, 1992.

[25] Bernhard Suhm, Brad Myers, and Alex Waibel. Multimodal Error Correction for Speech User Interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(1):60–98, 2001.

Appendix A

Documentation

This documentation is a quick guide to understand the system, use the system, and extend the system. Although it is tailored to the Galaxy configuration installed on `money.lcs.mit.edu`, it can be helpful in understanding the overall framework implementation as well.

This Galaxy configuration includes use of the Frame Relay Server component built by SLS in addition to the ones shown in Figure 2-1. The system uses the current version of Galaxy located at `/usr/sls/current`, version 3.2.2.

A.1 Galaxy Basics

We used SpeechBuilder 2.1 located at `http://speech.lcs.mit.edu` to build our language model domains off of which Galaxy is configured. To create a language model, you must specify a set of actions (commands) and keys (concepts). Using this system, the language models built using SpeechBuilder can be used for any mode of input implemented. Currently, these modes include Speech and T9.

When you build your domain in SpeechBuilder, you must specify a URL to which Galaxy will post the output. The callback application can be in the form of a CGI script that accepts the output frame for parsing or a Java application that can accept an entire frame. The frame includes the output frame, the n-best list, and other useful information. In order to use the Java application callback option, you must use the

Frame Relay Server component and specify the URL as `relay:<app_name>` . The `<app_name>` for the sample callback application written by the SLS group is test. The source code is located at
`/usr/sls/current/oxygen/java/src/echo/Echo.java` .
The frame can be further parsed using the API written by Scott Cyphers at
`http://www.sls.lcs.mit.edu/cyphers/fr` . The Frame Relay Server is also backwards compatible and can send output to CGI scripts, though the Java application is much more convenient.

Once you specify all the components of a domain, compile it and get the domain tarball. The following Speech and T9 subsections give instructions for how to run the domain from there.

A.2 Creating a Domain and Running It

This section first describes how to set up the system for Speech and T9. It then goes into the step by step process of how to run the system using Speech and T9 input modes.

A.2.1 Setting Up the Domain For Speech

1. Make the domain in SpeechBuilder. Download it. Untar it in your SpeechBuilder directory.

2. Copy the menu file in your
   `/home/username/SpeechBuilder/DOMAIN.domainname` directory over to the `/usr/lib/menus/` directory on your iPAQ by ssh'ing into your iPAQ. You can name this file anything you want. As long as it is in the specified menu directory, it will be added to the menu on the iPAQ after you run `update-menus` and reboot the iPAQ.

   The menu command should be of the form

   ```
   galaudio /dev/sound/dsp money.lcs.mit.edu <username> <portnum>
   ```

(You can also run this command on your local machine with the galaudio binary file.)

3. Update the audio mixer on the iPAQ to one where you can tweak the sensitivity of the equalizer and line-in. We ended up setting the line-in at the middle of the gradient and the equalizer to three-fourths to the top. You can play around with the settings.

A.2.2 Running the Domain Using Speech

1. **Galaxy Servers**

   Run the Galaxy components on money (or whatever machine happens to be hosting Galaxy and your domains). Do this by typing the command `./oxclass.cmd yes` from your `/home/username/SpeechBuilder/DOMAIN.domainname/` directory. The yes indicates that you would like a separate window to pop up for each different Galaxy server running, namely the HUB, Local Audio, NL, Speech Recognizer, and Frame Relay Server (if you specified `relay:<appname>` as your domain URL).

2. **Callback Application**

   Run your Java callback application. You should see the acknowledgement of your application's name in the Frame Relay Server window.

   We run our Java app by first running:

   ```
   source /home/depot/Speech/path
   ```

   to include the frame parsing Java classes necessary in our path. And then run:

   ```
   xterm -title "MyEcho.java" -e java MyEcho localhost
     <remote_port num> test &
   ```

   to actually run the Java app. Note that the `<remote_port num>` is the port number that the frame relay server is listening on found in

`/home/username/SpeechBuilder/DOMAIN.domainname/sb.sas`

under the Frame Relay Server specifications.

3. **Speech Interface**

   Finally, run the menu item on your iPAQ. The `###call_answered###` message should be sent to your Java app and your welcome message specified by your Java app should be spoken to you. Now you can speak any commands that you specified in your SpeechBuilder domain. Your Java app should parse these commands based on the output frame or n-best list data and execute whatever commands you choose.

   Your domain should now be running. If galaudio hangs on the iPAQ (you say something and it just tells you to wait but does not respond), try changing the menu command to

   `galaudio /dev/sound/dsp money <username> <portnum> push`

   It could be that the environment is noisy and the sound input needs to be forced in when you finish your phrase.

A.2.3 Setting Up the Domain For T9

1. Make the domain in SpeechBuilder. Download it. Untar it in your SpeechBuilder directory.

2. Copy over the modified versions of

   `speechbuilder.pgm` and `speechbuilder-common.pgm`

   to `/home/username/SpeechBuilder/DOMAIN.domainname/`

   from `/home/sagarwal/pgms/new/` .

3. Copy over `<domainame>.wlex` over to the `T9_rec` directory currently found at `/home/sagarwal/T9_rec` . The `T9_rec` directory must contain:

   `fst_build_T9_recognizer.cmd, lexicon.pl, t9.mistypes,`

   `t9.map.baseforms, t9.baseforms` .

4. Run the perl program `lexicon.pl` in the `T9_rec` directory on `<domainname>.wlex` by setting the input file name in `lexicon.pl` to `<domainname>.wlex` and running `perl lexicon.pl` .

5. Remove the header lines from the `t9.baseforms` file so that it only contains actual words from the domain.

6. Make sure your path contains: `/usr/sls/current/sls/bin` . This path contains the FST composition tools built by SLS.

7. Run `./fst_build_T9_recognizer.cmd` to build the final FST, `t9.fst` .

8. Now you must run the system.

A.2.4 Running the System with T9

In order to run T9, there are a lot of pieces that all need to be running simultaneously. The order that these processes are started in is also very important. The following steps details how we have been running the system.

1. **T9 Symbol Recognizer**

   The first step is to run the T9 Symbol Recognizer. This recognizer will listen on the port that Galaxy normally expects the Speech Recognizer. Just as described in the body of this thesis, the T9 Recognizer simply replaces the Speech Recognizer.

   To run the T9 Recognizer, first set the path correctly run the command:

   ```
   xterm -title "T9 Symbol Recognizer"
         -e /home/sls/Galaxy-3-2-1/galaxy/bin/symbol_rec
         -port 7325 -fst /home/sagarwal/T9_rec/t9.fst &
   ```

   This will start up the T9 Recognizer in a new window. You can tell it to run whichever FST you choose. The command here runs our T9 FST.

2. **Galaxy Servers**

   To start the Galaxy Servers with T9 is no different than any Galaxy domain created with the SpeechBuilder application and compiled in conjunction with Galaxy run with Speech. To start the Galaxy Servers, first go to your `/home/username/SpeechBuilder/DOMAIN.<domainname>` directory that you downloaded from SpeechBuilder and unpackaged. Then, run the command:

   ```
   xterm -title "Galaxy components" -e ./oxclass.cmd yes &
   ```

   The yes indicates that you would like a separate window to pop up for each different Galaxy server running. Don't worry if the Speech Recognizer dies. It has been replaced by the T9 Symbol Recognizer so the Speech Recognizer can no longer connect to the Galaxy Hub.

3. **Back-End Application**

   Running the Java application that parses the output frame is the same as with Speech. We run our Java app by first running:

   ```
   source /home/depot/Speech/path
   ```

   to include the frame parsing Java classes necessary. And then running:

   ```
   xterm -title "MyEcho.java" -e java MyEcho localhost
     <remote_port num> test &
   ```

   to actually run the Java app. Note that the `<remote_port num>` is the port number that the frame relay server is listening on found in
   `/home/username/SpeechBuilder/DOMAIN.domainname/sb.sas`
   as described earlier.

4. **T9 Server: Interface between T9 GUI and T9 Input Server**

   The T9 Server is a simple Java Server that we wrote to accept the input from the T9 GUI and send it to the T9 Input Server. The T9 Server tells the T9

Input Server to wait until it receives data from the user. Run the following commands to start the T9 Server.

```
cd /home/sagarwal/T9Server
xterm -title "T9 Server" -e java T9Server &
```

5. **Callback Server**

   The Callback Server is more of a proof of concept component and is not a necessary step. We programmed our MyEcho.java application to send commands to the Callback Server to prove that multiple domains can be running at the same time with different input modes and the commands can all be sent to the same repository. This repository is the Callback Server, the segway into the Selector. Run the following if you want to run the Callback Server.

```
cd /home/sagarwal/CallbackServer
xterm -title "Callback Server" -e java CallbackServer &
```

6. **T9 Input Server for Symbol Recognizer**

   The T9 Input Server is the replacement for the regular Galaxy Audio Server. A generic server was written by the speech group to allow text entry into Galaxy. By modifying it to query the T9 Server for T9 numerical input, we are able to send T9 input to Galaxy. Run the T9 Input Server with the following command:

```
xterm -title "MyLintest"
      -e java MyLintest localhost <remote port num> mylintest
<username> &
```

   The original Input Server can be found at

```
 /usr/sls/current/oxygen/lintest/java/src/lintest/Lintest.java
```

A.3 Extending the System

As you can see from running the system, the basic implementation is there. There are also many opportunities for small and large additions to extend the system. We discuss a couple of the smaller ones in this section.

A.3.1 Optimizing the FST's

A small, yet powerful, addition is optimizing the mistype and insertion/deletion penalties embedded in the FST for the T9 Recognizer. We came up with mistype penalties based on proximity of numbers on the keypad. The insertion/deletion penalties are currently arbitrary and rather large. Playing with these penalty values could greatly improve the performance and usability of the T9 input mode.

The penalties are implemented via the `/home/sagarwal/T9_rec/t9.mistypes.fst` file.
The format for an FST file is as follows.

```
FSTBasic MinPlus
I 0
F 0
T 0 0 9 2 5.0
T 0 0 9 3 5.0
T 0 0 9 4 5.0
T 0 0 9 5 4.0
T 0 0 9 6 2.5
T 0 0 9 7 5.0
T 0 0 9 8 2.5
T 0 0 9 9 0.0
T 0 0 9 , 50.0
T 0 0 , 9 50.0
...
<continues similarly for other input numbers>
```

The first line is a standard header for an FST file. The second line indicates that the initial state is 0. The third line indicates that the final state is 0. As mentioned in the body of this paper, an FST is a directed graph of nodes representing states and directed arcs representing transitions from state to state. Each line beginning with a `T` represents a Transition arc. Let us take the fourth line as an example. In English, it reads: Transition arc from state 0 to state 0, if the input is 9 the output is 2 with a penalty of 5.0. Since the most likely output for an input of 9 is 9, the transition: `T 0 0 9 9` has the lowest penalty associated. The `,` represents a null input and therefore can be used for representing insertions and deletions in an FST.

A.3.2 Implement Graffiti

A major contribution of this thesis has been to introduce the capability of additional input modes using the same platform and processing techniques. In order to take advantage of that, new input modes must be introduced. We suggest graffiti as the next input mode.

To implement graffiti, an interface must be made that runs on the iPAQ, converts graffiti to letters, and sends the sequence of converted letters to a server (the T9 Server can simply be extended) for disambiguation. The other piece that must be completed is the Graffiti FST to account for misunderstood letters and misspellings with associated penalties. These two pieces alone are enough to implement graffiti. It can then be run using the same steps as T9.