A Framework and Toolkit for the Construction of Multimodal Learning Interfaces

Minh Tue Vo

April 29, 1998

CMU-CS-98-129

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Thesis Committee

Alex Waibel, Chair
Bonnie John
Tom Mitchell
Allen Gorin, AT&T Laboratories

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

Copyright © 1998 by Minh Tue Vo

This research was sponsored by the DARPA under Department of the Navy, Naval Research Office under grant number N00014-93-1-0806, and Project GENOA under grant number 97047-ISX/SOW-600066. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DARPA or the U.S. Government.

ABSTRACT

Multimodal human-computer interaction, in which the computer accepts input from multiple channels or modalities, is more flexible, natural, and powerful than unimodal interaction with input from a single modality. Many research studies ([Hauptmann89], [Nakagawa94], [Nishimoto94], [Oviatt97b], [Chu97], to name a few) have reported that the combination of human communication means such as speech, gestures, handwriting, eye movement, etc. enjoys strong preference among users. Unfortunately, the development of multimodal applications is difficult and still suffers from a lack of generality, such that a lot of duplicated effort is wasted when implementing different applications sharing some common aspects. The research presented in this dissertation aims to provide a partial solution to the difficult problem of developing multimodal applications by creating a modular, distributed, and customizable infrastructure to facilitate the construction of such applications.

This dissertation contributes in three main areas: theory of multimodal interaction, software architecture and reusable application framework, and rapid application prototyping by domain-specific instantiation of a common underlying architecture.

The foundation of the application framework and the rapid prototyping tools is a model of multimodal interpretation based on semantic integration of information streams. This model supports most of the conceivable human communication modalities in the context of a broad class of applications, specifically those that support state manipulation via parameterized actions. The multimodal semantic model is also the basis for a flexible, domain-independent, incrementally trainable multimodal interpretation algorithm based on a connectionist network.

The second major contribution is an application framework consisting of reusable components and a modular, distributed system architecture. Multimodal application developers can assemble the components in the framework into a new application, accepting default options when appropriate and providing application-specific customizations when needed.

The third major contribution is a design process backed by a workbench of tools to permit the rapid prototyping of a multimodal application. This design process systematically constructs customizations needed to interpret multimodal inputs in a given domain, allowing an application structure created in the proposed framework to be instantiated for that domain.

The application framework and design process have been successfully applied to the construction of three multimodal systems in three different domains.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my thesis committee, especially my advisor, Dr. Alex Waibel. Without the help and support of Dr. Waibel, together with Dr. Bonnie John, Dr. Tom Mitchell, and Dr. Allen Gorin, my dissertation would never have taken form. Everyone on the committee took precious time out of their incredibly busy schedules to discuss this thesis with me and to provide invaluable advice and comments. Dr. Waibel has been advising me during all my years at CMU, and I'm glad all the hard work he spent on me has finally born fruit. Not only is he my advisor, he considers me a friend and a collaborator as well, for which I am doubly grateful.

I am indebted to all my colleagues who took the time to read my drafts and comment on the dissertation, especially Jie Yang, Matthias Denecke, and Michael Bett. Jie and Matthias also helped me practice my thesis defense. Special thanks to Wes Mingin who proofread the thesis and provided technical writing expertise. Any errors that remain are of course my own.

Numerous people were involved with many software packages that contributed significantly to my work. I would like to thank Markus Baur, Michael Finke, Bernhard Suhm, Monika Woszczyna, and the rest of the JANUS team for their help with JANUS; Wayne Ward, Eric Thayer, and other people on the SPHINX team for their help with SPHINX; Stefan Manke for creating the NPen++ handwriting recognizer; and Xing Jing and Weiyi Yang for writing the speech recording plug-in.

There remain so many teachers and colleagues to whom I am grateful. I would like to mention Dr. Bernd Bruegge who introduced me to object-oriented software engineering; Cindy Wood who helped me collect Wizard-of-Oz data for my research; Gregory Cortis Clark who agreed to be my "guinea pig" and created an innovative multimodal application using my system; and Sharon Burks who always had an answer for everything. Many thanks also to everyone who took part in my user observation sessions.

I would also like to thank my parents for believing in me and encouraging me to do my best in all my endeavors. My father has a passion for life-long learning which has rubbed off on me. My mother took wonderful care of me while I was living with my parents, and even now, after I have been on my own for quite a while, she still reminds me all the time to eat properly and take time off from work to relax.

I am also grateful to my future parents-in-law who embraced me into their family and provided me with a second home during the years I spent in Pittsburgh.

I have saved the best for last. There are no words that can describe what my bride-to-be, the light of my life, means to me. I underestimated the magnitude of the task that I undertook when I enrolled in the doctoral program at CMU, and without Thuy by my side, encouraging me all the way, I would never have made it. Now that the ordeal is over, I can devote all my time to our happiness together. This is for you, my darling!

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Motivating Example 2
1.2 Target Class of Applications 3
1.2.1 Types of Inputs 3
1.2.2 Types of Tasks 4
1.2.3 Supported Applications 4
1.3 Outline 4
1.4 What Is Not Covered 7
1.5 Notes 7

CHAPTER 2 BACKGROUND AND RELATED WORK 8
2.1 Automatic Speech Recognition and Understanding 8
2.1.1 Types of Speech Recognizers 8
2.1.2 Acoustic Modeling 10
2.1.3 Language Modeling 10
2.1.4 Measuring Recognition Accuracy 11
2.1.5 Speech Understanding 12
2.1.6 Speech Recognition APIs and Toolkits 14
2.2 Gesture and Handwriting Recognition 14
2.2.1 Handwriting Recognition 15
2.2.2 Pen-Based Gesture Recognition 16
2.2.3 Gesture-Enabled User Interface Toolkits 18
2.3 Other Input Modalities 19
2.4 Multimodal Human-Computer Interaction 20
2.4.1 General Discussions and Simulation Studies 20
2.4.2 Multimodal Integration Approaches 24
2.4.3 Actual Systems 26
2.4.4 Frameworks and Toolkits 27

CHAPTER 3 SEMANTIC INTEGRATION OF MULTIMODAL INPUTS 30
3.1 Input Modeling and Interpretation 30
3.2 A Multimodal Semantic Model 32
3.2.1 Combination of Multimodal Input Signals 32
3.2.2 Meaning of Multimodal Input Events 36
3.3 Multimodal Input Modeling 39
3.3.1 Traditional Grammar Formulations 39
3.3.2 The Multimodal Grammar Language 40
3.4 Multimodal Input Integration 43
3.4.1 Basic Mutual Information Network 43
3.4.2 The Multi-State Mutual Information Network 46
3.4.3 Training the MS-MIN 49

CHAPTER 4 MULTIMODAL APPLICATION FRAMEWORK ARCHITECTURE 50
4.1 Object-Oriented Concepts and Design Patterns 50
4.2 Overall System Design 52
4.2.1 System Architecture 54
4.2.2 Component Interface and Implementation 55
4.2.3 Input Processing Issues 55
4.3 Existing Software Components 57
4.3.1 The JANUS Speech Recognizer 57
4.3.2 The SPHINX Speech Recognizer 58
4.3.3 The NPen++ Handwriting Recognizer 58
4.3.4 The NetscapeSRec Speech Recorder Plug-In 59
4.4 Speech Components 60
4.4.1 The SRecServer Speech Recorder 60
4.4.2 The SpeechRecorder Interface 60
4.4.3 The SpeechRecognizer Interface 62
4.5 Pen Components 63
4.5.1 The XPRecServer Pen Recorder 63
4.5.2 The TmplGRec Gesture Recognizer 64
4.5.3 The PenRecorder Interface 69
4.5.4 The PenRecognizer Interface 70

4.6 Communication Layer 71
4.6.1 Client/Server Model 71
4.6.2 Remote Service Request 72
4.6.3 Switchboard-Based Communication 72
4.7 Graphical User Interface 73
4.7.1 SpeechPanel 75
4.7.2 PenPanel 76
4.7.3 InputCoordinator 77
4.7.4 MultimodalApplet 77

CHAPTER 5 MULTIMODAL DESIGN AND RAPID PROTOTYPING 81
5.1 Design Process 81
5.1.1 Selecting Action Frames and Parameter Slots 82
5.1.2 Designing the Input Model 82
5.1.3 Generating Unimodal Language Models 85
5.1.4 Instantiating the Multimodal Semantic Integrator 85
5.1.5 Implementing the Parameter Extraction Postprocessing 87
5.2 Multimodal Grammar Implementation 88
5.3 Visual Grammar Designer 92
5.3.1 Graphical Display of Grammar Components 92
5.3.2 Drag-and-Drop Editing 92
5.4 Random Sample Generator 94
5.5 N-gram Language Model Generator 94
5.5.1 Basic N-gram Counting Algorithm 95
5.5.2 Handling Recursive Grammar References 96
5.5.3 Computing N-gram Probabilities 98
5.6 Interpretation Engine Builders 98
5.6.1 Input Preprocessor Generator 99
5.6.2 Integration Network Generator 99
5.6.3 Postprocessor Generator 100

CHAPTER 6 DESIGN EXAMPLE: A MAP SYSTEM 102
6.1 Requirement Analysis 102
6.2 Design 104

6.3 Implementation 108

CHAPTER 7 EVALUATION 111

7.1 Evaluation of the Development Procedure 111

7.1.1 Portability 111

7.1.2 Ease of Development 114

7.1.3 Incremental Improvement 117

7.2 Evaluation of the Product 119

7.2.1 Performance of the MS-MIN 120

7.2.2 User Observation Results 124

CHAPTER 8 CONCLUSION AND FUTURE DIRECTIONS 140

8.1 Contributions 140

8.1.1 Theory of Multimodal Interaction 140

8.1.2 Software Architecture and Reusable Framework 141

8.1.3 Design Process and Supporting Tools 142

8.2 Future Directions 143

APPENDIX A GLOSSARY 145

APPENDIX B SUMMARY OF THE UNIFIED MODELING LANGUAGE 149

APPENDIX C SUMMARY OF DESIGN PATTERNS 150

C.1 Abstract Factory 150

C.2 Adapter 151

C.3 Factory Method 151

C.4 Observer 152

C.5 Template Method 153

C.6 Visitor 154

APPENDIX D USER OBSERVATION PROTOCOL FOR THE MAP APPLICATION 155

BIBLIOGRAPHY 172

INDEX 190

LIST OF FIGURES

Figure 1. Nigay and Coutaz's Multimodal Design Space 23

Figure 2. Input Grouping by Temporal Proximity 34

Figure 3. Alignment and Joint Segmentation of Multimodal Inputs 36

Figure 4. Multimodal Grammar Structure 42

Figure 5. Mutual Information Network Architecture 45

Figure 6. Path Score of Input Segmentation and Labeling 47

Figure 7. Output Path Over Multidimensional Inputs 48

Figure 8. Multi-State Mutual Information Network Architecture 49

Figure 9. Multimodal Application Framework Architecture 54

Figure 10. Gesture/Handwriting Combination 57

Figure 11. SpeechRecorder Interface 61

Figure 12. SpeechRecognizer Interface 62

Figure 13. Preprocessing of Gesture Strokes 65

Figure 14. Gesture Recognition Example 67

Figure 15. PenRecorder Interface 69

Figure 16. PenRecognizer Interface 70

Figure 17. Multimodal Applet User Interface 74

Figure 18. Control Flow for Multimodal Input Interpretation 79

Figure 19. Class Hierarchy for Multimodal Grammar Components 88

Figure 20. Implementing Optional and Repeating Grammar Nodes 89

Figure 21. Grammar Traversal with Polymorphic Recursive Method 90

Figure 22. Grammar Traversal with Visitor 91

Figure 23. Multimodal Grammar Designer 93

Figure 24. Example of State Machine for Input Preprocessing 99

Figure 25. Preprocessed Parse Tree for a Macro Concept 99

Figure 26. Postprocessor Skeleton for Parameter Extraction 101

Figure 27. Multimodal Map Application 110

Figure 28. Multimodal Appointment Scheduler Application 112

Figure 29. Multimodal Football Application 113

Figure 30. MS-MIN Incremental Learning 122

Figure 31. Vocabulary Acquisition During MS-MIN Training 123

Figure 32. Network Growth During MS-MIN Training 124

Figure 33. Modality Distribution in User Data 127

Figure 34. Modality Distribution for Each Task 127

Figure 35. Distribution of Interpretation Error Categories 135

Figure 36. Number of Multimodal Commands to Complete a Task 136

Figure 37. Incremental Learning with Real Data 137

Figure 38. Vocabulary Acquisition with Real Data 138

Figure 39. Network Growth with Real Data 138

Figure 40. Additional Incremental Training of Generated Network 139

Figure 41. The Abstract Factory Design Pattern 150

Figure 42. The Adapter Design Pattern 151

Figure 43. The Factory Method Design Pattern 152

Figure 44. The Observer Design Pattern 152

Figure 45. The Template Method Design Pattern 153

Figure 46. The Visitor Design Pattern 154

LIST OF TABLES

Table 1. Supported Applications 5

Table 2. Basic Gesture Vocabulary 67

Table 3. Gesture Recognition Accuracy for 5 Shapes 68

Table 4. Speech Recording/Recognition States 75

Table 5. Action Frames and Parameter Slots for QUICKTOUR 104

Table 6. Size of Software Component Libraries 114

Table 7. QUICKTOUR Application Development Effort 115

Table 8. QUARTERBACK Application Development Effort 116

Table 9. Incremental Improvement in QUICKTOUR 118

Table 10. MS-MIN Interpretation Accuracy on Artificial Data 121

Table 11. Values of z_N for Two-Sided $N\%$ Confidence Intervals 121

Table 12. Discarded Input Events in User Data 126

Table 13. Speech Recognition Accuracy 128

Table 14. Speech Recognition Accuracy After Adaptation 128

Table 15. Gesture Recognition Accuracy 129

Table 16. MS-MIN Interpretation Accuracy on Real Data 130

Table 17. Interpretation Accuracy for Each Input Type 131

Table 18. Effect of Recognition Errors on Interpretation Accuracy 131

Table 19. Interpretation Accuracy for Each Participant 133

Chapter 1
INTRODUCTION

Human-computer interface technology has come a long way since the early days of toggle switches and light-panel readouts. However, human-computer interaction still cannot compare even remotely to the richness and flexibility of communication taken for granted among humans. In everyday life we convey a wealth of information through a multitude of communication channels, among them speech, gesturing, writing, drawing, facial expressions, gaze, and many others. The advent of the graphical user interface (GUI) paved the way for computer applications that attempt to take advantage of the rich human communication modes by presenting information over multiple channels, including text, images, sound, video, virtual reality, etc. This form of *output* presentation has become familiar under the designation of *multimedia*. In contrast, applications that take advantage of multiple *input* channels have been appearing much more slowly. Because input channels or sources are also called *input modes* or *modalities*, such applications are said to support *multimodal human-computer interaction*.

Many researchers have pointed out the benefits of multimodality. Numerous user studies (e.g., [Hauptmann89], [Nakagawa94], [Nishimoto94], [Oviatt97b], [Chu97]) have reported that multimodal human-computer interaction results in greater flexibility and naturalness of expression than unimodal input methods, to the point where people prefer multimodal interfaces. Accordingly, the work presented in this dissertation rests on the assumption that multimodality is desirable and only explores issues involved in the construction of multimodal applications.

The thesis of this dissertation is that there is enough commonality among multimodal applications to justify the development of a common infrastructure and design methodology for their construction.

First, a broad class of multimodal applications supports a common notion of input interpretation, namely the derivation of an action or command to perform in response to user input. This lays the foundation for a multimodal semantic model that forms the core of a design process applicable to all applications belonging to that class. This design process incorporates a common multimodal interpretation algorithm structure that can be instantiated for specific applications.

Second, certain basic capabilities required to support commonly used input modalities are essentially the same across applications; these capabilities include the recording, recognition, and synchronization of inputs from the supported channels. The common functionality factors into reusable software components, and multimodal applications can be constructed with a system architecture that makes use of such components to reduce development time.

1.1 Motivating Example

Consider a map application that can display a geographical map and answer various queries such as: search for places matching some criteria and display their locations on the map, find the distance between two places and the best way to go from one to the other, manipulate the map view by zooming and panning, etc. The user can issue these queries multimodally by speaking and/or drawing gestures on the screen. Similar applications have been studied or implemented by many other researchers [Neal91][Matsu'ura94] [Cheyer95][Oviatt96][Martin97].

The construction of such an application involves many components, some specific to the map system and others mandated by the multimodal input requirements. It is unlikely that the map display and the underlying geographical database can be reused in a text editor or an appointment scheduler; however, those two applications, like the map system, must also have facilities for recording and recognizing speech and pen inputs if they are to support verbal and gestural commands. These facilities, having little to do with map manipulation, text editing, or appointment scheduling, remain essentially the same across all three applications.

The visual user interface component of the map system does not require great computational power; in fact, it is totally feasible to deploy it on the World Wide Web to run inside Web browsers. However, other components of the system may be very computation-intensive, especially the speech and gesture recognition subsystems. The computationally demanding processes can be offloaded to high-end workstations as servers, with the added advantage that multiple user interface components can make use of a single server on a time-shared basis. The infrastructure required by this distributed architecture is evidently not specific to the map application.

The developers of the map application must select and implement policies that govern the synchronization and interpretation of multimodal inputs. In other words, the application developers must answer questions such as, "What constitutes a verbal/gestural command (i.e., how to group input data from the two input channels)?", "What does it mean to interpret a spoken utterance and/or a gesture?", "How does one derive the correct interpretation given the inputs?", etc. Some details of the relevant policies may be specific to the map application, but the broad outline should be applicable to the text editor, the appointment scheduler, and other speech- and pen-enabled applications as well. In fact, the design process that the map application developers follow may contain steps that can be generalized to apply to the design of other applications.

The map application example illustrates the feasibility of deriving a common infrastructure and design methodology that could serve as the foundation of many multimodal applications in diverse domains. This example will be revisited in Chapter 6 which describes an actual implementation of a multimodal map system using the framework and design process presented in this dissertation.

1.2 Target Class of Applications

Computer applications can be classified along many dimensions using many different attributes. However, for the purpose of determining whether an application can be constructed using the framework and design process described in this dissertation, there are two highly relevant attributes:

- The *types of inputs* that the application must support, and
- The *types of tasks* that the user must perform using the application.

Depending on those attributes, an application may be

- *Strongly supported*, i.e., the application can be constructed using the framework and design process described in this dissertation;
- *Supported*, i.e., the application fits the proposed framework, but some modules may have to be added to accommodate this type of applications;
- *Not supported*, i.e., it is not possible to fit the application into the constraints of the proposed framework.

1.2.1 Types of Inputs

There are many types of inputs that can convey information from a human user to a computer program. The traditional input devices include the keyboard for typing and the mouse for clicking and dragging. Speech is becoming more and more popular as an input modality because of recent advances in automatic speech recognition technology. Lipreading has been shown to be an effective aid in improving speech recognition accuracy. Pen devices such as digitizing tablets, light pens, touch-sensitive screens, etc. can accommodate handwriting and pen-based gestures (simple pointing as well as signs and symbols drawn with the pen device). 3-dimensional hand gestures can be tracked by devices such as the Data Glove [Eglowstein90] and visual hand modeling techniques [Rehg93] [Kuch95]. Gaze tracking devices [Jacob91][Baluja94] enable the eye to be used for input. Future devices may be able to track facial expressions or body language, etc.

All of the above input modalities can be interpreted as *information streams*, or sequences of packets carrying information that may contribute to the discovery of what to do with the input[†]. These information packets are words or groups of words in the case of typing, speaking, or writing; pointing actions as well as signs and symbols in the case of gesturing; eye fixations (periods of relative stability in eye position) and saccades (sudden, rapid eye movements) in the case of gaze tracking; and so on.

Because these input modalities are information streams, they can all be accommodated by the semantic model described in Chapter 3. However, at the present only speech

[†] Chapter 3 defines this notion more precisely.

and pen modalities are strongly supported in the sense that the framework described in Chapter 4 includes software components supporting speech and pen inputs.

1.2.2 Types of Tasks

Multimodal systems currently make use of inputs in two ways: either for *data entry* or for issuing a *command*.

The data entry task is exemplified by dictation systems. Speech input is converted to words that a word processor or text editor records verbatim. There is no interpretation beyond the conversion of sounds into words. Another example is a gesture-based drawing application in which the user can make rough sketches of geometric shapes, and the program converts the sketches to perfectly regular drawings. The only interpretation involved is the recognition of geometric shapes from the raw sketches.

In contrast, systems that use multimodal inputs as commands must determine what action to take based on the information conveyed by the inputs; in other words, the inputs have to be interpreted. The result of input interpretation is a command that changes the state of the application in some way.

The semantic model described in Chapter 3 strongly supports the use of inputs to issue commands. However, data entry can also be accommodated by interpreting the data-carrying inputs as a special command that changes the state of the application by adding data in a predefined manner.

1.2.3 Supported Applications

Table 1 on page 5 summarizes the types of applications supported by the framework and design process described in this dissertation. The entries indicated by "**?**" are not precluded but it is uncertain whether the corresponding modalities would be useful for the given tasks.

1.3 Outline

The work presented in this dissertation addresses three challenges in the construction of multimodal applications:

- The development of a multimodal interpretation algorithm that can integrate information from multiple input modalities in a general and application-independent way. This necessitates a clear notion of what exactly is meant by multimodal interpretation.
- The design of a system architecture that clearly separates the functionality that remains the same across different multimodal applications from the application-specific functionality, such that the system infrastructure can be reused in different applications.

		Type of task	
		Data entry	Command
Type of input	Speech	✓	☑
	Handwriting	✓	☑
	Pen gestures	✓	☑
	3-D gestures	✓	✓
	Lip-reading	✓	✓
	Gaze tracking	**?**	✓
	Keyboard	✓	✓
	Mouse	✓	✓
	Facial expressions	**?**	✓

Table 1. Supported Applications

☑ = strongly supported; ✓ = supported; **?** = not precluded

- The rapid prototyping of a multimodal application. This requires the instantiation of the application structure and the multimodal interpretation algorithm for a specific application domain.

During the course of my research in multimodal interfaces, I have implemented several multimodal applications belonging to different domains (i.e., developed to solve different tasks). I have distilled from those experiences an underlying *design process* backed by a collection of reusable software components and tools. The software components and the underlying system architecture constitute an *application framework* that allows new multimodal applications to be built in a modular fashion by connecting existing components and providing application-specific customizations only when necessary. The associated *toolkit* helps automate certain steps in the design process when the multimodal interpretation algorithm is instantiated for a particular application.

The contributions of this research are threefold:

- A theory of multimodal interaction based on a multimodal semantic model, and a domain-independent multimodal integration algorithm that combines information from multiple sources following the semantic model;
- A modular, distributed, customizable application framework to facilitate the construction of multimodal applications from reusable components;
- A design process and a supporting toolkit to facilitate the instantiation of multimodal applications for specific domains, within the application framework, using the semantic model and the multimodal integration algorithm.

The remainder of this dissertation presents the results of this research in a systematic way.

Chapter 2 (page 8) surveys current literature on topics related to the research described in this dissertation. These topics include the automatic recognition of speech, gesture, and handwriting, as well as user studies in multimodal human-computer interaction and actual multimodal systems. This background discussion will help place the present work in the context of current multimodal research.

Chapter 3 (page 30) lays a theoretical foundation for the notion of interpreting multimodal inputs. The chapter describes a *semantic model* that offers a definition of what it means to interpret user input in a multimodal application. This leads to the development of the *Multimodal Grammar Language* (MMGL), a *multimodal input modeling language* that enables application developers to describe the kind of inputs an application expects, as well as their semantics. The chapter also presents an information-theoretic network architecture that performs *semantic integration* of multimodal inputs by alignment and joint segmentation of multiple information streams.

Chapter 4 (page 50) presents the *Multimodal Application Framework* (MMApp), a modular, distributed, and customizable infrastructure that can be used to develop multimodal applications. The components that compose the framework are described in detail and placed in the context of a common multimodal system architecture.

Chapter 5 (page 81) describes the design process that instantiates and customizes a common multimodal interpretation algorithm for a particular application. This design process makes use of the *Multimodal Toolkit* (MMTk), a design workbench that enables the automatic construction of components needed in the design process, based on an MMGL model of user inputs.

Chapter 6 (page 102) places the discussion in the previous three chapters into context by presenting the complete development of a map application, QUICKTOUR, using MMApp and MMTk, from requirement analysis to design and implementation. The principles previously described are clarified when they are applied to the construction of the QUICKTOUR application.

Chapter 7 (page 111) offers some validation of the work presented in previous chapters. The development procedure using MMApp and MMTk is evaluated with respect to development effort and portability (i.e., applicability to different task domains). To illustrate the workability of this development procedure, one of its products—the QUICKTOUR application—is evaluated with respect to enabling real users to accomplish given tasks using the multimodal interface.

Chapter 8 (page 140) summarizes the contributions of this dissertation and outlines some directions for future research.

1.4 What Is Not Covered

As mentioned in section 1.2.1 above, the multimodal semantic model proposed in this dissertation supports many input modalities, but only speech and pen modalities are strongly supported by the software. The discussion in this dissertation does not include the specifications and implementation details of software modules to record and recognize inputs from any modality other than speech, pen gesture, and handwriting.

Even for the strongly supported modalities, the description of the application framework in Chapter 4 is limited to the interface protocols between the recognizers and the rest of the framework, omitting many of the details involved in setting up speech and gesture/handwriting recognizers as well as the training and optimization procedures necessary for high recognition performance. The focus of the work described herein is the combination of information from multiple modalities, hence no attempt was made to optimize the component technologies.

The multimodal interpretation algorithm in Chapter 3 works on the outputs of the modality recognizers (e.g., text strings from the speech recognizer, gesture shapes from the gesture recognizer, etc.) rather than raw input signals. As such, the algorithm currently has no provision for constraining the recognition of one modality using another modality. Theoretically speaking, it should be possible to achieve higher recognition accuracy for each individual modality using information contained in other modalities. This is a topic for future research. The issue of data fusion at different levels of input representation is briefly discussed in section 3.2.1.

1.5 Notes

Some terms used throughout this dissertation need to be clearly defined or clarified with respect to the specific connotations of their use. Appendix A contains a compilation of these terms.

A few typographical conventions have been followed to improve readability. Names of programming entities such as classes, methods, and MMGL nodes are in a Helvetica 12 point typeface. Code and input examples are in `Courier 11 point`.

All company and product names that appear in this document are trademarks of their respective holders.

Chapter 2
BACKGROUND AND RELATED WORK

This chapter presents an overview of research related to multimodal human-computer interaction. The first two sections discuss the processing of input data in two major modalities—speech and pen—that are strongly supported by the application framework described in this dissertation. The next section presents an overview of other input modalities that fit within the constraints of the framework but are not as strongly supported. The last section surveys the literature on combining multimodal inputs in user interfaces.

2.1 Automatic Speech Recognition and Understanding

Automatic speech recognition (ASR) refers to the process of mapping an audio data stream to a sequence of words. This has been a target of active research since the 1950's. Despite dramatic advances that have been made since the early days of speech recognition research, the problem of recognizing unconstrained speech is still far from being solved. However, speech science and computer technology have advanced to the point where large vocabulary, continuous dictation on a desktop personal computer has become feasible, as attested by recent commercial products from Dragon Systems and IBM. [Waibel90] presents a selection of research papers on various speech recognition topics. [Jelinek98] is a more up-to-date compilation of speech recognition techniques.

2.1.1 Types of Speech Recognizers

Speech recognizers are usually classified along several dimensions that introduce serious design difficulties or significantly degrade performance. The most notable dimensions are briefly described below. Some materials in the following discussion are from [Waibel90].

Isolated and Continuous Speech

Isolated speech is spoken one word at a time, whereas the words are connected in continuous (normal) speech. *Continuous speech recognition* (CSR) is much more difficult than *isolated* (or *discrete*) *word recognition* (IWR) because the absence of clear word boundaries in continuous speech creates more words and phrases that are confusable (e.g. "I love you" and "isle of view"), and poorer articulation as well as stronger interphoneme coarticulation results in much greater variability in continuous speech.

Continuous speech can be further classified as *spontaneous speech* or *read speech*. Read speech, typically used in dictation systems, is much easier to recognize than spontaneous speech which may contain speaker induced noises (lip-smacks, pops, clicks, coughing, sneezing), hesitations, false starts, and other natural speech phenomena.

Because multimodality promotes flexibility and naturalness of expression, it is usually desirable to use a spontaneous, continuous speech recognizer in a multimodal application supporting speech, if a recognizer exists that can provide acceptable accuracy for the kind of inputs the application expects.

Vocabulary Size

Recognition accuracy and efficiency vary inversely with the vocabulary size as more words introduce more confusion and require more processing time. One way to manage the resulting complexity is to exploit higher-level knowledge (e.g., syntactic or semantic knowledge) to constrain the set of words that are legal at a given time. The constraining power of such high-level knowledge in a *language model* can be measured by *perplexity*, roughly the average number of words that can occur at any decision point.

Multimodal applications supporting speech may require a large vocabulary if flexibility is an important design goal. To achieve acceptable recognition performance, it is usually necessary to restrict at least the domain or the topic of spoken utterances. It would be ideal if speech recognition systems could acquire vocabulary during actual use, but this is still a subject of research.

Speaker Dependence/Independence

A speech recognition system has a number of model parameters that can be adjusted to fit a particular speaker's speech. Such a *speaker-dependent* system can achieve higher accuracy than a *speaker-independent* system that must model a variety of speakers' voices. However, accuracy comes at a price because a speaker-dependent system must be trained separately for each new speaker. Some systems take a speaker-independent approach but also incrementally adapt to new speakers.

For maximum flexibility, multimodal applications should use speaker-independent speech recognizers with speaker adaptation capabilities if possible.

Environmental Factors

Variability and noises in speech can also severely degrade recognition performance. The channel characteristics play an important role; recognition systems that achieve respectable performance on high-quality speech recorded using a close-talking microphone may deliver mediocre performance on telephone-quality speech or speech recorded using a table microphone. Environmental and speaker induced noises also affect recognition accuracy, as do cross-talk (simultaneous talking by several people, also called the "cocktail party effect"), speaking rate, and speaker stress.

For desktop applications it is usually feasible to employ high-quality microphone and speech recording hardware. However, in some deployment situations (e.g., palmtop computers, wearable computers for field workers, etc.) the speech recognition system may have to handle noisy input. Achieving high accuracy in these situations is still an active goal of current research.

2.1.2 Acoustic Modeling

Digitized speech waveforms are usually transformed to a more compact representation that retains or even enhances perceptual cues thought to be relevant to recognition. This *feature extraction* process is usually based on spectral-encoding strategies that exploit information contained in the power spectrum of the speech signal. Important examples include *linear predictive coding* (LPC) [Markel76], *Fast Fourier Transform* (FFT), and *cepstral analysis* [Schafer75]. The feature vectors extracted from the speech signal can be compressed further by clustering them using *vector quantization* (VQ) [Gray84].

The encoded speech signals are matched against *acoustic models* which represent speech units (phonemes, syllables, words, etc.). Successful acoustic modeling techniques include stochastic and connectionist approaches. The most popular stochastic modeling approach employs *Hidden Markov Models* (HMMs) [Rabiner89] to estimate the probability that a speech unit, say a phoneme, is active at some point in time given the speech signal at that point. Connectionist approaches train *neural networks* to perform that probability estimation. Neural network algorithms that have been studied include the Time Delay Neural Network (TDNN) [Waibel89a][Waibel89b][Waibel89c], Learning Vector Quantization (LVQ) [Kohonen88], and Linked Predictive Neural Network (LPNN) [Tebelskis90]. Hybrid approaches that combine neural networks with conventional approaches such as HMM and Dynamic Time Warping (DTW) have also been studied with good results [Bourlard88][Burr88][Huang88][Lippmann87][Sakoe87].

2.1.3 Language Modeling

Given the acoustic probabilities estimated by the acoustic models, a speech recognizer must perform a *search* through the space of all possible sentences to find the sentence that best matches the acoustic signals, in the sense of maximizing the *a posteriori* probability of the sentence given the acoustics. The complexity of the search grows exponentially with vocabulary size and quickly becomes unmanageable without some way of constraining the search to only the most promising regions of the hypothesis space. *Language modeling* refers to the exploitation of non-acoustic sources of information (e.g., syntax, semantics, pragmatics, dialogue, etc.) to constrain the search.

One type of language modeling, termed *N-gram modeling* [Jelinek90], employs a statistical approach to predict the likelihood of encountering a word, based on preceding words. A bigram model (N=2) uses a single preceding word whereas a trigram model (N=3) uses two preceding words. N-gram language models become too large very quickly as N gets larger, hence *tree-based* statistical language models have been developed to estimate dependencies on a large number of preceding words using decision trees [Bahl89].

A *grammar-based* language model defines a set of (possibly infinite) acceptable sentences via a grammar. The approaches that have been proposed use finite-state grammars [Lowerre80], context-free grammars [Ney87], or unification grammars [Hemphill89]. Finite-state grammar approaches are the most easily and efficiently implemented.

A desirable goal for multimodal human-computer interaction is to let the user employ any speaking style, vocabulary, phrasing, etc. within the designated domain. Grammar-based language models are usually inadequate for this purpose as they restrict the user to a fixed set of sentences. Statistical language models favor sentences covered by the models, yet still allow for almost arbitrary sentences, and thus are more suitable when flexibility is an important design criterion. However, the choice of language model can be made on a per-application basis without affecting how the speech recognizer is interfaced to the application program. If an application would benefit from a fixed command language, perhaps because recognition performance requirements dictate tight constraints on the language, it would be better to use a grammar-based language model.

2.1.4 Measuring Recognition Accuracy

There are several ways to measure speech recognition accuracy. Let us assume a data set composed of N_s spoken utterances that have been transcribed into N_s sentences consisting of N_w words. Processing the utterances with a speech recognizer produces N_s recognition hypotheses. Each hypothesis sentence can be aligned with the corresponding reference (transcribed) sentence so as to maximize the number of matching words. For instance, the reference sentence "Please zoom out five times" and the hypothesis sentence "Zoom in fifty five times" would be aligned as follows:

```
REF:  PLEASE zoom OUT ***** five times
HYP:  ****** zoom IN  FIFTY five times
```

There are 3 correctly recognized words (in lowercase) and 3 errors (in uppercase). The first error is a *deletion* (the word "please" was omitted from the reference sentence), the second is a *substitution* (the reference word "out" was replaced by "in"), and the third is an *insertion* (the word "fifty" was not in the reference sentence).

The *sentence accuracy* over the data set is the fraction of hypothesis sentences that match the corresponding reference sentences perfectly:

$$SA = \frac{n_s}{N_s}$$

where n_s is the number of correctly recognized sentences.

The *word correct rate* is the fraction of words that are counted as correct when hypothesis and reference sentences are aligned:

$$WR = \frac{n_w}{N_w}$$

where n_w is the number of matched words in the aligned sentences.

The *word error rate* is the number of word errors divided by the total word count:

$$WE = \frac{n_{del} + n_{ins} + n_{sub}}{N_w}$$

where n_{del}, n_{ins}, and n_{sub} count the deletions, insertions, and substitutions, respectively.

The *word accuracy* is the complement of the word error rate:

$$WA = 1 - WE = \frac{N_w - n_{del} - n_{ins} - n_{sub}}{N_w}$$

Typically the word accuracy is lower than the word correct rate because of the insertion errors (it is easy to see that each reference word is either correct, deleted, or substituted, hence $n_w + n_{del} + n_{sub} = N_w$).

Word accuracy is usually considered the best measure of recognition performance.

2.1.5 Speech Understanding

The speech recognition process simply maps speech sounds to words. This is sufficient for automatic dictation applications; however, speech-enabled applications normally have to extract some kind of meaning from the words, i.e., to discover the concepts that the words represent. In other words, the goal is to build machines that can *understand* speech.

Gorin et al. [Gorin91] propose a formalization of the concept of understanding in terms of an operational definition. For any particular task, the goal is to map input messages into meaningful action. The set of all possible input messages we call language, and the mapping we call understanding. Research literature on speech understanding basically describes the various means that have been devised to implement this mapping.

For measuring speech understanding performance, it is more appropriate to use *concept accuracy* rather than word or sentence accuracy. This means comparing the output of the mapping implemented by the speech understanding system against a reference answer. Interestingly enough, Boros et al. [Boros96] observed a nearly linear relationship between word accuracy and concept accuracy.

Chapter 3 of this dissertation describes a semantic model that extends the notion of understanding to cover multimodal input messages which may include speech and other modalities; therefore, a survey of speech understand techniques is highly relevant. Some materials in the following discussion are from [Waibel90], [Gorin91], and [Gorin94].

Wordspotting

One way of extracting information from a spoken utterance is to detect the presence of certain keywords, either by examining the text output of a normal speech recognizer or

by running a special recognizer—called a *wordspotter*—that attempts to classify only the words in its vocabulary, skipping over out-of-vocabulary speech.

Many research papers on wordspotting concentrate on the algorithms for detecting and locating keywords in speech rather than on the application to speech understanding [Myers80][Higgins85][Wilpon90][Zeppenfeld93]. A speech understanding system based on wordspotting was proposed in [Newell71] but not actually constructed. More recent efforts are reported in [Rose91][Rohlicek92][Tsuboi92].

A variant of wordspotting is key-phrase spotting [Kawahara97]. A related approach is described in [Nagai94] where a concept represented by several phrases is a unit of semantic interpretation, and a spontaneously spoken sentence is regarded as a sequence of concepts. This bears some resemblance to the action frame and parameter slot model of multimodal semantic interpretation described in Chapter 3.

Syntactic/Semantic Parsing

The traditional approach to natural language understanding involves parsing text input using a grammar. Because spontaneous speech may contain ungrammatical fragments and speech recognition errors can only exacerbate this problem, much research has focused on developing robust parsing techniques that can skip over out-of-language fragments [Hayes81][Carbonell84][Ward91][Lavie93] or correct errors during parsing by transforming the input string [Wagner83].

Natural language parsers can use finite-state grammars [Lowerre80], context-free grammars [Brown94], recursive transition networks [Ward91], or augmented transition networks [Woods83]. Purely syntactic parsers build parse trees using phrase structure rules and thus require semantic interpretation rules that map syntactic non-terminals to semantic concepts for understanding [Thompson63]. Other approaches integrate syntactic and semantic constraints into a single grammar [Burton76][Brown94][Stahl96].

Learning the Mapping from Language to Meaning

Instead of writing grammars manually, it is possible to learn the mapping from language to meaning. Some systems do this by constructing grammar rules or transition networks [Anderson77][Vidal89]. Other systems train connectionist networks to implement the mapping [McClelland89][Gorin91]. [Tishby94] describes a method that exploit a statistical-algebraic dual structure to learn associations using very few examples.

Speech Understanding Systems

Some early speech understanding systems include Carnegie Mellon's Hearsay [Reddy73][Erman80] and Harpy [Lowerre80] as well as BBN's HWIM ("Hear What I Mean") [Woods83]. These were the products of the ARPA Speech Understanding Project initiated by the Defense Advanced Research Projects Agency. The Air Travel Information Service (ATIS) project resulted in a number of speech understanding systems for the airline reservation task, including Carnegie Mellon's ATIS system based on the Phoenix

grammar and robust parser [Ward91]. MIT's Voyager system is a speech understanding system for a tourist's city guide task [Zue90]. SRI's GEMINI system [Dowding93] is one of the natural language understanding components of the Open Agent Architecture (see section 2.4.4).

Strictly speaking, the JANUS speech translation system [Waibel91][Waibel96b] is not exactly a speech understanding system. However, JANUS translates between languages by converting sentences in the source language to an intermediate semantic representation called *interlingua* before generating sentences in the output language. The mapping from input sentences to their interlingua representations makes use of speech understanding techniques.

2.1.6 Speech Recognition APIs and Toolkits

Several Application Programming Interfaces (APIs) have been proposed to standardize the integration of speech recognition capabilities into application programs. These include Microsoft's Speech API [Rozak97], Sun's Java Speech API [Sun97], and SRAPI [SRAPI97] developed jointly by Novell, Dragon Systems, IBM, and other partners. The SpeechRecorder and SpeechRecognizer interfaces described in Chapter 4 do not represent an attempt to propose another speech API; rather, they are simplified interfaces that include only enough functionality to allow the proposed multimodal application framework to make use of speech recording and recognition capabilities. Implementations of the SpeechRecorder and SpeechRecognizer interfaces could rely on one of the established speech APIs, although this is not the case in the current version of the framework.

A number of research projects have focused on developing speech recognition toolkits which contain components that can be customized to construct a speech recognizer for a given application. This approach is more flexible than building a monolithic recognizer that may not satisfy the requirements of all speech applications. The same philosophy is behind the development of the multimodal application framework described in Chapter 4. Cambridge University's Hidden Markov Model Toolkit (HTK) is a library of components for building HMM-based systems [Young92]. The Oregon Graduate Institute developed the CSLU Speech Toolkit [Schalkwyk98] as well as a visual application builder that facilitates the rapid construction of telephone dialog applications using a visual programming paradigm. Carnegie Mellon's JANUS Recognition Toolkit (JRTk) [Finke97] implements powerful object-oriented speech recognition components within a flexible scripting shell.

2.2 Gesture and Handwriting Recognition

Handwriting recognition is another research topic that has received much attention in the last few decades. There are two ways to present data to a handwriting recognizer: in *off-line* systems, the written words are captured as static images; on the other hand, *on-line* systems capture dynamic information as the handwritten strokes are being traced by a digitizing device such as a tablet or a touch-sensitive screen. On-line recognition is more

relevant to this dissertation because the data streams from digitizing devices are usually available to multimodal applications supporting handwriting input.

The input to digitizing devices may represent more than simple handwritten words; the strokes can form shapes or symbols that carry significance beyond the letters of the alphabet. These shapes may represent *symbolic/iconic gestures* that convey meaning in an intuitive way (e.g., crossing out something to remove it) or as part of a specialized language (e.g., proofreader's symbols for editing a manuscript). They may also be used as *deictic gestures* that serve to indicate objects or regions of interest. Because the digitizing devices that capture hand-drawn strokes usually feature a stylus (although a fingertip works as well on some touch-sensitive screens), handwriting and gestures acquired this way are grouped under the designation of *pen-based inputs*.

2.2.1 Handwriting Recognition

Handwriting recognition features many difficulties similar to speech recognition. There are many different writing styles: block printing vs. cursive, isolated characters vs. continuous writing, etc., with continuous cursive writing recognition being the hardest problem because of segmentation difficulties and coarticulation effects, just as for continuous speech recognition. Writer dependence presents other difficulties parallel to speaker dependence. Vocabulary size may also significantly affect recognition performance.

[Govindaraju97] presents an overview of several paradigms that have been developed to recognize handwriting. [Tappert90] focuses on on-line recognition. Some materials from the following discussion are from [Govindaraju97] and [Rubine91].

Holistic vs. Analytical Recognition

These two broad paradigms parallel the two psychological theories of visual recognition of words. According to the *holistic* theory, words are identified directly from their global shapes; the *analytical* theory claims the opposing point of view that words are recognized by the identification of the constituent characters.

Holistic approaches to handwriting recognition usually extract global features from the word image and match them against features of words in a lexicon [Madhvanath96]. Features that have been studied include word length, number and direction of strokes, ascenders/descenders and holes, endpoints and crosspoints, etc. Lecolinet and Crettez [Lecolinet91] describe an interesting recognition method based on *graphemes*, which are "significant" visual structures in a word image. Simon and Baret's approach [Simon89] involves decomposing a cursive word into a pseudo-periodic (regular) signal modulated by non-periodic (irregular) signals.

Analytical approaches represent characters explicitly in terms of relationships among features. Some recognition algorithms perform segmentation and character recognition first, then use a lexicon to single out the best match by simple text string matching

[Favata92]. Other approaches use the lexicon to drive the segmentation and character recognition process [Schenkel94][Manke95][Govindaraju97].

Segmentation and Recognition Algorithms

Analytical recognition systems typically divide the whole recognition process into two distinct stages: the segmentation stage which locates character boundaries, and the character recognition stage which attempts to recognize each segmented character. An integrated approach is also possible where segmentation and character recognition are interwoven, typically using a dynamic time warping algorithm similar to speech recognition [Schenkel94][Manke95].

Many segmentation algorithms have been studied. [Eastwood97] describes a neural network based approach. [Lecolinet91] presents a method based on graphemes. Ke Han and Sethi [KeHan95] suggest a set of heuristic rules based on associations between certain geometric/topologic features and the English language characters.

Diverse recognition algorithms are available as well. [Guyon91] and [Manke95] describe neural network approaches based on the Time Delay Neural Network (TDNN) and the Multi-State Time Delay Neural Network (MS-TDNN), respectively. [Schenkel94] presents a method based on Hidden Markov Models (HMMs). [Gader94] considers applications of fuzzy set theory to handwriting recognition. A genetic algorithm that evolves an optimum match is described in [Menier94]. Other approaches include template matching [Kolzay71], statistical matching [Arakawa78], linguistic matching [Fu81], alignment of letter prototypes [Edelman90], contour analysis [Yamada96], and chain code [Kim97].

2.2.2 Pen-Based Gesture Recognition

Pen-based gesture recognition is very similar to handwriting recognition, except the gesture alphabet may be arbitrary rather than restricted to an alphabet of characters. [Rubine91] contains an excellent overview of gesture recognition methods, from which some materials in the following discussion were adapted.

Taxonomy of Gestures

[Milota95] presents a taxonomy of gestures (not necessarily pen-based gestures only) for use in multimodal interfaces.

Gestures are classified into one of three different types:

- *Arbitrary gestures* cannot be interpreted without being learned. They can be *referential* (referring to actions, objects, circumstances, etc.) or *modalizing* (referring more specifically to the individual's opinion).
- *Mimetic (imitative) gestures* are iconic in nature, so that an observer can deduce their meaning. They can be *analogical* (expressing the relationship

between the gesture and its referent) or *connotative* (using one of the secondary features of a referent to represent the whole).

- *Deitic (pointing) gestures* cannot be used without the referent being present in the situation in which the gesture occurs. They can be *specific* (pointing to a specific object), *generic* (pointing to a whole class of objects by pointing to an object in that class), or *mimetic deictic* (pointing with an additional motion that selects among objects, e.g., following a line to distinguish that line from other objects).

Some gestures can be considered *illustrative gestures*, which may be *deictic*, *spatiographic* (outlining the spatial configuration of the referent), *kinemimic* (outlining the action), or *pictomimic* (outlining the properties, e.g., big or small).

For the purpose of interacting with a computer interface, gestures can be classified according to semantic categories, namely

- *Manipulate* (re-orient)
- *Change* (correct, modify, replace by, undo)
- *Create or destroy*
- *Establish relationship*
- *Retrieve/store*
- *Name*
- *Confirm*

Gestures may also refer to attributes of objects, including

- *Intensity*
- *Direction*
- *Velocity*
- *Accuracy*
- *Size*
- *Orientation*
- *Location*

Recognition Algorithms

Template matching algorithms compare a given input template to one or more prototypical templates of each expected gesture type. [Lipscomb91] presents an interesting variation on template matching based on multiple templates at different resolutions.

The approach in [Newman79] uses dictionary lookup of zoning features, derived by dividing space into zones and representing input strokes by the zones they traverse.

A discrimination net (or decision tree) classifies inputs represented as feature vectors by testing features one by one using conditions at each node in a tree until a leaf node is reached. Coleman used a hand-crafted discrimination net [Coleman69] whereas Berthod and Maroy trained theirs from examples [Berthod79].

Statistical matching approaches derive classifiers from statistics such as average feature vector per class or per-class variances/correlations of the individual features. Examples can be found in [Hand82] and [Rubine91].

Linguistic matching is an application of automata and formal language theory to pattern recognition. An input gesture is represented by pattern primitives and composition operators expressing the relation between the primitives. This linguistic representation is then parsed using a grammar for each pattern class. [Fu81] describes a hybrid approach where statistical recognition is used to classify path segments and linguistic recognition is used to classify the pattern based on the relationships between the path segments. [Shaw70] presents an approach based on a picture description language (PDL).

Other gesture recognition approaches include neural networks [Hollan88] and Learning Vector Quantization (LVQ) [Shankar93].

Gesture-Based Systems

[Coleman69] describes a text editor that employs proofreader's symbols for editing commands. Buxton's musical score editor [Buxton85] supports simple gestures to indicate note duration and scoping operations. Margaret Minsky's Button Box [Minsky84] is a complete Logo programming environment that uses gestures for selection, movement, and path specification. [Rhyne86] presents a gesture-based spreadsheet program. [Murase88] describes a flowchart editor that recognizes hand-drawn flowchart symbols. Gesture-based drawing editors are described in [Kurtenbach91] and [Rubine91].

The above are applications that use only some form of pen-based gestures for input (possibly in addition to keyboard and mouse). Systems that support voice input as well are surveyed in section 2.4.3. It is interesting to note that from their descriptions, there appears to be no inherent difficulty in building the equivalents of these applications using the framework presented in this dissertation, as they all belong to the class of supported applications described in section 1.2.

2.2.3 Gesture-Enabled User Interface Toolkits

There is a plethora of user interface toolkits that facilitate the construction of graphical user interfaces (GUIs); however, the majority of these toolkits focus on the assemblage of widgets (interface components) and usually support a very restricted event handling model that does not have any provision for alternative input mechanisms other than

keyboard and mouse. To remedy this deficiency, several toolkits have been developed to incorporate gesture input as an integral part of the user interface.

Some examples of systems used to construct gesture-based interfaces are HITS from MCC [Hollan88] and Artkit from the University of Arizona [Henry90]. Dean Rubine built the GRANDMA ("Gesture Recognizers Automated in a Novel Direct Manipulation Architecture") toolkit as part of his Ph.D. thesis [Rubine91]. Bohm et al. [Bohm92] developed the GIVEN ("Gesture-driven Interactions in Virtual ENvironments") toolkit for 3D virtual interaction. Carnegie Mellon's Garnet system [Myers90] did not incorporate gestures from the start; however, its "interactor" mechanism was general enough to permit easy integration of gesture input [Landay93]. Amulet [Myers97], the successor to Garnet, retains this functionality.

2.3 Other Input Modalities

Besides speech and pen, several other input modalities have been studied in research literature. The following have been classified in section 1.2.1 as supported, if not as strongly supported as speech and pen, by the work presented in this dissertation.

Lip-reading is a method to improve speech recognition by analyzing lip movement, a visual information source tightly and synchronously coupled to the acoustic speech act. Lip movement during a spoken utterance consists of *visemes* [Jackson88], the counterpart of phonemes in acoustic speech. Acoustically confusable speech units are usually easily distinguishable visually, hence the usefulness of lip-reading. [Bregler93] describes an integrated acoustic/visual continuous speech recognition system based on the MS-TDNN. [Waibel96a] contains a survey of some other systems.

3-dimensional gestures are more general than 2-dimensional, pen-based gestures but much more difficult to process. [Fels90] describes a system that uses a Data Glove to control a speech synthesizer. [Sturman94] contains a survey of glove-based systems. 3D gestures can also be captured using hand modeling techniques [Rehg93][Kuch95].

Eye movement is another useful source of information because it usually serves to identify the focus of attention while a user is performing a task. [Jacob91] outlines some eye movement based interaction techniques. [Baluja94] describes a neural-network-based gaze tracker that does not require bulky, intrusive headgear. EagleEyes [Olivieri95] is a system that allows the user to control a multimedia interface by eye and head movements.

Lip-reading and gaze tracking algorithms may require a stable, constant-sized image of the user's face. Face tracking [Yang95] is a way to accomplish this. The face tracker described in [Yang95] isolates a face from the background of a video image using skin color distribution as well as shape and movement models. Face tracking also has applications in video conferencing. In addition, gaze direction information can be extracted from the face image using a face model [Rainer97].

2.4 Multimodal Human-Computer Interaction

This section surveys existing research on the combination of multiple input modalities in user interfaces. Most of the systems described here support speech input, some kind of gesture input, and possibly eye movement input.

2.4.1 General Discussions and Simulation Studies

The Wizard-of-Oz paradigm [Salber93] has become familiar in user studies of multimodal interaction. Using this approach, participants in a study are allowed to interact with a simulated multimodal system, and a hidden operator (the "Wizard") carries out the desired operations. This type of simulation study is very useful when a working system is not yet available and researchers want to obtain data about users' interaction patterns in order to build the actual system. The myriad multimodal simulation studies conducted over the years have yielded a great store of important knowledge about multimodal human-computer interaction. The results of many of these studies can be interpreted as providing favorable support for the paradigms underlying the multimodal framework at the core of this dissertation, as will be noted throughout this section.

Appeal of Multimodality

Many researchers have reported on the utility of combining input modalities.

Hauptmann [Hauptmann89] devised a simulation study for the combination of speech and gestures in a direct manipulation interface and found that most people strongly prefer combined inputs over speech or gestures alone.

Cohen et al. [Cohen89] discussed the benefits of the synergistic use of direct manipulation and natural language, pointing out that the combination overcomes limitations of each modality when used separately. Natural language excels at specifying objects and actions by description, whereas direct manipulation enables users to learn which objects and actions are available in the system. In addition, graphical rendering and manipulation of context yield a partial solution to the difficult problem of natural language anaphora.

Nakagawa et al. [Nakagawa94] evaluated a multimodal robot control simulation system and found that the combination of speech and touch screen input helped cover up the deficiencies of unimodal input, reduce the number of operational errors, and permit more kind of useful operations.

Nishimoto et al. [Nishimoto94] designed a multimodal drawing tool supporting speech, mouse, and keyboard, based on the principles of less physical movement, easiness to learn, and transparency of the system. They reported that in an evaluation with inexperienced users, the addition of speech input reduced average operation time to 71%, the number of commands to 77%, and the amount of mouse pointer movement to 53% compared to a keyboard-and-mouse only interface.

Oviatt et al. [Oviatt97b] identified performance difficulties with speech-only interaction in a map task and reported a strong user preference to interact multimodally.

Chu et al.'s study [Chu97] was not strictly a simulation because the participants were simply interviewed concerning the effectiveness of different interaction mechanisms; however, the results also indicated that most people would prefer to combine voice command, hand motion/gesture, and eye motion in a virtual reality based computer aided design system.

Based on the above studies, the work reported in this dissertation rests on the assumption that multimodal human-computer interaction is useful, and concentrates on the mechanisms of multimodal interface construction.

Media Integration Issues

Bellik [Bellik97] examined the technical problems encountered when designing multimodal interfaces and found that it would be necessary to take into account the temporal relationship of events, including the response time of input devices. The same sequence of speech and pointing events may result in entirely different interpretations depending on how they are grouped, and the paper contends that temporal proximity may indicate a high probability of co-references which would require merging input data from different devices. Temporal proximity is the default input grouping mechanism in the MMApp framework described in Chapter 4.

Oviatt et al. [Oviatt97a] reported that data collected in user studies contained enough information to extract temporal distribution parameters that could be used to fine-tune a temporal proximity model for input synchronization. Moreover, the results of the study revealed many insights into what kind of multimodal combinations are likely to be found in practice.

The authors of the study found that knowledge of the command type provided considerable predictive information about its likelihood of being expressed multimodally; in particular, spatial location commands were very likely to be composed multimodally because pen input excels in conveying location information and graphic rendering. At a semantic level, the spoken and written modes consistently contributed different and complementary (rather than redundant) information. In addition, the study also showed that 59% of all multimodal constructions did not contain any spoken deictic, only 25% contained a spoken deictic and a corresponding pen input that overlapped in time, and only 17% involved a simple point-to-speak pattern (drawn graphics and signs/symbols accounted for 76% of all pen input). The authors concluded that interpretation of spoken deictics via synchronous pointing (à la Put-That-There [Bolt80]) is unlikely to play a large role in handling the type of multimodal combinations found in practice. This result indicates that more flexible multimodal semantic models, such as the integration model described later in Chapter 3, are needed to reap the benefits of multimodality.

[Loken-Kim94], [Oviatt94a], [Oviatt94b], [Robbe96], and [Oviatt97a] all report that factors such as input modality, availability of different input means, task presentation

format, and user training have a significant influence over people's language and discourse patterns. These considerations may play an important role in the design of multimodal interfaces; however, the incorporation of these factors into multimodal designs is beyond the scope of this dissertation.

Taxonomy of Multimodal Interaction

Milota and Blattner's gesture taxonomy [Milota95] outlined in section 2.2.2 is actually presented in the context of multimodal interaction with gestures and voice. In addition to gesture categories, the taxonomy also classifies ways in which speech and gestures are combined:

- *Parallel* speech and gesture input
  - For economy of utterance
  - To illustrate spatial or visual attributes
  - To disambiguate an utterance
  - To provide redundancy
  - For emphasis
  - For organization
  - For confirmation
- *Alternating* speech and gesture input
  - To supply nonverbal equivalents
  - Because a needed word may be unknown
  - Because a time delay may have occurred

In addition, abstract relationships may be indicated by speech with examples given by gestures:

- Relation of attributes (e.g., bigger or smaller)
- Order (including hierarchies)
- Conditionals
- Categorization or abstraction (e.g., vehicle and car)
- Selection of a set of objects
- Aggregate operations (e.g., find the maximum)

Martin et al. [Martin97] proposed a theoretical framework for studying and designing multimodal interfaces around a definition of "multimodality" as "the cooperation between several modalities in order to improve the interaction." There are six basic "types of cooperation" between modalities:

- *Complementarity:* different chunks of information from the same command are transmitted over more than one modality;
- *Redundancy:* the same chunk of information is transmitted using more than one modality;
- *Equivalence:* a chunk of information may be transmitted using more than one modality;
- *Specialization:* a specific chunk of information is always transmitted using the same modality;
- *Concurrency:* independent chunks of information are transmitted using different modalities and overlap in time;
- *Transfer:* a chunk of information produced by one modality is analyzed by another modality.

Complementarity and redundancy are grouped together as "fusion." The cooperation types may be involved in several "goals of cooperation," comprising

- Adaptation to users;
- Adaptation to environment;
- Intuitiveness or faster learning;
- Fast interaction;
- Recognition and understanding.

Coutaz et al. [Coutaz95] proposed similar criteria for characterizing and assessing aspects of multimodal interaction under the umbrella of the *CARE* properties: *complementarity*, *assignment*, *redundancy*, and *equivalence*.

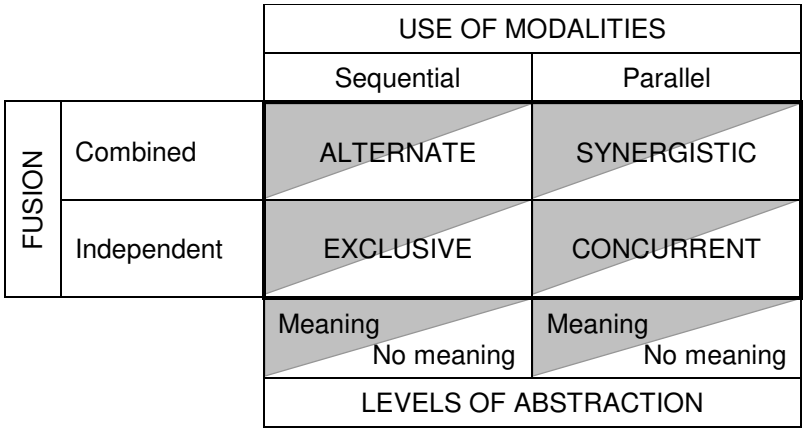


Figure 1. Nigay and Coutaz's Multimodal Design Space

Nigay and Coutaz [Nigay93] suggested a design space for multimodal systems in terms of concurrent processing and data fusion (see Figure 1). In this design space,

systems are classified along three dimensions: Levels of Abstraction (Meaning or No meaning), Use of Modalities (Sequential or Parallel), and Fusion (Combined or Independent).

A multimodal system can be described by a set of features (e.g., the commands its support) which are located in the design space and assigned a weight (e.g., frequency of use). The position of the whole system in the design space is the center of gravity of its features. The multimodal framework described in this dissertation strongly supports systems in the "Synergistic" category; however, the other three categories are also supported with suitable implementations of input grouping and interpretation policies.

Error Repair Strategies

In real-world situations, nothing works perfectly all the time. There are usually many possible causes when a multimodal application fails to do the right thing: recognition errors, interpretation errors, human errors, out-of-vocabulary words, or a combination of the above. Logically, there are three possible strategies when a failure occurs: ignore the error completely, let the user repeat or rephrase the failed command, or allow the user to correct the error. The first alternative is of course untenable in any practical applications. The second alternative is reasonable and easy to implement. The third alternative, allowing error repair, is more difficult to implement but also more easily acceptable to users.

McNair and Waibel [McNair94] described an ingenious method to let the user correct a speech recognition error by highlighting the erroneous words on the hypothesis display and respeaking only that part. The system dynamically adjusts the recognizer's language model according to the context of the repaired words and recomputes the N-best list of hypotheses.

Suhm [Suhm97] took a multimodal approach to error repair and developed an interface that allows users to correct errors using speech, spelling, editing gestures (e.g., cross to delete, caret to set insertion point, etc.), and handwriting. The combination of these modalities is quite powerful because strengths of one modality can compensate for weaknesses of another modality. For instance, easily confusable words may cause frequent speech recognition errors, but those words are usually distinguishable by spelling or handwriting.

2.4.2 Multimodal Integration Approaches

Early multimodal systems [Bolt80][Neal91][Koons93] were *speech-driven* in the sense of using speech as the main source of semantic contributions and limiting the other modalities (usually gestures or gaze) to simple deictics that disambiguate the speech input (although [Koons93] also describes a system that allows fully symbolic/iconic gestures instead of simple pointing).

To exploit the full potential of multimodal interaction, systems that support multiple input channels must be *fully multimodal* in that all elements of a command can be in any

modality capable of expressing them. This is supported by all the systems listed below as well as the semantic integration algorithm described in Chapter 3.

The Multimodal Definite Clause Grammar (MM-DCG) [Shimazu95] integrates multimodal inputs by allowing grammar rules to retrieve input symbols from any number of modalities. Data fusion requires unification of Prolog variables using Prolog predicates embedded inside grammar rules. The embedded predicates also play the role of semantic interpretation rules that map grammatical structures to application-specific semantics.

The semantic model proposed in this dissertation also leads to the development of a grammar language for multimodal input modeling. However, in contrast to the MM-DCG approach, grammars written in this modeling language does not govern the parsing of multimodal inputs according to grammar rules. Rather, the input interpretation algorithm is based on a connectionist network that can be trained from data, and a grammar-based input model serves only as a starting point to instantiate such a network.

The "melting-pot" fusion mechanism in PAC-Amodeus [Nigay95] is based on mapping each input event to a time-stamped set of "structural parts" or semantic slots, and fusing these melting-pot representations in a domain-independent way. The fusion of semantic information occurs at a higher level of representation than in the semantic integration approach described in Chapter 3, since individual input events are partially interpreted before the fusion.

[Vo96] describes a frame-merging approach that represents a much extended, domain-independent version of the approach outlined in [Vo93a]. Input from each modality is parsed and transformed into a semantic frame containing slots that specify command parameters. The information in these *partial frames* may be incomplete or ambiguous if not all elements of the command were expressed in a single modality. A domain-independent frame-merging algorithm combines the partial frames into a *complete frame* by selecting slot values from the partial frames to maximize a combined score. This is similar in some ways to the melting-pot algorithm described above.

[Johnston97] proposes a unification-based approach for multimodal integration. Multimodal inputs are transformed into typed feature structures that represent the semantic contributions of different modalities. A unification operation then combines these typed feature structures into a single typed feature structure that represents the interpretation.

The input integration approach proposed in this dissertation is similar to the above algorithms in the sense that there is a general and domain-independent data fusion mechanism underlying the input integration process; domain-dependent aspects come into play only in the manipulation of application-specific data types.

One important difference that distinguishes the algorithm described in Chapter 3 from the surveyed integration approaches is trainability. The connectionist network underlying the multimodal integration algorithm proposed herein can automatically learn the mapping from input messages to output actions given correctly classified input examples.

Furthermore, the network is capable of learning incrementally and improving its interpretation accuracy during actual use.

2.4.3 Actual Systems

No survey of multimodal systems would be complete without mentioning Bolt's classic Put-That-There system [Bolt80]. Bolt's paper demonstrated the utility and feasibility of incorporating pointing to disambiguate deictic references in spoken commands. However, since then many researchers have pointed out that multimodal interaction needs more flexibility to be successful. In particular, Oviatt et al. [Oviatt97a] described a corpus of user study data that would not benefit greatly from disambiguating spoken deictics via synchronous pointing.

CUBRICON [Neal91] is another system that uses pointing and natural language references to disambiguate one another when appropriate. It is more flexible than Put-That-There, being able to infer the intended referent of a point gesture that is inconsistent with the accompanying natural language input.

Koons et al. [Koons93] developed a multimodal interface that integrates simultaneous input from speech, gaze, and hand gestures (via a Data Glove). The system employs an interesting frame-based approach to perform data fusion. Data from each input stream is transformed into a set of time-stamped, interconnected frames that describe the structure of the incoming data. During evaluation of the frames, missing or ambiguous information triggers subgoals that attempts to find additional information from other modalities or to ask the user for it. The association of information from different modalities depends on temporal proximity determined by the time-stamps.

TAPAGE [Faure93] is a document editing system supporting voice and gesture. Voice input serves to specify states and actions as well as objects by description. Gesture input is used for drawing, writing, pointing out objects and positions, and producing gesture commands that trigger selection tools. The gesture recognizer is capable of recognizing horizontal and vertical strokes that compose a sketch of a table, and transforming the sketch into a perfectly regular table.

MultiksDial [Matsu'ura94] is a multimodal, keyword-based, spoken dialogue system equipped with multiple input channels including spontaneous speech and designation by touch, multiple output channels of graphics and voice responses, and sensors to detect the user's actions and plan interactive strategies. The system is the basis of a real-world application in a directory or map information guidance task.

HearingAid [Stoehr95] uses speech input to help resolve ambiguities and enhance its programming-by-demonstration paradigm.

MATIS (Multimodal Airline Travel Information System) [Nigay95] is a multimodal front-end for the ATIS task, supporting both individual and synergistic use of speech,

direct manipulation, keyboard, and mouse. The system is implemented in the PAC-Amodeus software architecture described below.

AlFresco [Stock93] is a multimodal interface to an image database of Fourteenth Century Italian frescoes and monuments. Users may refer to items by combining pointing with linguistic demonstratives.

The Olga dialogue system [Beskow96] integrates interactive spoken dialogue, 3D animated facial expressions, gestures, lip-synchronized audio-visual speech synthesis, and a graphical direct manipulation interface.

JEANIE [Vo96] is a multimodal appointment scheduler built on top of a previous version of the multimodal framework described herein. Users can employ spoken commands, symbolic gestures (e.g. arrow to move a meeting), handwriting (for names and meeting topics), or combinations thereof to schedule, cancel, and modify meetings on a computerized calendar. Inputs from multiple modalities can supply redundant information (e.g., cross out a meeting and say "Cancel this for me") or complementary information (e.g., draw a rectangle to specify the time and duration visually and say "Schedule a meeting with John about the budget"). Multimodal integration is based on a frame-merging approach (see 2.4.2 above).

MVIEWS [Cheyer97] is a system for annotating, indexing, extracting, and disseminating information from video streams for surveillance and intelligence applications. Users can speak and draw to add annotations, generate reports, collaborate with remote participants, and specify commands for object tracking, image processing, and setting alerts. The system is implemented within the Open Agent Architecture described below.

An interesting and unique application of multimodal interaction to the visual programming paradigm is Leopold et al.'s keyboardless visual programming interface [Leopold97]. The system supports voice, handwriting, and gesture inputs for entering and modifying expressions in Formulate, a form-based visual language.

2.4.4 Frameworks and Toolkits

Early multimodal systems were mostly *ad hoc* implementations, but in the last few years the idea of creating a reusable framework for multimodal applications has spread.

Kamio et al. [Kamio94] developed a rapid prototype system in the form of a user interface (UI) design support tool. Using the tool, developers design a multimodal UI visually by putting UI objects on a panel and establishing links between them to specify panel transitions that describe plan-goal scenarios (what to do when a certain event appears). The UI design support tool then generates a script that drives the MultiksDial multimodal interface (see section 2.4.3 above).

The toolkit described in Chapter 4 does not contain a visual interface builder. However, there is a visual grammar designer that allows application developers to construct a

model of the multimodal inputs an application expects, from which model customized software modules can be generated to instantiate a multimodal application.

The Multimodal Definite Clause Grammar (MM-DCG) [Shimazu95] is the first reported grammatical framework for multimodal interfaces. The major features of MM-DCG include the capability to handle an arbitrary number of modalities as well as temporal information in grammar rules. The authors contend that temporal information, such as input arriving time and the interval between two inputs, plays an important role in interpreting multimodal inputs; therefore, temporal information is tightly integrated into the grammar formulation in the form of time variables and time-out specifications. The data fusion mechanism of MM-DCG was discussed in 2.4.2. MM-DCG rules can be translated into Prolog predicates that parse multimodal inputs according to the grammar.

The application framework proposed in this dissertation does not rely on a grammar to parse multimodal inputs. Rather, the underlying semantic framework is a semantic model of multimodal integration by joint segmentation and alignment of parallel input streams. A multimodal grammar can be constructed using the provided toolkit but its purpose is to provide an analysis of input messages rather than to drive the semantic interpretation process (see section 3.1).

VisualMan [Wang95] is a device- and application-independent model of selection and manipulation using eye-gaze, voice, and manual response. The paper also describes a prototype user interface with eye-tracker, 3D controller, voice-key detector, and keyboard implemented using the VisualMan model to allow users to select and manipulate cubes in 2D and 3D spaces.

The PAC-Amodeus software architecture model and the generic “melting-pot” fusion mechanism [Nigay95] form a reusable global platform applicable to the software design and implementation of multimodal interactive systems. PAC-Amodeus defines the levels of abstraction appropriate for performing engineering tradeoffs such as setting the boundaries between the levels of abstraction. The core component—the Dialog Controller—is a set of cooperating agents that capture parallelism and information processing at multiple levels of abstraction. The melting-pot fusion mechanism was discussed in section 2.4.2.

The application framework proposed in Chapter 4 also specifies a system architecture for multimodal applications. The purpose of this system architecture is to capture a common application infrastructure that ties together many reusable components. As such, the proposed system architecture is probably more restrictive than PAC-Amodeus or the Open Agent Architecture described below; however, it is well integrated with the semantic model proposed in Chapter 3 and provides explicit interface specifications for the processing of speech and pen input modalities.

SRI’s Open Agent Architecture (OAA) [Moran97] provides access to agent-based applications through intelligent, cooperative, distributed, and multimodal agent-based user interfaces. The system supports the creation of applications from agents that were *not* designed to work together, thereby facilitating wider reuse. The supported modalities

include handwriting, gesture, and spoken language in addition to the traditional graphical user interface modalities.

A rich set of multimodal interactions in the OAA is implemented by the Modality Coordination (MC) agent. It is responsible for combining the inputs in different modalities by resolving references, filling in missing information for an incoming request, and resolving ambiguities using context, equivalence, or redundancy. The MC agent is equivalent to a combination of input coordinator and semantic integrator in the MMApp framework of Chapter 4.

The architectures surveyed above represent quite general multimodal frameworks that have been successfully employed to construct several multimodal applications. However, the framework and toolkit proposed in this dissertation still have their *raison d'être*. One distinguishing feature is the multimodal semantic model that serves as a unifying theme tying together all the components and providing uniform support for all input modalities alike. Another feature that seems to be lacking in software architectures such as PAC-Amodeus and OAA is the rapid prototyping of multimodal applications, which the design process and supporting toolkit in Chapter 5 address in detail.

Chapter 3
SEMANTIC INTEGRATION OF MULTIMODAL INPUTS

This chapter describes a semantic model for a broad class of multimodal applications. This model is based on the alignment and joint segmentation of input streams from multiple modalities. Combining this model with a context-free grammar formulation gives rise to the *Multimodal Grammar Language* (MMGL), an approach to modeling multimodal inputs for the purpose of prototyping an application. From the semantic model, a connectionist network based on mutual information has been developed to produce an input alignment/segmentation that maximizes an *a posteriori* probability score.

3.1 Input Modeling and Interpretation

In the absence of extensive input data collected for a given task domain, an application developer may have to construct a model of what the user might say, write, draw, etc. in order to implement a prototype of the application. Even if data is available (e.g., to train the speech recognizer), the application developer may want to create a semantic model that describes how the application would respond to each set of multimodal stimuli drawn from the collected data. When the application is implemented and deployed, it has to interpret user inputs, i.e., map the inputs to appropriate responses. Thus, the application developer is faced with two distinct but related tasks: *input modeling* and *input interpretation*. Let us define these two concepts more precisely.

Given the set of all possible input messages for a particular task and the desired responses, we define *input interpretation* as the mapping from input messages to corresponding responses. This is modeled after the operational definition of understanding in [Gorin91]. The desired response for a particular input message is its *semantic value*.

A *input model* is a set of couples $\langle I,S \rangle$ where I is an input message and S is its semantic value. The purpose of such a model is usually to describe the set of input messages that the application is most likely to encounter in practice, so not all possible input messages may be included. It is also possible to associate the input model with a probability distribution function that specifies the likelihood of encountering a particular input message.

An *interpretation algorithm* is a function that maps input messages to semantic values. The *accuracy* of such an algorithm over a set of input messages is the fraction of semantic value assignments that match the predefined correct mapping.

Typical speech-enabled applications employ a grammar-based approach that intermixes input modeling and interpretation. The input model is a grammar that describes the syntax of spoken inputs, plus a mapping that assigns semantic tags to the syntactic elements of the grammar. The interpretation algorithm is a parser that matches each input utterance against grammar rules to break it down into components and map them to semantic tags, which are assembled to form the semantic value of the input utterance. Because the interpretation algorithm is driven by the input model (i.e., the grammar), it cannot generalize beyond the set of input messages described by the input model. The grammar must therefore be written to *maximize coverage*, i.e., to parse correctly all the input messages that the application may encounter. The more accurate and flexible the grammar, the larger and unwieldy the input model becomes.

The limitations of the grammar language (e.g., the context-free nature of some grammars) may also drive the flexibility of the grammar and the accuracy of the input model in opposite directions. Consider the following (admittedly contrived) example:

```
ACTION ::= VERB OBJECT
VERB   ::= buy | sell | call
OBJECT ::= coat | sofa | boss
```

(The `::=` symbol indicates a grammar production rule that expands the left hand side to one of the alternatives—separated by the `|` symbol—on the right hand side. Non-terminal symbols are in uppercase; terminal symbols are in lowercase.)

This grammar is very flexible in the sense that you can add coverage for other verbs and objects simply by adding words to the appropriate lines, without changing anything in the parsing algorithm. The phrases that the user might say to the application, such as "buy coat," "sell sofa," "call boss," etc. can all be parsed. However, if used as an input model, the grammar would also admit phrases like "buy boss," "call coat," etc. which are extremely unlikely to occur in practice. If one purpose of the input model is the generation of example input messages to evaluate an application prototype, the flexibility of the grammar is detrimental to the accuracy of the evaluation.

It follows from the above discussion that some separation between modeling and interpreting user input is beneficial. The interpretation algorithm may use the input model as a data source and a starting point for parsing input messages, but should not rely on the input model as the only data source. In other words, the interpretation algorithm should be able to generalize from the input model and produce reasonably accurate interpretation even for input messages not explicitly coded into the input model. It is also clear that we need to generalize the notion of grammar-based input modeling to encompass more than just the speech modality.

Section 3.2 below describes a semantic model for multimodal interpretation. A grammar-based approach to modeling multimodal input derived from this semantic model is presented in section 3.3. A multimodal interpretation algorithm based on an information-theoretic connectionist network is described later in section 3.4.

3.2 A Multimodal Semantic Model

The previous section defines input interpretation as the mapping between input messages and their semantic values—desired responses from the application. This section refines this notion and extends it to cover multimodal inputs by answering two questions:

1) What constitutes an input message in a multimodal application?

2) What is the semantic value of a multimodal input message?

3.2.1 Combination of Multimodal Input Signals

As mentioned in section 1.2.1, most of the conceivable input modalities used in human communication can be interpreted as *information streams*. The use of the word "stream" implies a sequence of data packets or *tokens*, which may be words and phrases in spoken or written modalities, shapes in gestures, or eye fixations and saccades in gaze, etc. Representing input data as a stream reduces 2- or 3-dimensional inputs (e.g., pen-based or 3D gestures) to a single temporal dimension. Each input stream is one-dimensional but the combination of multiple streams gives rise to multidimensionality.

Let us consider the notion of information stream more formally. The tokens in a stream are delivered sequentially by an *information source* in the classical sense of information theory. That is, the token sequence carries information contents both within the input channel and with respect to the output space of all possible semantic values.

The within-channel information content can be characterized by several measures. If the value v_n of the token at time n is regarded as drawn from a random variable, the information content of that one token is the number of bits carried by the token, given by

$$i(v_n) = -\log_2 P(v_n)$$

Given a token with value v_n at time n, the probability distribution of the next token (represented by the random variable V_{n+1}) may be influence by v_n. The information about the distribution of V_{n+1} given v_n can be measured by

$$J(v_n, V_{n+1}) = \sum_{v \in V_{n+1}} P(v \mid v_n) \log_2 \frac{P(v \mid v_n)}{P(v)}$$

shown in [Blachman68] to be the unique non-negative measure of how much information a value of one random variable provides about a second one.

If two consecutive tokens are grouped into a *fragment* (also called a *segment*), we can measure the mutual information of the tokens in the fragment:

$$I(v_n, v_{n+1}) = \log_2 \frac{P(v_{n+1} \mid v_n)}{P(v_{n+1})}$$

We can also treat fragments as "macro tokens" and compute their information content measures as we did for simple tokens.

If input messages composed of tokens are assigned output semantic values, the information about the output provided by a single input token is

$$J(v_n, S) = \sum_{s \in S} P(s \mid v_n) \log_2 \frac{P(s \mid v_n)}{P(s)}$$

where S is the random variable representing the output semantic value. We can also measure the information provided by fragments composed of multiple tokens.

When multiple input channels are involved, we can group tokens from different modalities into cross-channel fragments and compute their information contents with respect to the output space. This leads to the notion of input alignment and joint segmentation that will be explored further in section 3.2.2.

In summary, the token streams delivered by the input sources carry information that contributes to the selection of an appropriate semantic value to assign to the input, hence they are termed information streams.

Unimodal and Multimodal Input Events

The token sequence in each unimodal input stream is partitioned into *input events* possibly separated by periods of inactivity (i.e., absence of meaningful data). What constitutes an input event in the unimodal stream is application and modality dependent. Consider speech as an example. Data coming over the speech channel may be divided into utterances based on periods of silence or prosody information such as a drop in pitch at the end of a sentence. In some applications it suffices to consider each utterance a speech input event. In other applications, one or more consecutive utterances may form a *speech act* that is assigned a meaning; in that case input events are complete speech acts.

We can now define a *multimodal input event* as a group of one or more unimodal input events that are jointly interpreted, i.e., assigned a single semantic value. Thus what we called input messages in section 3.1 are termed multimodal input events in multimodal applications. They are the elements in the domain of the mapping we call multimodal input interpretation.

Input Event Partitioning and Grouping Policies

As previously discussed, acceptable policies of how a unimodal input stream is partitioned into input events depend on the application and the modality. In addition, the above definition of multimodal input event leaves unspecified the policy that determines when to group unimodal input events into a combined multimodal input event. We can select particular policies on a per-application basis, but it is also possible to employ reasonable application-independent policies.

In the applications presented in my previous works [Vo93a][Vo95][Vo96], speech and pen input events start when the user begins speaking or drawing, and end when no input signal is detected within a predefined time-out interval[†] (this is similar to turn-taking in dialogs). Input events from different modalities are grouped if they occur close together in time; i.e., if they overlap or if one event starts within a time-out interval after another event ends (see Figure 2). Oviatt et al. [Oviatt97a] show that it is possible to estimate typical time lags between sequential input events that should be jointly interpreted, based on data collected in user studies. It should also be possible to estimate turn-taking pauses similarly, although I know of no published research on that topic. The time-out parameters can thus be determined empirically.

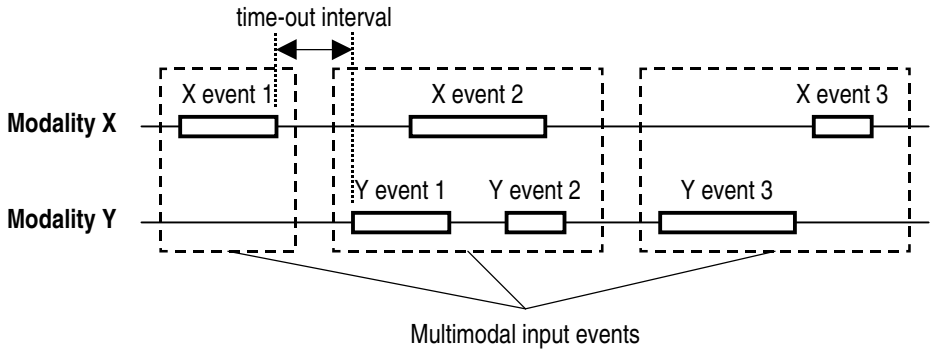


Figure 2. Input Grouping by Temporal Proximity

In some applications, an entirely different input grouping policy may be more suitable. For instance, in certain domains it may be feasible to perform a preliminary semantic analysis of the input data, determine whether pieces of information from different modalities fit together, and group input events accordingly. In some situations it may be preferable to let the user decide when the application should start interpreting all the inputs accumulated so far (e.g., by pressing a "Go" button after speaking or drawing).

Among the input grouping policies outlined above, the temporal proximity model is fairly application independent. Furthermore, experience with previously implemented multimodal applications suggests that this model is easily accepted by users because it feels quite intuitive. Accordingly, the temporal proximity model is the default input grouping policy in the application framework of Chapter 4, although application developers are of course free to substitute different policies depending on the target applications.

[†] The motivation for this scheme originated from observations made during the demonstrations of early systems. When a handwriting recognizer was demonstrated, visitors who tried the system were asked to press a "Recognize" button after writing a word, but most of the time they would stop writing and then invariably waited for the system to do something. The same thing happened during the demonstration of a speech translation system.

Different Levels of Input Fusion

In deriving a semantic value for a multimodal input event, the input data from different modalities must be combined in some way. This input fusion may happen at different levels:

1) *Fusion of raw signals*. This is the lowest level of fusion because a semantic value is derived from the combination of unprocessed signals such as speech sounds and pen coordinates. This is difficult at best and does not present any obvious way of describing semantic mappings in a general and domain-independent manner; hence it will not be explored further in this dissertation.

2) *Fusion of partial interpretations*. This is the highest level of fusion because each modality is interpreted separately to produce a (possibly incomplete) semantic value, and these partial results are then merged to yield the complete interpretation. [Vo96] describes an approach that employs semantic frames to represents partial interpretations that are combined using a frame-merging algorithm. This semantic representation was in fact the inspiration for the semantic model described in this chapter. Beyond that, high-level fusion is not covered in this dissertation.

3) *Fusion of intermediate symbolic representations*. This is an intermediate level because the raw signals are converted to a more convenient representation (such as a text string for speech input or a sequence of gesture shapes for pen input) before being combined and interpreted together. Thus the total interpretation process is split into a unimodal recognition stage (e.g., speech and handwriting recognition) and a semantic assignment stage. The multimodal interpretation algorithm described later in section 3.4 works at this level of input fusion.

4) *Hybrid multi-level fusion*. Speech and pen recognition systems usually exploit within-channel information content to improve accuracy by predicting subsequent symbols based on previous symbols in the data sequence (one example is the bigram/trigram language models described in section 2.1.3). Given two input channels, say speech and pen, one can imagine that knowing what the user said may constrain the kind of pen gestures one expects, and vice versa; i.e., there is cross-channel information content as well. The interdependence between the channels suggests that a joint language model of some kind should improve recognition accuracy beyond single-channel language models. This joint language model must obviously be governed by the semantics of the multimodal inputs, hence input fusion is distributed between the low level of raw signals and the intermediate or high level of semantic assignment. How this can be accomplished is an open question left to future research.

3.2.2 Meaning of Multimodal Input Events

Section 3.1 defines input interpretation as the mapping between input messages and desired responses from the application. As discussed in section 1.2.2, the applications targeted by the research presented in this dissertation are those that interpret user inputs as commands or *actions* to perform. The user is trying to accomplish a task, and if the application carries out the right action then we are justified in saying that it has successfully interpreted whatever the user said, wrote, drew, etc. The available actions should also accept *parameters* that could cause actions with the same name to have different effects. For instance, a *Delete* action in a word processor would not be meaningful unless we specify what to delete by supplying some kind of parameters.

Accordingly, we define *multimodal input interpretation* as the mapping from multimodal input events (defined in section 3.2.1 above) to corresponding *parameterized actions* that the application should perform in response to the input events; i.e., the semantic values of multimodal input events are parameterized actions.

Action Frames and Parameter Slots

As previously discussed, the input from each modality can be represented as an information stream consisting of a sequence of tokens which may contribute information towards determining the output action and its parameters. A multimodal input event can be regarded as a set of parallel streams that can be aligned and jointly segmented such that each part of the segmented input influences part of the interpretation (see Figure 3). The overall semantic value is called an *action frame* because it specifies the action to be carried out in response to the input. Each part of the segmented input is a *parameter slot* that specifies one action parameter. The input segments in each parameter slot should contain enough information to determine the value of the corresponding parameter.

Another way to view the alignment and joint segmentation process is to regard the combination of input segments in each parameter slot as a *cross-channel fragment* in the information streams, as briefly mentioned in section 3.2.1. The information content of the cross-channel fragment with respect to the output space of semantic values determines what parameter slot is the best label for the fragment.

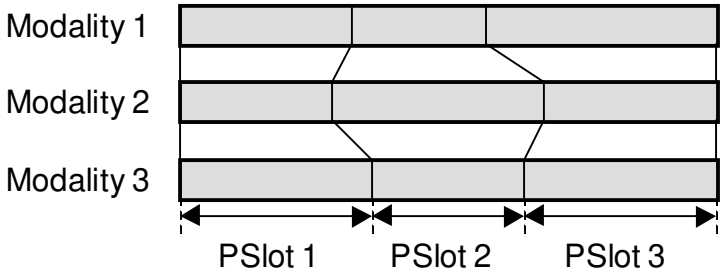
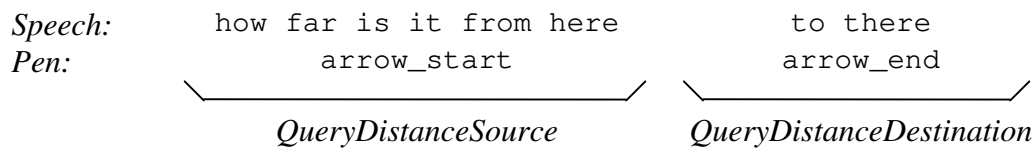


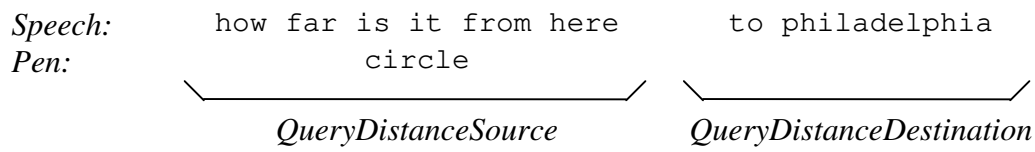
Figure 3. Alignment and Joint Segmentation of Multimodal Inputs

Consider the following illustrative example. Suppose we have a map navigation system that allows the user to ask for information by speaking and drawing on the screen.

The user might say “How far is it from here to there?” while drawing an arrow between two points on the displayed map. The speech input stream consists of the words in the utterance whereas the pen input stream contains a pair of *arrow_start* and *arrow_end* tokens. The interpretation of this input combination is a *QueryDistance* action frame containing a *QueryDistanceSource* parameter slot followed by a *QueryDistanceDestination* parameter slot. The input streams are segmented and aligned as follows:



If the destination point is somewhere outside the displayed area, the user might say “How far is it from here to Philadelphia?” and circle the starting point instead. In this case the input segmentation becomes



For the utterance “How far is it from Pittsburgh to Philadelphia?” the parameter slots would consist of speech segments only.

It should be noted that the parameter slots in Figure 3 do not have to be all distinct. It is obvious that adjacent parameter slots in a sequence (PSlot1 and PSlot2, or PSlot2 and PSlot3 in Figure 3) must be different, but PSlot1 and PSlot3 may be the same for instance. This accommodates the fact that pieces of information about the same parameter may be separated by data relevant to another parameter.

Selection of Parameter Slots

Another way to formulate the input segmentation in the above example is

Action: how far is it
Source: from <somewhere>
Destination: to <somewhere>

The advantage of this formulation is the orthogonal partitioning of the parameter space. Any number of different questions that involve a pair of source and destination points on the map (“how long does it take to drive,” “how many gas stations on the way,” “are there any road blocks along the way,” to name a few) result in the same general segmentation, where only the *Action* slot is different for each question.

The above is a perfectly valid parameter selection strategy. However, in the applications I developed using the design process presented in this dissertation, I have tended to follow the pattern of the *QueryDistance* example above. In this scheme, the action and its parameters are more tightly bound, such that the sequence of parameter slots implicitly

determines the action (e.g., *QueryDistanceSource* + *QueryDistanceDestination* = *QueryDistance*). A different question involving a pair of source and destination points, e.g., "how long does it take to drive," would necessitate a different parameter slot sequence such as *QueryTimeSource* + *QueryTimeDestination* = *QueryTime*. The structure of the two questions are very similar, hence the same input segment may have to be classified under *QueryTimeSource* in one input event and under *QueryDistanceSource* in another input event, depending on the context. This is not a problem because input tokens such as "how far" and "how long" carry enough information content to distinguish the two cases, and once the source (or destination) segment is correctly classified, the information content of this classification helps constrain the value of the other segment.

The reason for this parameter selection method concerns the accuracy of the multimodal interpretation algorithm described later in section 3.4. In my experiments, the orthogonal parameter selection method resulted in much lower interpretation accuracy. I believe that orthogonal parameter selection, although theoretically elegant, can create input-output associations that are too complex and subtle for my interpretation algorithm to learn. This cannot be seen clearly in the above example because it is too simple, and from a human point of view there is no obvious reason why it would be difficult to learn mappings such as "how far is it" → *Action* and "from <somewhere>" → *Source*. However, in practice, the great variety of phrasing and word order frequently causes important keywords to be associated with too many different contexts so that the interpretation algorithm cannot learn the salient associations. The tighter action/parameters binding in effect "cheats" to overcome this limitation of the interpretation algorithm by providing more information and imposing more constraints to increase accuracy.

In summary, the semantic model presented here supports all parameter selection strategies, but the design process based on the algorithm in section 3.4 more strongly supports the strategy illustrated in the *QueryDistance* example, with tightly bound actions and parameters.

Semantic vs. Temporal Alignment

Note that the alignment and segmentation process described above makes no mention of any temporal constraints. (Even if the temporal proximity policy is used, it only affects the partitioning and grouping of input events; the semantic alignment and segmentation process is applied to multimodal input events after they have been partitioned and grouped according to any policy.) The various parts of the input streams are aligned purely based on their semantic contents. This way there is no time-alignment constraints on the way the user interacts with the system.

It is possible to force some kind of temporal alignment based on time-stamps assigned to input tokens. However, if this is done without regards to semantics, we may end up with cross-modal combinations that do not make sense, or we may have to impose constraints on the way the user interacts with the system (e.g., a gesture must be drawn right at the time certain words are spoken). This would severely compromise the flexibility of the user interface and undermine the justification for multimodal support.

Some multimodal systems [Bolt80][Koons93] rely on time-stamps to align co-referents from different modalities (e.g., spoken deictics and pointing gestures). However, user studies such as the one reported in [Oviatt97a] have shown that when users are free to interact multimodally with computers in any way they want, most of the time temporal alignment will fail because of missing spoken deictics or lack of time overlap between co-referents.

If temporal alignment is used at all, it should be used only to constrain the semantic alignment by imposing restrictions on the possible locations of alignment boundaries according to time-stamps[†]. This would be properly part of the interpretation process and thus irrelevant in the semantic modeling stage.

3.3 Multimodal Input Modeling

As defined in section 3.1, an input model is a set of couples $\langle I,S \rangle$ specifying a set of input messages and their associated semantic values. Usually it is not feasible to list explicitly all the input messages covered by the model; rather, the set of input messages is often defined implicitly by a set of rules, e.g., using a grammar.

3.3.1 Traditional Grammar Formulations

The most popular type of grammar is the *context-free grammar* (CFG). A CFG consists of a set of *grammar production rules* that expand a *non-terminal* symbol to a sequence of other symbols. *Terminal* symbols are not expanded further. Several rules may expand the same symbol to different *alternatives*. An input sentence (i.e., a sequence of input tokens) matches a grammar symbol if the sentence can be produced from the symbol by applying a series of grammar production rules. The grammar is termed context-free because the expansion of a symbol does not depend on its context (i.e., where the symbol occurs in a sequence or what its relationship is to surrounding symbols).

A formulation equivalent to the CFG is the *recursive transition network* (RTN). An RTN consists of nodes connected by labeled arcs. The arc labels may be terminal symbols or names of other RTNs. Traversing an arc labeled with the name of an RTN is similar to calling a subroutine in a programming language in that the new RTN is traversed in turn before traversal resumes in the calling RTN. An input sentence is covered by an RTN if it can be produced by traversing the RTN from a start node to a stop node and outputting terminal labels along the way.

[†] It should be possible to learn the constraint parameters by amassing statistics from user data in a way similar to how Oviatt et al. estimated temporal proximity parameters [Oviatt97a].

3.3.2 The Multimodal Grammar Language

There are three deficiencies in the CFG and RTN formulations that preclude their use for multimodal input modeling, at least without modifications:

- There is no explicit modeling of input symbols from different modalities and how they are combined in each input message.
- There is no explicit modeling of semantic value assignment. Sometimes the non-terminal symbols represent a mixture of syntactic and semantic categories. Often a separate mapping must be added to convert syntactic parse trees generated by the grammar to semantic values.
- There is no probabilistic model that specifies the relative likelihood of input messages covered by the grammar. This deficiency is easily remedied by assigning probabilities to alternative expansions in grammar production rules (or alternative arcs emanating from the same RTN node).

The deficiencies of the CFG and RTN in modeling multimodal inputs can be overcome by embedding the CFG formulation into the multimodal semantic model described in section 3.2 above. By introducing action frames and parameter slots into the grammar and modeling multimodal input combinations at the parameter slot level (as illustrated in Figure 3 on page 36), we achieve two goals at once: introducing the notion of multimodality and specifying the semantics of the input messages.

The result of embedding the CFG inside the multimodal semantic model is the *Multimodal Grammar Language* (MMGL).

Multimodal Grammar Structure

An MMGL model is a grammar structure consisting of *nodes* and *sequences*. A node plays the same role as a terminal or non-terminal symbol in the CFG. A sequence is an ordered set of zero or more nodes. Each sequence is also labeled with a positive *weight*.

There are six types of nodes:

- A Toplevel represents an entire input model and contains one or more sequences, each of which contains exactly one AFrame;
- An AFrame represents an action frame and contains one or more sequences, each of which consists of one or more PSlots;
- A PSlot represents a parameter slot and contains one or more UnimodalNodes (at most one for each input modality);
- A UnimodalNode specifies a sub-grammar for a single input modality and has the same structure as a NonTerm, with the addition of a label specifying the modality;
- A NonTerm is a non-terminal node consisting of one or more sequences, each of which contains zero or more NonTerms or Literals;

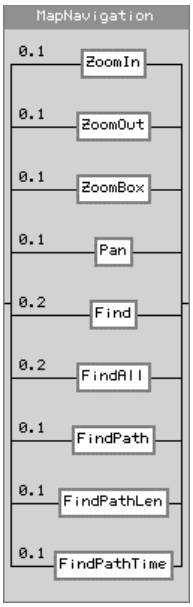
- A Literal is a terminal node containing a text string representing one or more input tokens.

Figure 4 on page 42 shows grammar excerpts (from the design example of the QUICKTOUR map application in Chapter 6) that illustrate all the basic building blocks. Section 5.2 describes how the MMGL grammar structure can be implemented using an object-oriented programming language such as Java.

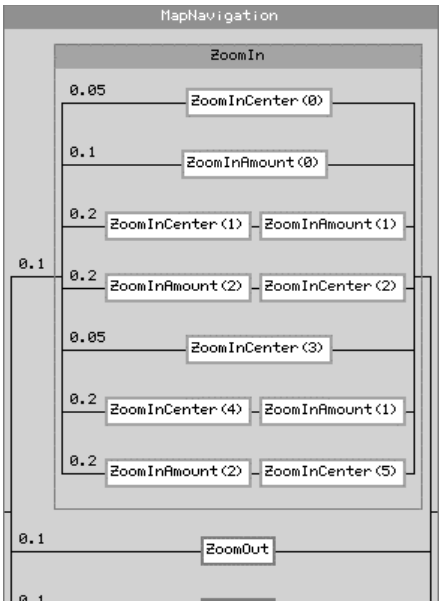
It is easy to see that the NonTerm and Literal nodes compose a structure equivalent to a CFG. Grammar production rules are represented by the expansion of NonTerms into their constituent sequences of sub-nodes.

Multimodal semantic values are modeled by the AFrame and PSlot nodes. Defining a PSlot as a set of UnimodalNodes for distinct input modalities models the alignment and joint segmentation of multimodal inputs into parameter slots (see section 3.2.2). Different PSlot nodes may represent the same parameter slot in the context of different input messages; in this case they are assigned the same name with different numeric tags (e.g., *ZoomInCenter(0)* and *ZoomInCenter(1)* in Figure 4) to distinguish the instances of the parameter slots.

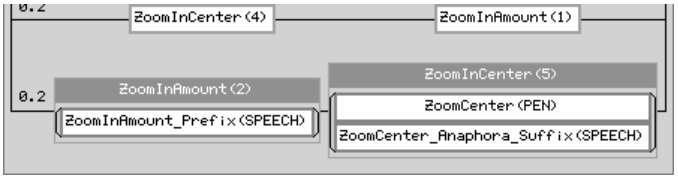
The sequence weights define a probability distribution that specifies the relative likelihood of multimodal input combinations modeled by an MMGL grammar. The probability that a node is expanded to one of its constituent sequences is the weight of that sequence divided by the total weight of all the sequences that constitute the node.



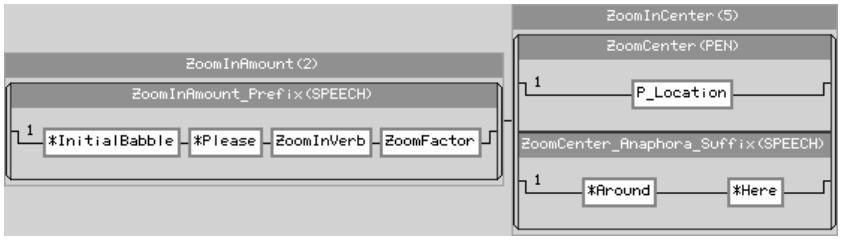
a) Toplevel *"MapNavigation"*



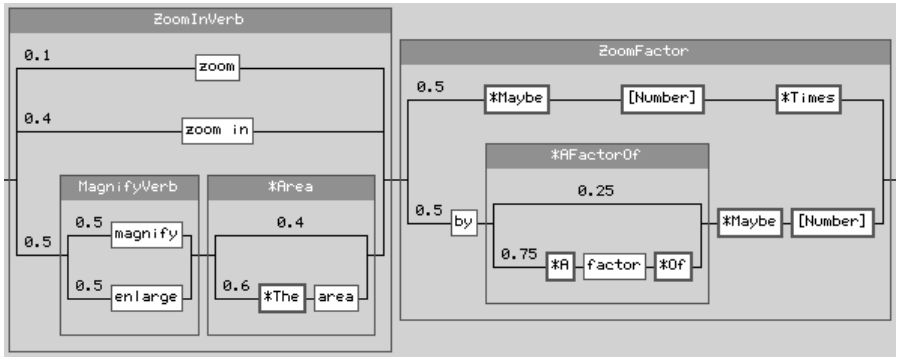
b) AFrame *"ZoomIn"*



c) PSlots *"ZoomInAmount"* and *"ZoomInCenter"*



d) UnimodalNodes in *"ZoomIn"* PSlots



e) NonTerms and Literals

Figure 4. Multimodal Grammar Structure

3.4 Multimodal Input Integration

Given the semantic model described above, the first step in interpreting a multimodal input event is to find an alignment and joint segmentation of the input streams. This *input integration* process produces input segments labeled as parameter slots, which can be postprocessed to extract the actual action parameters. The parameter-extraction postprocessing is necessarily domain-dependent, but it is possible to devise domain-independent algorithms to integrate multimodal input streams. This section describes such an algorithm based on a connectionist network.

3.4.1 Basic Mutual Information Network

Suppose we have a sequence of input tokens $t_m, m = 1 \ldots M$, that is to be associated with one of several output classes $c_n, n = 1 \ldots N$. It is reasonable to select the *maximum a posteriori* (MAP) hypothesis, or the output class having the greatest *a posteriori* probability given the input:

$$\begin{aligned} c_{MAP} &= \underset{c_n}{\operatorname{argmax}} P(c_n \mid t_1 t_2 \ldots t_M) \\ &= \underset{c_n}{\operatorname{argmax}} \frac{P(t_1 t_2 \ldots t_M \mid c_n) P(c_n)}{P(t_1 t_2 \ldots t_M)} \end{aligned} \tag{1}$$

The second line follows from Bayes' theorem. If we make the simplifying assumption that the input tokens are independent as well as conditionally independent given the target output, i.e.

$$P(t_1 t_2 \ldots t_M) = \prod_{m=1}^{M} P(t_m)$$

$$P(t_1 t_2 \ldots t_M \mid c_n) = \prod_{m=1}^{M} P(t_m \mid c_n)$$

then it follows from Equation (1) that

$$c_{MAP} = \underset{c_n}{\operatorname{argmax}} P(c_n) \prod_{m=1}^{M} \frac{P(t_m \mid c_n)}{P(t_m)} \tag{2}$$

This is a Bayesian classifier [Mitchell97] applied to a "bag of words" model, meaning that the input is considered an unordered collection of independent words. Since the logarithm function is monotonically increasing, $f(x)$ and $\log_2 f(x)$ reach their respective maximum values at the same x value for all $f(x)$; thus

$$\begin{aligned} c_{MAP} &= \underset{c_n}{\operatorname{argmax}} \left(\log_2 P(c_n) + \sum_{m=1}^{M} \log_2 \frac{P(t_m \mid c_n)}{P(t_m)} \right) \\ &= \underset{c_n}{\operatorname{argmax}} \left(\log_2 P(c_n) + \sum_{m=1}^{M} I(t_m, c_n) \right) \end{aligned} \tag{3}$$

where $I(t_m, c_n) = \log_2[P(t_m \mid c_n)/P(t_m)]$ is the *mutual information* of input token t_m and output class c_n.

The right hand side of Equation (3) can be implemented by a connectionist network.

A connectionist network is a computational structure that relies on massively connected simple processors to perform complex computations. The network architecture presented here consists of a number of *input units* connected to many *output units*. Each unit (input or output) has an *activation level*. The activation of output unit c_n is a weighted sum of input unit activation levels, where an input unit t_m has activation 1 if the token t_m is present in the input sequence, and 0 otherwise. The connection weight from input t_m to output c_n is $w_{mn} = I(t_m, c_n)$. There is also a bias connection with weight $w_n = \log_2 P(c_n)$. The output activation is thus given by the expression in Equation (3), hence the output with the highest activation is selected as the most probable hypothesis given the input sequence. This network architecture was first proposed by Gorin et al. [Gorin91].

Although the simplifying independence assumption does not usually hold in practice, this mutual information network has been shown to learn input-output associations quite successfully [Gorin91][Miller93][Sankar93].

The input independence assumption implies that classification does not depend on the order of the input tokens. To take into account the fact that adjacent input tokens sometimes form phrases or sentence fragments having significant information contents, we can introduce higher-order input units which are activated when particular token sequences occur. With the addition of units representing pairs of adjacent tokens, the mutual information network becomes equivalent to a MAP decision rule in a first-order Markovian setting (in which the probability distribution of each token depends only on the immediately preceding token) [Gorin91]. Input units representing fragments of three or more consecutive tokens model longer-range dependencies in the input channel.

Connection weights of high-order input units representing fragments are computed via *excess mutual information*; i.e., the connection weight from a unit representing the fragment $t_l t_m$ to the output unit c_n is given by

$$w_{lmn} = I(t_l t_m, c_n) - I(t_l, c_n) - I(t_m, c_n)$$

The mutual information network architecture is depicted in Figure 5 on page 45. Some of the connections (e.g., from t_1 to c_3, from t_2 to c_2 and c_3, etc.) are omitted for clarity. Input unit t_{23} represents the two-token fragment $t_2 t_3$, while input unit t_{234} represents the three-token fragment $t_2 t_3 t_4$.

Two important differences distinguish the network architecture in Figure 5 from traditional neural networks (e.g., backpropagation networks). First, the network topology is dynamic and data-driven: input units (and their associated connections) are created only

when the corresponding tokens or fragments are detected in some input sequence. Second, the high-order input units in layers 2 and above are *not* hidden units; the connections from lower-order units to higher-order units simply represent the fact that a fragment becomes active only if its constituent tokens are active in the correct order in an input sequence.

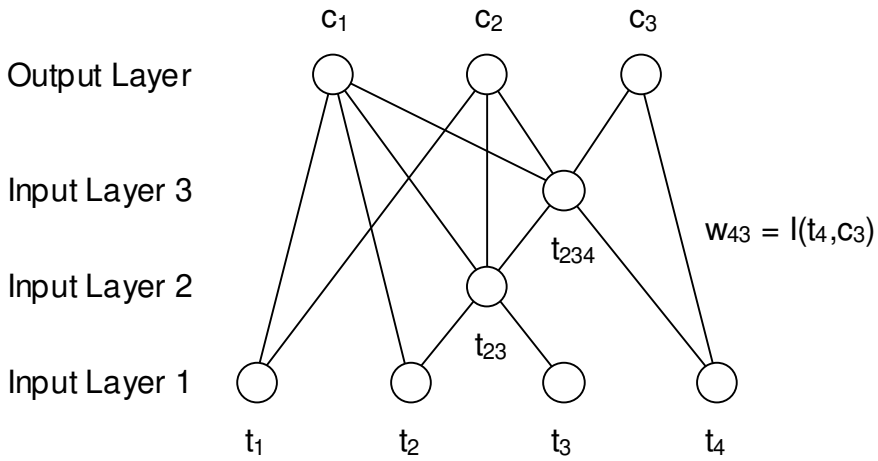


Figure 5. Mutual Information Network Architecture

Because the number of high-order input units can explode quite rapidly, we prune away units that are not useful for classification. This can be done using a measure called *salience* which is indicative of input relevancy with respect to classification [Gorin95]. The salience of a fragment *f* with respect to the output classes $c_n, n = 1 \ldots N$, is defined as

$$S(f) = \sum_{n=1}^{N} P(c_n \mid f) I(f, c_n) \tag{4}$$

The above quantity is precisely the information that *f* provides about the random variable representing the output classes, as discussed in section 3.2.1 in connection with the notion of information streams. High-order input units representing fragments are pruned based on their salience and the internal mutual information between the tokens in each fragment.

While the mathematical model underlying the mutual information network is not new, its reformulation into this network architecture is convenient for a number of reasons. A network formulation makes explicit the conditional independence assumptions in the model by showing separate connections from different input units to the same output unit. The connections could be labeled with $P(t_m \mid c_n)$ (the way Bayesian networks are normally formulated); however, this labeling suffers from several deficiencies: there is no scaling or normalization, input combination is multiplicative, and the associations measured by the weights can only have values between 0 and 1. Labeling the connections with $\log[P(t_m \mid c_n)/P(c_n)]$ provides a normalization factor in the form of $P(c_n)$, results in additive input combination, and produces weights that can be positive to indicate excita-

tory associations, negative to indicate inhibitory associations, or zero when there is no association. Furthermore, Tishby and Gorin showed that this network architecture exhibits a dual structure (statistical and algebraic) that can be exploited to learn associations with high confidence from very few examples [Tishby94].

3.4.2 The Multi-State Mutual Information Network

The basic mutual information network described above assigns a single label to each input sequence. As the goal of input integration is to produce a sequence of labels (i.e., parameter slots) by segmenting the input, we need to extend the network to handle this case.

As shown in the previous section, with suitable independence assumptions, the output activation levels in the basic mutual information network can be regarded as estimates of $\log_2$ of the *a posteriori* probabilities $P(c_n \mid t_1 t_2 \ldots t_m)$. We want to develop a similar score for input segmentations and label assignments.

Assume we have an input segmentation $s_1 s_2 \ldots s_k$ where each s_i is a group of adjacent input tokens (in the general case the group may consist of subgroups from different modalities). We want to label each segment s_i with the name of an output class representing a parameter slot. Consider a possible label assignment $c_1 c_2 \ldots c_k$, where the labels are drawn from the list of output classes that represent all the available parameter slots. The "goodness" of this label assignment can be evaluated using the *a posteriori* probability $P(c_1 c_2 \ldots c_k \mid s_1 s_2 \ldots s_k)$. We need a method to estimate this probability.

By the definition of conditional probability,

$$P(c_1 c_2 \ldots c_k \mid s_1 s_2 \ldots s_k) = P(c_k \mid s_1 s_2 \ldots s_k \wedge c_1 c_2 \ldots c_{k-1}) P(c_1 c_2 \ldots c_{k-1} \mid s_1 s_2 \ldots s_k) \quad (5)$$

We can now estimate the two factors in the above product separately and combine the result to arrive at an estimate for $P(c_1 c_2 \ldots c_k \mid s_1 s_2 \ldots s_k)$.

The value $P(c_k \mid s_1 s_2 \ldots s_k \wedge c_1 c_2 \ldots c_{k-1})$ represents the dependence of the k^{th} label assignment on both the input data and the past history of the first $k-1$ label assignments. We now make the bold step of integrating syntactic and semantic probabilities in one expression by replacing the first $k-1$ input segments with their respective labels to get the equivalent input sequence $c_1 c_2 \ldots c_{k-1} s_k$, and stating that

$$P(c_k \mid s_1 s_2 \ldots s_k \wedge c_1 c_2 \ldots c_{k-1}) \approx P(c_k \mid c_1 c_2 \ldots c_{k-1} s_k) \quad (6)$$

The *a posteriori* probability expression $P(c_k \mid c_1 c_2 \ldots c_{k-1} s_k)$ turns the semantic labels c_i into syntactic elements in the input. This is a way of exploiting history by using past information in new estimates. The advantage of this transformation is that

$P(c_k \mid c_1 c_2 \ldots c_{k-1} s_k)$ can be estimated by feeding the transformed input sequence $c_1 c_2 \ldots c_{k-1} s_k$ to a mutual information network.

To estimate the second factor in Equation (5), we make the simplifying assumption that the partial output sequence $c_1 c_2 \ldots c_{k-1}$ depends only on the input tokens in the first $k-1$ segments; thus

$$P(c_1 c_2 \ldots c_{k-1} \mid s_1 s_2 \ldots s_k) = P(c_1 c_2 \ldots c_{k-1} \mid s_1 s_2 \ldots s_{k-1}) \tag{7}$$

Combining Equations (5), (6) and (7) yields

$$P(c_1 c_2 \ldots c_k \mid s_1 s_2 \ldots s_k) \approx P(c_k \mid c_1 c_2 \ldots c_{k-1} s_k) P(c_1 c_2 \ldots c_{k-1} \mid s_1 s_2 \ldots s_{k-1}) \tag{8}$$

Let $\tilde{P}(c_1 c_2 \ldots c_k \mid s_1 s_2 \ldots s_k)$ denote an estimate of $P(c_1 c_2 \ldots c_k \mid s_1 s_2 \ldots s_k)$ that satisfies the above recurrence relation exactly. We can use recursive decomposition to obtain

$$\begin{aligned}
\tilde{P}(c_1 \ldots c_k \mid s_1 \ldots s_k) &= P(c_k \mid c_1 \ldots c_{k-1} s_k) \tilde{P}(c_1 \ldots c_{k-1} \mid s_1 \ldots s_{k-1}) \\
&= P(c_k \mid c_1 \ldots c_{k-1} s_k) P(c_{k-1} \mid c_1 \ldots c_{k-2} s_{k-1}) \tilde{P}(c_1 \ldots c_{k-2} \mid s_1 \ldots s_{k-2}) \\
&= \cdots \\
\tilde{P}(c_1 \ldots c_k \mid s_1 \ldots s_k) &= P(c_k \mid c_1 \ldots c_{k-1} s_k) P(c_{k-1} \mid c_1 \ldots c_{k-2} s_{k-1}) \ldots P(c_1 \mid s_1)
\end{aligned} \tag{9}$$

This is a novel way of combining between-channel information (associations between input tokens and output classes) and within-channel information (associations among output classes). Taking the logarithm of both sides yields

$$\begin{aligned}
\log_2 \tilde{P}(c_1 \ldots c_k \mid s_1 \ldots s_k) = \log_2 P(c_1 \mid s_1) + \log_2 P(c_2 \mid c_1 s_2) + \\
\log_2 P(c_3 \mid c_1 c_2 s_3) + \ldots + \log_2 P(c_k \mid c_1 \ldots c_{k-1} s_k)
\end{aligned} \tag{10}$$

Each term in the above sum is estimated by an output activation in the mutual information network, hence the sum can be interpreted as the score of a path that goes through the segment labels $c_1 c_2 \ldots c_k$ in order, as illustrated in Figure 6.

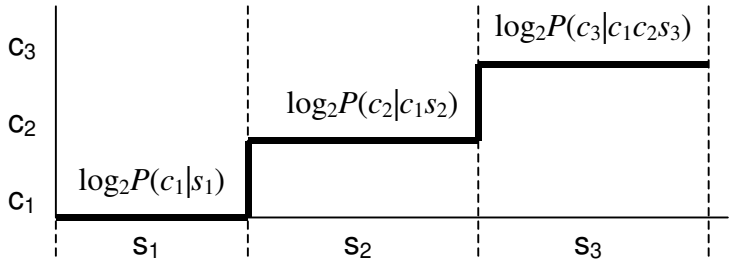


Figure 6. Path Score of Input Segmentation and Labeling

Using a dynamic programming algorithm similar to the Viterbi search [Viterbi67] or Dynamic Time Warping (DTW) [Sakoe71][Rabiner78] in speech recognizers, we can find an input segmentation and a corresponding label assignment that together maximize

the path score. Multiple input modalities are accommodated by implementing the path score maximization algorithm over more than one input dimension, where each dimension extends along one input stream. Figure 7 shows a path over two input dimensions.

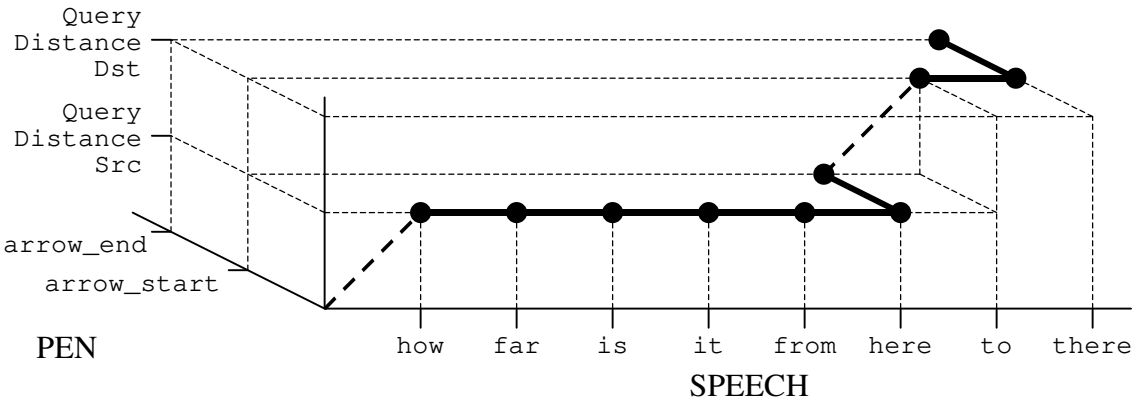


Figure 7. Output Path Over Multidimensional Inputs

The above path score maximization procedure effectively adds an extra layer on top of the basic mutual information network (see Figure 8 on page 49). Each output unit of the mutual information network represents an output state, and the top layer produces the best sequence of states that fits the input, in a manner reminiscent of the Multi-State Time Delay Neural Network (MS-TDNN) [Haffner92]. The MS-TDNN uses a Time Delay Neural Network (TDNN) [Waibel89a] to assign classification scores to inputs in a time sequence, then employs dynamic programming to determine the best sequence of output states with respect to the input sequence, thus combining input segmentation and classification in a single step. The MS-TDNN has been applied to a variety of classification problems that involve alignment and segmentation, including continuous speech recognition [Haffner92], word-spotting [Zeppenfeld93], lip-reading [Bregler93], and handwriting recognition [Manke95].

Because of the similarity between the way the MS-TDNN augments the TDNN and the way the network architecture in Figure 8 augments the mutual information network, this new network architecture is called the *Multi-State Mutual Information Network*, or MS-MIN.

The current implementation of the MS-MIN produces a parameter slot sequence in only one action frame for each multimodal input event. However, it would be straightforward to modify the dynamic programming algorithm in the state layer to produce a path through more than one action frame. Such an enhanced network would be able to parse a multimodal input event that maps to a sequence of two or more actions. This could accommodate input sentences such as “Cancel the meeting with Fred and schedule a meeting with John” in an appointment scheduler that supports *CancelMeeting* and *ScheduleMeeting* actions.

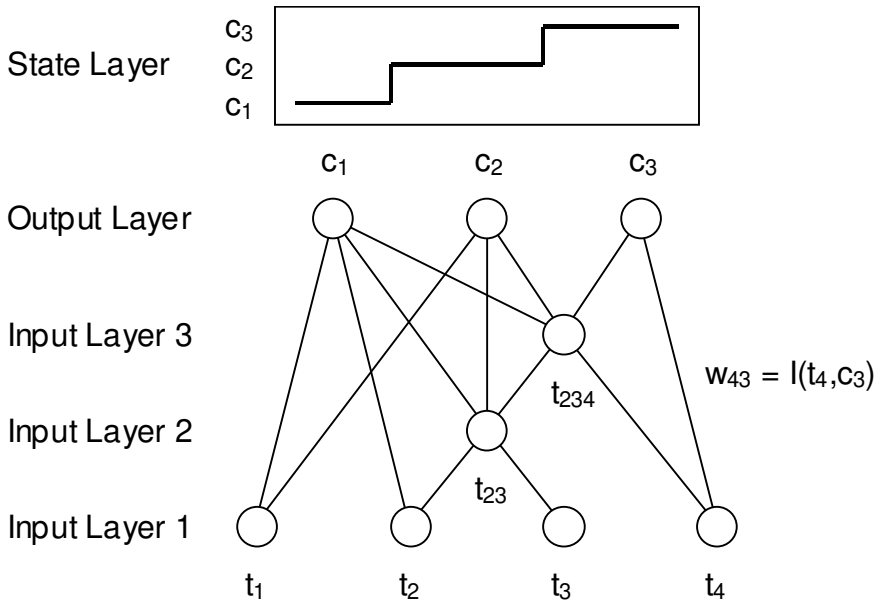


Figure 8. Multi-State Mutual Information Network Architecture

3.4.3 Training the MS-MIN

In backpropagation neural networks, the connection weights are incrementally adjusted during training by a gradient descent algorithm to minimize a classification error function [Rumelhart86]. In contrast, the weights in a mutual information network can be computed directly from input-output occurrence probabilities observed in the training data, independent of the order in which the training examples are presented, as shown in Equation (3). These probabilities can be estimated by a simple counting procedure [Gorin91]. Section 5.6.2 describes how the weights can be automatically generated from a grammar-based input model. Thus this network architecture enjoys a definite advantage over backpropagation networks because the training time is drastically reduced or eliminated altogether.

In principle, the mutual information network is capable of learning incrementally during actual use, as demonstrated by Gorin and others [Gorin91][Miller93][Sankar93]. The MS-MIN inherits this capability; however, more experiments are needed to determine to what degree this is true in practice, when the task domains are complex and a great amount of training data must be used to achieve adequate coverage. Section 7.2.1 outlines some test results that suggest the MS-MIN can indeed learn incrementally from examples presented to it.

Chapter 4
MULTIMODAL APPLICATION FRAMEWORK ARCHITECTURE

There are many system components needed in the construction of a multimodal application. Although the application details may vary from domain to domain, it is possible to abstract the common elements into a domain-independent infrastructure. This chapter describes the *Multimodal Application Framework* (MMApp), a collection of software components and a system architecture that constitute such an infrastructure.

The software components of MMApp are implemented using three programming languages. User interface components are written in Java [Arnold96], an object-oriented language featuring platform independence and easy network deployment over the World Wide Web. Computation-intensive components are written in C [Kernighan88] and C++ [Stroustrup91] to take advantage of the speed offered by those languages. Some C/C++ components supply a shell based on the Tool Command Language (Tcl) [Ousterhout94], an embeddable language that provides scripting capabilities on top of a C/C++ core.

Some of the technologies underlying the components of MMApp have been targets of extensive research. Recognition engines for speech, gestures, and handwriting fall under this category. Rather than duplicating the efforts already made in these fields, it is more advantageous to harness the power of existing systems in the development of MMApp. The contribution of this work with respect to the incorporation of existing components consists of developing a uniform and reusable Application Programming Interface (API) for these components, allowing MMApp to make use of their services in a generic way. This approach enhances the modularity of the framework and enables transparent component replacement.

4.1 Object-Oriented Concepts and Design Patterns

The design of the software components in the MMApp framework and the MMTk workbench in the next chapter is founded upon object-oriented techniques. This section presents a brief overview of the object-oriented concepts behind the design.

Objects and Classes

Every concept in an object-oriented system is represented by an *object*. Each object has a *type* represented by a *class* which encapsulates data and operations on that data. The data describes the state of the object and the operations (called *methods*) describe its the behavior. An object is an *instance* of its class, hence object creation or construction is also called *instantiation*.

A basic tenet of object-oriented design is data hiding: the data or state of an object should only be accessed or modified via the object's methods. Data hiding ensures that changing the implementation of a class will not break any code that uses the class, as long as the methods of the class and their behavior do not change.

Inheritance and Polymorphism

A class can be declared a *subclass* of another class (called a *superclass* or *base class*). A subclass is said to *extend* (or be derived from) its superclass, because it *inherits* all the data and behavior of the superclass and may add data and behavior of its own. A collection of classes related by the superclass-subclass relationship forms a *class hierarchy*.

A subclass may *override* a method inherited from its superclass by defining a method of the same name and parameter types (which together form the *signature* of the method). Because the subclass inherits the behavior of the superclass, an instance of the subclass may be referenced as if it belonged to the superclass; in that case, a call to an overridden method through the superclass reference must be resolved to either the superclass or the subclass version of the method. If the method resolution takes into account only the type of the object reference (i.e., the superclass version of the method will be called), it is termed *static binding* and may be performed at compile-time. If the method resolution occurs at runtime and looks up the correct method based on the actual type of the object (i.e., the subclass version of the method will be called), it is termed *dynamic binding*. Dynamically bound methods are sometimes called *virtual methods*.

If a superclass declares a virtual method that is overridden in subclasses, a call to the method may resolve to different methods at runtime depending on the actual object type. This behavior is called *polymorphism*. It is useful when some program code must perform different operations for different object types, but the actual type of an object is not known at compile-time.

Abstract Interface

If several classes support the same kind of operations, but each class implements the operations differently, the classes can be declared subclasses of a common superclass. The superclass declares methods for the operations, and the subclasses override the methods to supply their own implementations. The sole purpose of the superclass is to serve as a placeholder for the method declarations; no instance of the superclass will ever be constructed. This type of superclass is called an *abstract class*, and the methods it declares (for the sole purpose of letting subclasses override them) are *abstract methods*.

An abstract class and its abstract methods form an *abstract interface*. The subclasses that override the abstract methods are *concrete implementations* of the interface.

Design Patterns

The influential work of Gamma et al. [Gamma95] has been instrumental in promoting the use of *design patterns* as a way of facilitating object-oriented design. Design patterns

capture design experience in a form that people can use effectively, making it easier to reuse successful designs and architectures.

The design of the software components described in this dissertation relies heavily on several design patterns described in [Gamma95]:

- The *Abstract Factory* pattern provides an interface for creating objects without specifying their concrete classes;
- The *Adapter* pattern converts the interface of a class into another interface clients expect;
- The *Factory Method* pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate;
- The *Observer* pattern defines a one-to-many dependency between objects so that state changes are automatically broadcast to the dependents;
- The *Template Method* pattern defines a skeleton of an algorithm, deferring some steps to subclasses;
- The *Visitor* pattern represents an operation to be performed on the elements of an object structure.

Appendix C describes the above design patterns in more detail.

Unified Modeling Language

The easiest way to describe an object-oriented design is to use a graphical notation. The Unified Modeling Language (UML) [Fowler97] offers such a notation. UML is an emerging standard for describing software systems in terms of an object-oriented design. This dissertation only makes use of the UML graphical notation for class diagrams; however, UML actually encompasses much more than the class diagram notation.

Class diagrams describe classes, methods, inheritance, interfaces, and object associations. Appendix B contains a summary of the UML class diagram notation. This notation is used throughout Chapter 4 and Chapter 5 as well as for the description of design patterns in Appendix C.

4.2 Overall System Design

The main design goal for the MMApp framework was to construct an infrastructure that is modular, distributed, and customizable.

MMApp Is Modular

MMApp includes a library of software components that can be assembled to create a multimodal application. This library contains speech and pen input recorders and recognizers, multimodal event handlers, interprocess communication facilities, and a user

interface in the form of a Java applet. The user interface includes ready-made objects that handle input capture and synchronization, communicate with speech and pen input recognizers, and direct the control flow of the multimodal interpretation process.

This modularity permits multimodal application developers to customize each system component separately or even replace certain modules if the need arises. For example, a different audio capture engine for speech recording or another speech recognizer more suitable for the target platform could be substituted for the default components without affecting the rest of the framework.

MMApp Is Distributed

The distributed nature of the framework serves to spread the computational load among multiple machines, improve the responsiveness of the system, and allow resource sharing using a client/server architecture.

Each major component of the system runs as a separate process which could be hosted on a different machine if necessary. Computation-intensive components such as the speech/gesture recognizers and the multimodal semantic integrator are configured as backend servers that normally run on powerful workstations and serve multiple user interface clients. Interface client components, which handle input capture and output presentation and thus do not need as much computational power, are part of Java applets that can run inside Web browsers on any machine, including PCs and network computers.

MMApp includes a communication layer that presents an abstract interface for inter-process communication, hiding all the details of network protocols and synchronization.

MMApp Is Customizable

A major philosophy that permeates the design of MMApp is to provide defaults that are immediately useful in application development, but always allow the developer to override the default behavior or substitute alternate implementations to suit the requirements of a particular application.

The basic components of the framework can be easily adapted to different task domains. To construct the user interface of a new multimodal application, it is only necessary to derive a subclass from the supplied multimodal applet and customize the domain-dependent parts; all the basic multimodal input handling capabilities are automatically inherited.

Using the design process and the toolkit described in Chapter 5, application developers can construct a multimodal input model that serves as the starting point for the instantiation and customization of the application. The tools used in the design process can automatically generate a statistical language model to adapt the speech recognizer to the target domain, as well as multimodal interpretation modules for the application.

Other possible customizations include the modification of gesture templates to change the gesture vocabulary, and the substitution of a different transport protocol in the communication layer for the default text-based scheme.

4.2.1 System Architecture

Figure 9 shows a block diagram of the overall system architecture.

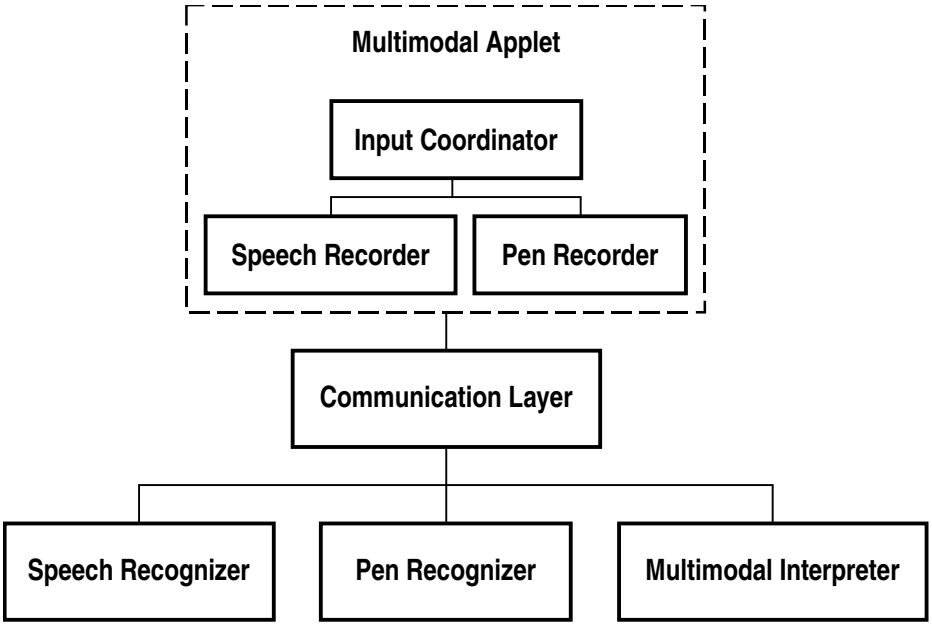


Figure 9. Multimodal Application Framework Architecture

The multimodal applet is the user interface; the applet window presents a view onto a domain-dependent representation of application data and state in the form of objects to be manipulated. The speech and pen recorders are responsible for capturing audio and pen inputs and requesting the services of the speech and pen recognizers to convert the inputs to text and gesture shapes. The input coordinator groups input events into combined multimodal events and sends the aggregate inputs to the multimodal interpreter for semantic integration. The communication layer provides the foundation for all interprocess communication.

The current implementation of MMApp strongly supports only the speech and pen modalities; however, other input modalities are still supported by the framework, requiring only the addition of appropriate recorders and recognizers.

The major components of the MMApp framework are described in sections 4.4 to 4.7.

4.2.2 Component Interface and Implementation

The modular design of the MMApp framework is founded upon a clean separation between component interface and implementation. Each system component depicted in Figure 9 is an object that exposes a well-defined interface. Other objects communicate with the component only through this abstract interface and do not rely on any particular concrete implementation of the component. An entirely different implementation can be easily inserted, provided the interface remains the same.

When existing software components (see section 4.3) are incorporated into MMApp, their native APIs must be adapted to fit the component interfaces specified by MMApp. Each component is wrapped inside a software layer that exposes the right interface and internally calls the component's native API to do the work. These "wrappers" are instances of the Adapter design pattern (see Appendix C.2). In these cases, the contribution of the present work consists of the specifications of the wrapper APIs—the component interfaces—that make it possible to integrate diverse implementations of the same software subsystem into MMApp in a modular and generic way. Examples of wrapper APIs are presented in sections 4.4 and 4.5.

4.2.3 Input Processing Issues

One important appeal of multimodal systems is the promise of a flexible and natural style of human-computer interaction made possible by the interplay among multiple modalities. Because MMApp relies on speech and pen recognition subsystems to convert raw input data into intermediate symbolic representations (words, shapes, etc.), the properties of these subsystems have a significant impact on the perceived interactive characteristics of the whole system. The components that compose the relevant subsystems have to be selected with care to suit application-specific needs. The default choices in MMApp reflect the domain-independent goal of maximizing flexibility by minimizing the number of constraints on the users' interaction with the system.

Speech Recognizer Selection

As explained in section 2.1, there are many types of speech recognizers. This variability does not matter greatly from the perspective of interfacing the speech recognizer with the rest of the MMApp framework, as the main function of the speech recognizer remains the same: converting audio data to text. However, the choice of speech recognizer does affect the flexibility and generality of the interaction with the users. Discrete-word recognizers achieve high accuracy but require the speaker to pause between words, resulting in an unnatural style of interaction. Multimodal human-computer interaction is attractive precisely because of the flexibility and naturalness it offers; therefore, it is desirable to configure the speech recognizer such that there is as little constraint on the speaking style as possible. A large-vocabulary, continuous speech recognizer would be the most suitable for this purpose.

There are two large-vocabulary, continuous speech recognition systems available at Carnegie Mellon University, namely JANUS (see section 4.3.1) and SPHINX (see section 4.3.2). Either one of them, as well as myriad other systems available from research institutions or on the market, could be installed as the speech subsystem in MMApp. As a demonstration of the modular architecture of the MMApp framework, wrappers for both JANUS and SPHINX are provided to implement the SpeechRecognizer API described in section 4.4.3. It is possible to wrap this API around any other equivalent speech recognizers.

Handwriting Recognizer Selection

The same criteria that influence the speech recognizer selection mandate the choice of a handwriting recognizer flexible enough to let the user write in a variety of styles. This eliminates systems that can handle only block letters or require the user to print characters in separate boxes. There exist handwriting recognition systems that can process continuous cursive scripts [Schenkel94][Menier94][Manke95]. MMApp includes support for the NPen++ system described in section 4.3.3. The modular design of the MMApp framework allows the use of any other equivalent handwriting recognizer, provided that the API is appropriately adapted.

Gesture Recognition Model

Each pen input event delivers a series of strokes—sequences of points between pen-down/pen-up occurrences—that together make up a gesture. Much of the current research on gesture recognition surveyed in section 2.2.2 focuses on mapping each gesture to a single shape designation (one exception is [Kurtenbach91] which does mention "compound gestures"). Unfortunately, there are many things that seem very intuitive but cannot be expressed as a single gesture shape. For instance, the user might elect to circle a group of objects, then immediately follow that with a symbolic gesture that specifies an operation on the designated objects (e.g., a cross to delete the objects). With the single-label approach, the user is forced to do this in two separate steps as two distinct pen input events.

The MMApp framework assumes a more general gesture recognition model that subsumes the single-label approach. A gesture is interpreted as a sequence of one or more components, each of which is labeled with a shape designation. With this gesture recognition model, a limited gesture vocabulary or alphabet can have great expressive power. MMApp includes a template-based gesture recognizer that supports this model.

Integrating Gesture and Handwriting Recognition

The above gesture recognition model can be extended to incorporate handwriting recognition by classifying handwritten words as a gesture component type in addition to existing shape types. In this model, a pen input event maps to a sequence of gesture/handwriting components; gesture components are labeled with shape designations whereas handwriting components have the label *handwriting* and carry an attached data

packet containing the recognized handwritten text. Figure 10 shows an example of a handwritten word followed by an arrow gesture.

Figure 10. Gesture/Handwriting Combination

The combined gesture/handwriting model is very flexible and intuitive from a usability point of view, but its implementation reveals many difficulties. One major hurdle is the need to distinguish handwritten words from gesture strokes. A heuristic that works reasonably well in many cases is to invoke the handwriting recognizer on strokes that do not closely match any shape in the gesture vocabulary. This heuristic is effective if the handwriting is cursive and continuous as in Figure 10, but often confuses block letters with gesture shapes. Experiments are needed in future research to evaluate the effectiveness of this and other gesture/handwriting combination heuristics in variable settings.

4.3 Existing Software Components

This section briefly presents some existing software components that provide services exploited by MMApp. The wrapper APIs around these components are described later.

4.3.1 The JANUS Speech Recognizer

JANUS was first conceived in 1990 as the recognition backend of a speech-to-speech translation system [Waibel91]. Version 2 (1994) added the capability of recognizing spontaneous speech containing hesitations, false starts, and other non-speech noises. Version 3 (1996) restructured the recognition system as a toolkit of powerful and flexible modules built on top of a Tcl scripting shell. This is the version supported by MMApp. All speech recognition accuracy figures reported in Chapter 7 were obtained with JANUS as the speech recognition subsystem.

The JANUS Recognition Toolkit (JRTk) is a flexible architecture for experimenting with language specific phenomena [Finke97]. Raw speech data is preprocessed to extract one or more streams of input features derived from Mel-scale, cepstral, or Perceptual Linear Predictive (PLP) filters processed using Linear Discriminant Analysis (LDA). The context-dependent acoustic units are modeled via continuous density Hidden Markov Models (HMMs). The system includes explicit noise models to cope with breathing, lip-smack, and other noises inherent in spontaneous speech. Many recent improvements have been introduced, such as speaker normalization, polyphonic modeling, dictionary learning, morpheme-based, phrase-based and class-based language models, etc.

The recognition performance of JANUS has been reported in many research papers [Lavie97][Zeppenfeld97][Woszczyna98]. On an appointment negotiation task (spontaneous speech), a word accuracy of 88% (German) and 77% (English) was achieved using a 5,000-word vocabulary. The accuracy is 93% for the 65,000-word North American Business News task (speaker independent read speech). On Switchboard and Call Home, two extremely difficult speech recognition tasks involving multi-topic spontaneous human-human speech over the telephone line, JANUS was among the group of top systems in 1996 and 1997 evaluations organized by NIST, with a word error rate of as low as 26% (SWB 96 test set).

4.3.2 The SPHINX Speech Recognizer

The SPHINX speech recognition system was the product of speech recognition research at Carnegie Mellon University in the 1980's and 90's. It forms the front-end of the CMU Air Travel Information Service (ATIS) system described in [Ward95]. The version supported by MMApp, SPHINX-II, is deployed in projects ranging from large-vocabulary dictation systems [Alleva92] to wearable computers [Smailagic96].

The SPHINX-II system [Huang93] is based on HMMs with multiple codebooks. Features extracted from speech data include dynamic features (LPC cepstrum coefficients, differenced LPC cepstrum coefficients, power, and differenced power) and neural-network-based speaker-normalized features. Acoustic models include semi-continuous HMMs, senone, and tree-based allophonic models. The search module employs multiple threshold pruning and a hybrid search strategy that uses Viterbi search for the model state graph and Viterbi beam search for the word level model graph. Language modeling incorporates some experimental techniques such as long distance bigrams and modified back-off language models.

SPHINX-II achieves 97% accuracy on the 1,000-word DARPA Resource Management task [Huang93] and 84% on the 20,000-word Wall Street Journal task as of 1994 [Rosenfeld94]. The latter performance figure was obtained using a system similar to that employed in MMApp, but since then much improvement has resulted from more recent advances, including the development of SPHINX-III.

4.3.3 The NPen++ Handwriting Recognizer

The NPen++ on-line handwriting recognition system [Manke95][Manke98], developed by Stefan Manke at University of Karlsruhe, is writer independent and can handle any common writing style—cursive, hand-printed, or a mixture of both. High recognition performance can be achieved even with dictionary sizes up to 100,000 words, without training or adaptation for a particular writer. This is achieved by exploiting dynamic writing information (i.e., the temporal sequence of data points recorded on pen input devices).

NPen++ employs robust preprocessing techniques to transform the sequence of data points into a sequence of N-dimensional feature vectors. The original sequence is first

resampled using equidistant points and smoothed in order to remove sampling noise. The words are then rotated and scaled using baseline and centerline information derived by regression. Finally, feature vectors are extracted from the normalized coordinate sequence. The features include relative x-y movements, curvature, writing direction, and *context-bitmaps* representing low-resolution, bitmap-like descriptions of each coordinate's proximity. The context-bitmaps are local in space but global in time and serve to encode temporal long-range context dependencies that cannot be modeled by the other features which are strictly local.

The recognition component integrates recognition and segmentation of words into a single neural network architecture originally proposed for continuous speech recognition. This Multi-State Time Delay Neural Network (MS-TDNN) combines a high-accuracy pattern recognition network with a non-linear time alignment algorithm for finding strokes and character boundaries in isolated handwritten words.

NPen++ has been tested on writer independent recognition of isolated words with dictionary sizes from 1,000 to 100,000 words. Word recognition rates range from 98.0% for the 1,000-word dictionary to 91.4% on a 20,000-word dictionary and 82.9% for the 100,000-word dictionary without using any language model.

4.3.4 The NetscapeSRec Speech Recorder Plug-In

The Netscape Navigator browser supports dynamically loaded modules that extend the capabilities of the base software. These so-called plug-ins can be accessed by Java applets running inside the browser. Unlike Java applets, the plug-in modules are free from security restrictions because they employ native C code, making it possible for plug-ins to access low-level system services that are hidden from applets.

NetscapeSRec is a Netscape plug-in written by Xing Jing (with collaboration from Weiyi Yang for the Windows 95/ NT version) at the CMU Interactive Systems Lab to provide speech recording capabilities directly accessible by Java applets [Jing97]. When the browser loads a Hypertext Markup Language (HTML) page containing an `<embed>` tag that references the NetscapeSRec plug-in, the browser instantiates a Java object that represents a handle on the plug-in. Applets instantiated on the same HTML page can now record speech by invoking the plug-in object's native methods, which in turn transfer control to native C code in the plug-in module.

The NetscapeSRec plug-in does not require a separate process to be started and maintained as is the case with the SRecServer program described in section 4.4.1 below. Recording functionality automatically becomes available once the browser loads the appropriate HTML document. However, access to this functionality is restricted to clients written in the Java language to run as applets inside the browser.

4.4 Speech Components

The speech subsystem of MMApp consists of the SpeechRecorder and the SpeechRecognizer components. The SpeechRecorder provides audio capture and playback capabilities, while the SpeechRecognizer is responsible for mapping spoken inputs to text strings. Both the SpeechRecorder and SpeechRecognizer are abstract interfaces represented by abstract base classes. Concrete implementations of SpeechRecorder and SpeechRecognizer are subclasses that extend the abstract base classes using object-oriented inheritance. Different audio capture and recognition engines can be plugged into the system in a modular fashion without changing a single line of application code because clients of the speech subsystem see only the abstract interfaces. MMApp includes SRecServer—a program that exports speech recording services over the network—as well as support for the NetscapeSRec plug-in described in section 4.3.4. Wrappers are included to support both the JANUS and SPHINX speech recognizers.

4.4.1 The SRecServer Speech Recorder

SRecServer is a stand-alone program that exports speech recording services. It was developed for early multimodal applications at the CMU Interactive Systems Lab and later integrated into MMApp. Client applications that need to record speech input can send requests to SRecServer over the network. Client/server connections use Transmission Control Protocol/Internet Protocol (TCP/IP) sockets. Service requests are simple text strings, hence client programs can be implemented in any language. SRecServer is also capable of playing back previously recorded speech.

The actual data transfer between the audio hardware and the computer memory is normally controlled by a low-level platform-dependent audio driver. SRecServer supports several types of audio drivers and hides their differences behind a high-level interface. Client programs need not know what audio platform is being used for speech recording.

SRecServer has the disadvantage of requiring an extra computer process to be started and maintained, unlike the plug-in described in section 4.3.4 above. However, there is no language-dependent requirement on the part of client applications, and a single SRecServer instance can service multiple clients.

There exist versions of SRecServer for Sun SparcStations running Solaris, Hewlett-Packard workstations running HP-UX, DEC ALPHA workstations running Digital Unix, and Intel PCs running Linux.

4.4.2 The SpeechRecorder Interface

Figure 11 on page 61 is a class diagram that shows the SpeechRecorder interface in UML notation (see Appendix B).

SpeechRecorder is an abstract base class that exposes an interface for speech recording and playback. The two concrete subclasses, SocketSpeechRecorder and PluginSpeechRecorder, are wrappers around the previously described speech recording programs.

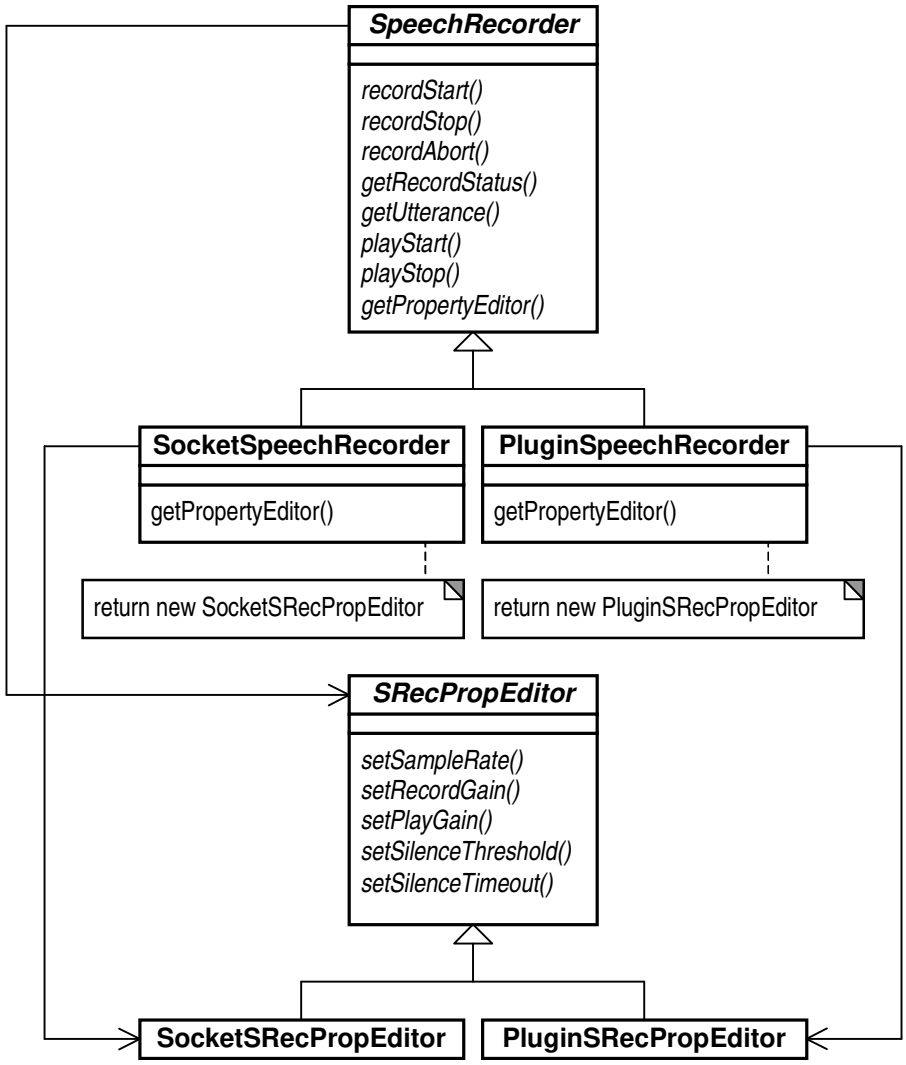


Figure 11. SpeechRecorder Interface

The SpeechRecorder API consists of methods to control speech recording/playback and adjust recording parameters. Calling recordStart() puts the SpeechRecorder in listening mode; speech data is not actually saved until the signal level exceeds a silence-detection threshold[†]. Recording ends when recordStop() is called or when the signal level falls below the threshold for longer than a preset time-out, indicating that the user has stopped speaking. Listening and recording state transitions are reported by getRecordStatus(). getUtterance() retrieves the recorded speech data. playStart() and playStop() controls the playback of previously recorded speech. getPropertyEditor() returns an object that knows

[†] Setting the threshold to zero results in immediate recording mode.

how to change the recording parameters of the SpeechRecorder; the exact type of the returned SRecPropEditor object is left for SpeechRecorder subclasses to specify using the Factory Method design pattern (see Appendix C.3).

SocketSpeechRecorder adapts the SpeechRecorder API to the service interface exported by SRecServer (see section 4.4.1). Calls to SpeechRecorder methods are converted to SRecServer requests and sent to the server program via TCP/IP sockets. SocketSpeechRecorder creates a SRecPropEditor subclass that knows how to set recording parameters by sending SRecServer requests.

Similarly, PluginSpeechRecorder adapts the SpeechRecorder API to the NetscapeSRec interface (see section 4.3.4) by routing API method calls to the plug-in. The SRecPropEditor subclass created by PluginSpeechRecorder also uses the plug-in API to adjust recording parameters.

4.4.3 The SpeechRecognizer Interface

The class diagram in Figure 12 shows the SpeechRecognizer interface.

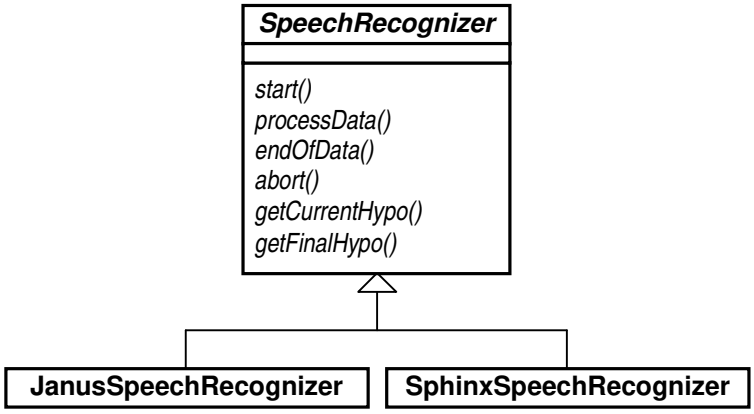


Figure 12. SpeechRecognizer Interface

SpeechRecognizer is an abstract base class that exposes an interface for speech recognition. The two concrete subclasses, JanusSpeechRecognizer and SphinxSpeechRecognizer, are wrappers around the speech recognition systems described in section 4.3.

A recognition job begins when start() is called. processData() should then be repeatedly invoked to transfer speech data to the recognizer, followed by a call to endOfData() at the end of the speech data stream. getCurrentHypo() can be called at any time to retrieve the best recognition hypothesis given the input data up to that point. After the completion of data transfer, getFinalHypo() retrieves the best recognition hypothesis for the entire input utterance.

JanusSpeechRecognizer adapts the SpeechRecognizer API to the JANUS API, and SphinxSpeechRecognizer does the same for SPHINX. Both JANUS and SPHINX require the raw speech data to be preprocessed using digital signal processing (DSP) algorithms

to compress each frame (block) of speech data down to a few feature values. The two speech recognition systems support different feature sets and data transfer protocols, but the adapted APIs handle the differences in a manner transparent to clients of the SpeechRecognizer API.

4.5 Pen Components

The pen subsystem of MMApp consists of a PenRecorder which captures pen strokes and a PenRecognizer which maps pen input to a sequence of gesture/handwriting components. As with SpeechRecorder and SpeechRecognizer, PenRecorder and PenRecognizer are abstract base classes that are extended by concrete implementation subclasses for different pen input capture and recognition engines. MMApp includes two pen recorders, one implemented in Java and the other configured to export pen recording services over the network for X Windows applications. The TmplGRec gesture recognizer provides gesture recognition services. A wrapper is included to support the NPen++ handwriting recognizer.

4.5.1 The XPRecServer Pen Recorder

The XPRecServer program complements SRecServer (see section 4.4.1) in that it exports pen recording services from a stand-alone server process in much the same way SRecServer exports speech recording services. The program currently runs only in the X Windows graphical environment.

XPRecServer supports several low-level pen device drivers. The default driver allows the mouse to be used for pen input. Using the mouse as a pen input device is a universal hardware-independent solution because touch-sensitive screens and digitizing tablets normally come with drivers to simulate mouse input. XPRecServer also provides direct support for some digitizing tablets and touch-sensitive screens for which mouse-simulation drivers are not available.

Application windows requiring pen input capabilities must be registered with XPRecServer. The program uses the registered window handles in two ways:

- If the mouse is used for pen input, XPRecServer intercepts mouse events on the target windows using X Windows pointer grabbing functions;
- As pen strokes are being recorded, XPRecServer tracks the mouse movements and draws “electronic ink” traces on the target windows to provide visual feedback of the pen recording process.

Using XPRecServer for pen recording involves the same tradeoffs as those listed for SRecServer in section 4.4.1: an extra computer process must be started and maintained, but a single XPRecServer instance can service multiple clients, and there are no special requirements for the client applications except that they must run on X Windows. The last point can be eliminated if XPRecServer is ported to other windowing environments

such as MacOS or Windows 95/Windows NT. The techniques used in XPRecServer should be applicable in any windowing environments that allow mouse event interception and drawing operations given a window handle.

4.5.2 The TmplGRec Gesture Recognizer

The gesture recognizer integrated into MMApp is an experimental system described briefly as part of the JEANIE multimodal calendar application presented in [Vo96]. It uses template matching to recognize individual gesture components and dynamic programming to produce the best gesture sequence. As is the case with the speech recognition subsystem, the modular design of the MMApp framework permits the use of other gesture recognizers as long as their APIs are appropriately adapted.

Template-Based Gesture Recognition

Given one or more pen strokes, each consisting of a sequence of coordinates, the gesture recognizer attempts to classify the stroke combination as one of many possible shapes. Each shape class in the gesture vocabulary is represented by one or more templates, which are simply prototypical gestures of that shape. The input gesture is compared to each template by first applying a transformation that deforms the gesture to match the template as closely as possible, then computing the residual difference using a mean-squared-error (MSE) measure. Allowable transformations are combinations of translation, rotation, and linear scaling along each coordinate axis. The template that produces the lowest residual difference below a set threshold is considered the best match. The input gesture is labeled as “unknown” if all the residual difference scores are greater than the threshold.

Stroke Preprocessing

In order to reduce the number of template comparisons and make it easier to implement the deformation algorithm, the gesture strokes are preprocessed using a vertex reduction algorithm as illustrated in Figure 13. Each stroke is first resampled so that adjacent points are uniformly spaced. “Sharp corners” are identified by analyzing the rate of change of the curvature at each point of the stroke. The portions of the stroke between consecutive corners are further reduced to a few sample points; in effect the curves are approximated by straight segments. The number of segments are selected based on curvature, and their endpoints are chosen to minimize the mean squared error between the original curve and the approximating segment sequence.

The preprocessing step reduces each gesture stroke to a few straight segments. The segment counts of the strokes in the input gesture constitute a *signature* of the gesture. Only templates with matching signatures are compared to the input, eliminating unneeded computation.

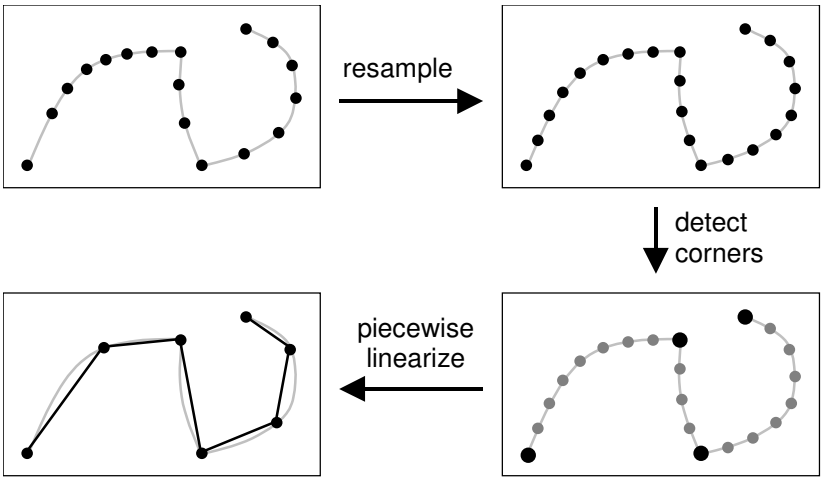


Figure 13. Preprocessing of Gesture Strokes

Template-Matching Deformation

To be compared to a template, a preprocessed gesture is deformed using a combination of translation, rotation, and scaling transformations. As scaling and rotation do not change the centroid, the gesture is first translated so that its centroid coincides with the centroid of the target template. Suppose the translated gesture consists of the points $(x_1, y_1) \ldots (x_n, y_n)$, or in matrix form:

$$V = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} v_1 \mid v_2 \mid \cdots \mid v_n \end{bmatrix} \text{ where } v_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Scaling by a factor of s_x along the x-axis and a factor of s_y along the y-axis is equivalent to multiplying the coordinate matrix by the scaling matrix

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

while rotating by an angle θ is equivalent to multiplying by the rotation matrix

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

The transformed gesture is thus

$$U = \begin{bmatrix} u_1 \mid u_2 \mid \cdots \mid u_n \end{bmatrix} = SRV = TV \tag{11}$$

where the combined transformation matrix is

$$T = SR = \begin{bmatrix} s_x \cos\theta & -s_x \sin\theta \\ s_y \sin\theta & s_y \cos\theta \end{bmatrix} \tag{12}$$

and

$$u_i = Tv_i = \begin{bmatrix} x_i s_x \cos\theta - y_i s_x \sin\theta \\ x_i s_y \sin\theta + y_i s_y \cos\theta \end{bmatrix} \tag{13}$$

The residual difference between the transformed gesture and the target template

$$\tilde{U} = \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 & \cdots & \tilde{x}_n \\ \tilde{y}_1 & \tilde{y}_2 & \cdots & \tilde{y}_n \end{bmatrix} = \begin{bmatrix} \tilde{u}_1 \mid \tilde{u}_2 \mid \cdots \mid \tilde{u}_n \end{bmatrix}$$

is the mean squared error

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=1}^{n} \| u_i - \tilde{u}_i \|^2 \\ &= \frac{1}{2}\sum_{i=1}^{n} [(x_i s_x \cos\theta - y_i s_x \sin\theta - \tilde{x}_i)^2 + (x_i s_y \sin\theta + y_i s_y \cos\theta - \tilde{y}_i)^2] \end{aligned} \tag{14}$$

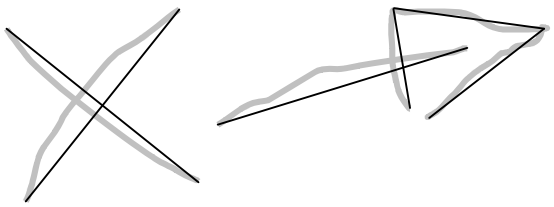
This is a function of 3 variables (s_x, s_y, and θ) that can be minimized using gradient descent. The minimum value of the mean squared error E is the match score for the corresponding template.

Gesture Sequence Segmentation

The above algorithm finds the best match for a series of one or more strokes. Given a sequences of strokes in a pen input event, we must find the best sequence of templates that match the input. To this end, groups of consecutive strokes are extracted from the input gesture and the best match is found for each group. A segmentation of the input gesture into successive groups of strokes is assigned a score computed by summing the match errors of the stroke groups. To induce a preference for stroke combinations over individual strokes, a transition penalty is added to the score whenever a new stroke group is started in the gesture segmentation. A dynamic programming algorithm finds the segmentation with the lowest score.

As an illustrative example, consider the preprocessed gesture shown in Figure 14 on page 67. No group of 3 or more consecutive strokes has a close match. The match scores for stroke groups of length 1 or 2 are organized in a score matrix as shown. The dynamic programming algorithm finds the lowest-scored path through the matrix. The final result is the label sequence *cross–arrow* with a recognition score of 0.7.

The above segmentation algorithm assumes that gesture components are cleanly separated at stroke boundaries. It is possible to extend the algorithm to detect mid-stroke component transitions, at the cost of added complexity and increased computation time.



	Stroke 1	Stroke 2	Stroke 3	Stroke 4
line	0.00	0.00	0.00	
cross	0.14			
triangle				0.23
arrow			0.31	

Transition penalty = 0.25
Best path score = 0.14 + 0.25 + 0.31 = 0.7

Figure 14. Gesture Recognition Example

Basic Gesture Vocabulary

The default set of templates in the gesture recognizer includes the shapes listed in Table 2.

Name	Examples
Point	
Circle	
Rectangle	
Triangle	
Line	
Arc	
Arrow	
Cross	

Table 2. Basic Gesture Vocabulary

This gesture repertoire is small but surprisingly expressive in an intuitive way. A pointing gesture can indicate a single object or screen location. Circling one or more objects is a natural way of grouping or calling attention to them as a whole. A rectangle can be used to delineate an approximate area of the screen. A line or an arrow serves to join two objects or locations. An arrow can also indicate a direction or movement. A cross can designate a screen position in an X-marks-the-spot fashion. Crossing something out to indicate removal is quite intuitive as well.

Combinations of basic shapes extend their semantics in useful ways. One example is a circle around a group of objects, followed by a cross or an arrow; the cross indicates that all the circled objects should be deleted, whereas the arrow signifies that the objects should be moved as a group in the indicated direction. Another example is an arrow to signify a move operation, followed by a cross to indicate the precise destination point.

It should be noted from the previous examples that a gesture by itself may be ambiguous; however, additional information from other modalities—speech for instance—may serve to disambiguate and select the correct meaning of the gesture. This is one important aspect of the appeal of multimodal interaction.

Gesture Recognition Accuracy

TmplGRec was tested on 500 gestures drawn by 5 people. Each person drew 20 samples of each of 5 shapes: arrow, circle, cross, rectangle, and triangle. Table 3 lists the recognition accuracy for each gesture shape.

Shape	**Accuracy (%)**
Arrow	74
Circle	80
Cross	49
Rectangle	69
Triangle	93
Overall	73

Table 3. Gesture Recognition Accuracy for 5 Shapes

An analysis of the confusion matrix (which tabulates how many times an example of gesture A is mistakenly recognized as gesture B) reveals that 39% of cross gestures were classified as two separate lines, and 25% of the rectangles were classified as circles. Most of the errors on crosses can be easily eliminated by adding templates and tweaking transition penalties. However, the errors on rectangles indicate an inherent weakness in the template-matching algorithm: reducing each pen stroke to a small number of line segments effectively blurs the distinctions between rectangles and circles!

TmplGRec was developed to satisfy the requirement of recognizing shape sequences instead of single shapes. In hindsight, it seems perfectly feasible to implement the dynamic programming segmentation algorithm on top of a proven single-shape recognizer such as Dean Rubine's [Rubine91] instead of the simplistic template matcher. However, the TmplGRec gesture recognizer did serve as a concept demonstration of gesture integration into the multimodal application framework.

4.5.3 The PenRecorder Interface

Figure 15 shows a class diagram for the PenRecorder interface.

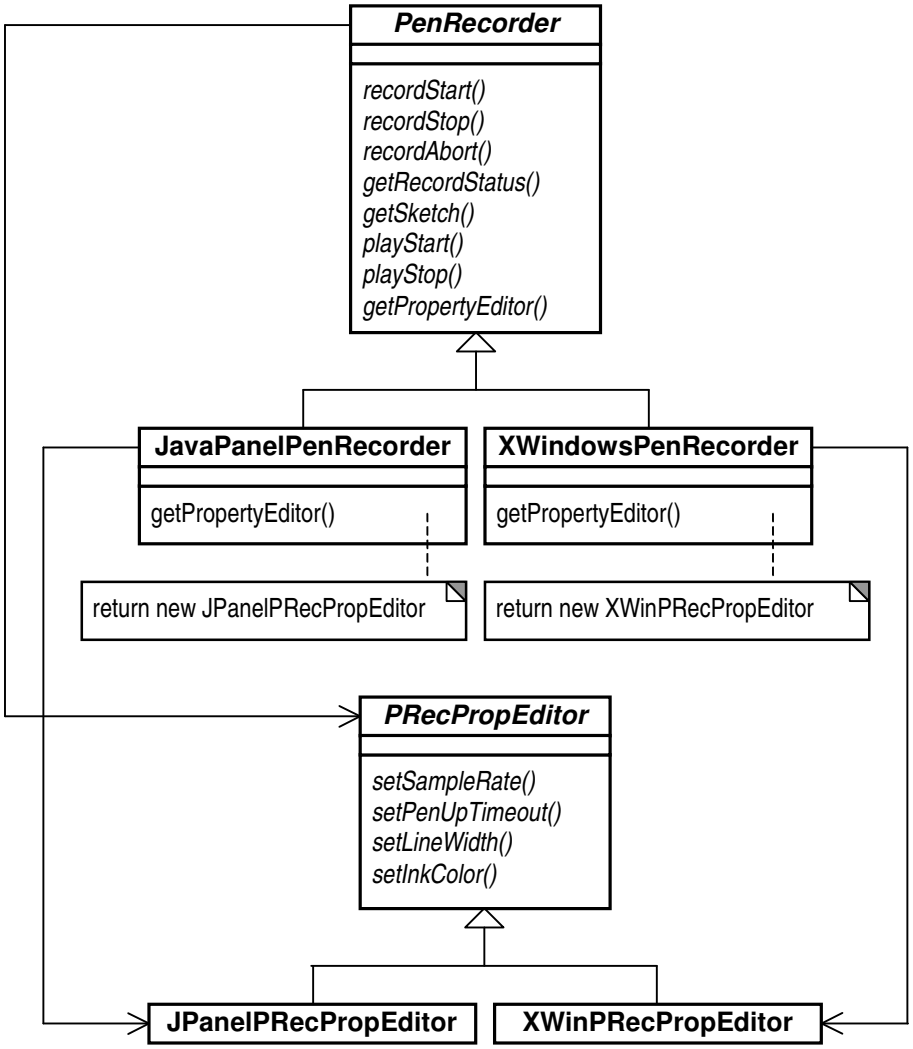


Figure 15. PenRecorder Interface

As in the SpeechRecorder class hierarchy, PenRecorder is an abstract base class that defines the relevant interface implemented by concrete subclasses. It requires an input device capable of delivering a stream of pen position coordinates as well as indications of pen-up/pen-down states (pen strokes are traced only during pen-down periods). Calling recordStart() enables the PenRecorder to retrieve data from the input device, but actual recording does not start until the first pen-down event occurs. Point coordinates and pen-up/pen-down notifications received from the input device are accumulated as pen strokes until either recordStop() is called or the pen remains up for longer than a preset-timeout, indicating that the user has finished drawing or writing. While pen strokes are being recorded, visual feedback is provided in the form of "electronic ink" traces that simulates the experience of scribbling on paper with a real pen. getSketch() retrieves the recorded

pen data after recording stops. Recorded pen strokes can be played back on the screen using playStart() and playStop(). The PRecPropEditor object returned by getPropertyEditor() can change recording parameters and appearance properties of the electronic ink feedback.

Since the default user interface is written in Java and employs the Java Abstract Window Toolkit (AWT), the easiest way to incorporate PenRecorder functionality is to build upon the AWT as well. JavaPanelPenRecorder is implemented as an AWT Panel container that captures mouse clicks and drags as pen input data and draws pen stroke traces on top of every widget (interface components) contained in the Panel. Because any AWT widget can be placed inside the Panel container, pen-input capture capabilities can be trivially added to any part of the user interface. As explained in section 4.5.1, using the mouse as a pen input device automatically provides support for all other pen input devices that can simulate mouse input with appropriate drivers.

XWindowsPenRecorder is a wrapper around XPRecServer (see section 4.5.1). It converts PenRecorder method calls to appropriate service requests transmitted to the server program via TCP/IP sockets. This PenRecorder implementation is useful for X Windows applications that do not use the default Java applet user interface.

4.5.4 The PenRecognizer Interface

The class diagram for the PenRecognizer interface is depicted in Figure 16. The PenRecognizer API is parallel to the SpeechRecognizer API.

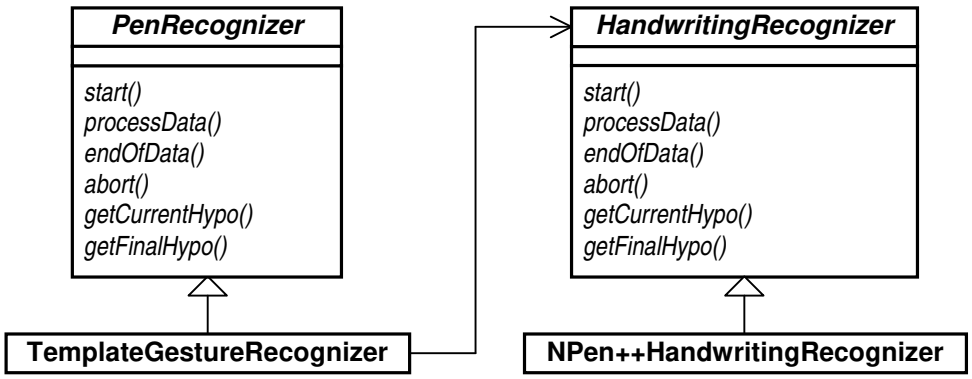


Figure 16. PenRecognizer Interface

TemplateGestureRecognizer is a concrete subclass that adapts the PenRecognizer API to the TmplGRec system described in section 4.5.2. API method calls are converted to service requests and sent via TCP/IP sockets because TmplGRec is installed as a server. The gesture recognizer can be configured to detect potential handwriting strokes and forward them to the NPen++ handwriting recognizer (described in section 4.3.3) through the NPen++HandwritingRecognizer wrapper. Recognition results are sequences of gesture/handwriting components.

4.6 Communication Layer

The distributed modules in MMApp communicate with each other over a communication layer that hides most of the networking details. Inter-module communication follows a client/server model that allows location-transparent access to services. API invocations on distributed components are forwarded across the network as remote service requests.

The communication API is implemented in a Java class library. The functionality of the communication layer is similar to that supported by Java Remote Method Invocation (RMI) or Common Object Request Broker Architecture (CORBA). MMApp incorporates its own communication API instead of relying on RMI or CORBA for the following reasons:

- At the time the first version of MMApp was written, Java RMI had still not been released, and CORBA was not in widespread use;
- RMI supports communication among Java processes only, whereas some components of MMApp are written in C/C++ and Tcl;
- CORBA does support multiple languages but is fairly complicated to use and contains far more capabilities (which carry associated overhead) than MMApp requires.

However, isolating interprocess communication into a separate layer means that future versions of MMApp could be implemented on top of CORBA (or equivalent distributed object models) with little difficulty.

4.6.1 Client/Server Model

In MMApp, access to services such as speech and pen recording and recognition is regulated by objects that implement the Server interface, and components that make use of such services are represented by objects that implement the Client interface. A Client can invoke a Server method to send a request for a particular service and receive a result object in return. A service request can be synchronous, meaning that the method call blocks until the result is received, or asynchronous, in which case the method calls immediately returns a handle that can be used to access the result when it becomes available.

Sophisticated systems for remote method invocation usually include automatic marshalling/unmarshalling facilities which pack and unpack arbitrary method arguments and return values as needed for data transfer. The MMApp communication layer does not handle this automatically; if arguments are required for a particular service request, they have to be packaged as a single request object by the application code.

4.6.2 Remote Service Request

If the actual server program is running in a separate process, Server method calls are forwarded as remote services requests. This is done transparently from the perspective of the Client, which simply invokes Server methods as usual. Underneath, the requests may be serviced by a local object or relayed to another process via an interprocess pipe (if the other process is on the same machine) or a TCP/IP socket (if the other process is on another machine across the network).

Client and Server Proxies

Transparent remote service requests are implemented by Proxy objects that represent a remote Server or Client. A server Proxy implements the Server interface by forwarding requests over a data stream connected to the remote process; a client Proxy does the same for results from those service requests. Proxy is actually an abstract base class; concrete subclasses implement different types of data stream depending on the communication channel (interprocess pipe or TCP/IP socket). The way request and result objects are encoded in the data streams depends on the transport protocol, which can be specified when the Proxy is created.

Transport Protocols

A transport protocol is a specification of how requests and results are encoded to be transmitted over the communication channel. This is represented by a Transport abstract base class, the interface of which includes methods to send and receive Request and Result objects. Different transport protocols are implement as concrete subclasses of Transport. Following the Abstract Factory design pattern (see Appendix C.1), MMApp objects make use of a TransportFactory class which delegates to its subclasses the instantiation of the right Transport subclass. Changing the transport protocol is as simple as specifying a different TransportFactory object during initialization.

Because MMApp includes components implemented in several different programming languages, the default transport protocol is text-based so that Java, Tcl, and C/C++ components can communicate with one another easily. Service requests are simple text strings that specify commands to be executed. This simple scheme suffices for all the applications that have been built so far using MMApp; however, application developers are free to implement other transport protocols if the need arises. For instance, for communication among Java-only modules, a Transport subclass that uses the object serialization mechanism built into Java would be able to transmit and receive arbitrary Java objects instead of just text strings.

4.6.3 Switchboard-Based Communication

The default user interface in MMApp is implemented as an applet (see section 4.7). This creates a small obstacle to distributing components across the network. Due to Java security restrictions in current Web browsers, an applet can only communicate with its

Web server machine. Because distributed MMApp modules may reside on arbitrary machines depending on available resources, the user interface module may not be able to communicate directly with the servers it needs.

To work around this problem, the communication layer includes a central router program called the *Switchboard*, which acts as a naming directory service and a request router. Server processes can register themselves with the Switchboard; client requests are sent to the Switchboard and relayed to the appropriate registered servers identified by their registered names. Running the Switchboard process on the Web server solves the applet security problem because the user interface module in the applet can communicate with the Switchboard, and through it, with all the registered servers.

The API to communicate with the Switchboard is implemented in both Java and Tcl so that every component in MMApp can make use of it, not just the Java components. The Switchboard program itself is written in Java and utilizes multithreading to support multiple clients and servers at the same time.

4.7 Graphical User Interface

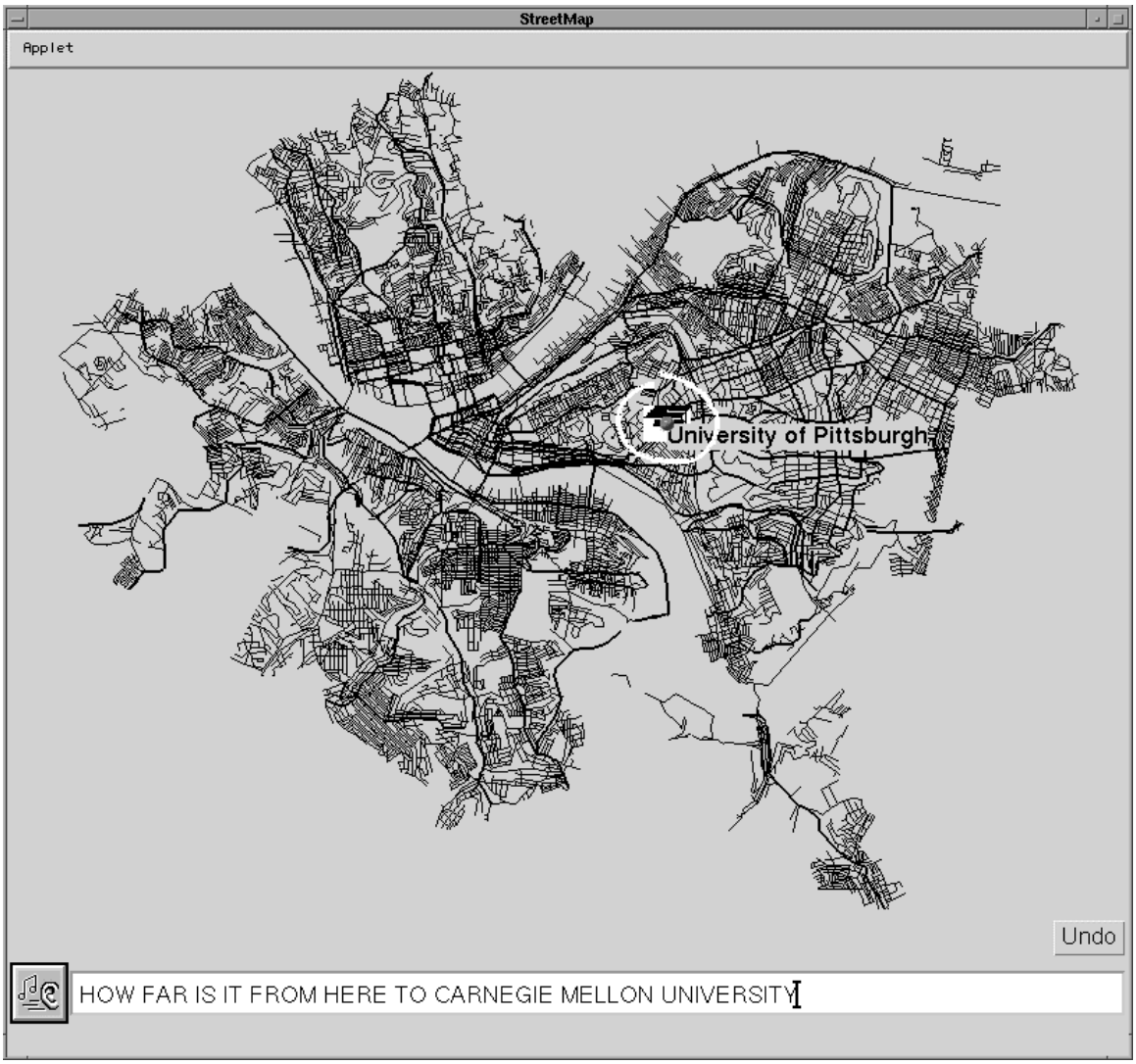
MMApp includes a group of components that can be assembled into a customizable graphical user interface. The visual interface components are implemented in Java to take advantage of platform independence, object-oriented code reuse by inheritance, and World Wide Web deployment capability. The default user interface is a Java applet that can be subclassed to add application-specific functionality. Implementing the user interface as an applet has many advantages:

- Such an applet can be widely deployed over the World Wide Web to take advantage of the distributed architecture of the MMApp framework;
- The plug-in mechanism in modern Web browsers greatly facilitates platform-specific installation procedures such as those needed to install drivers for speech recording;
- The object-oriented nature of Java makes it very easy to customize the user interface for different applications and still inherit basic multimodal input handling capabilities.

Figure 17 on page 74 shows a screen capture of the multimodal applet interface in a map navigation application. Except for the map displayed in the largest part of the window, everything is inherited from the MultimodalApplet base class in the MMApp library. The sketch underneath the picture depicts the layout of various components in the interface.

The next several subsections describe the components that constitute the user interface. Although MMApp includes a default user interface in the form of an applet, the components are fully reusable individually. If required, application developers are free to

create an entirely different user interface and still make use of the components offered by the framework to shorten development time.



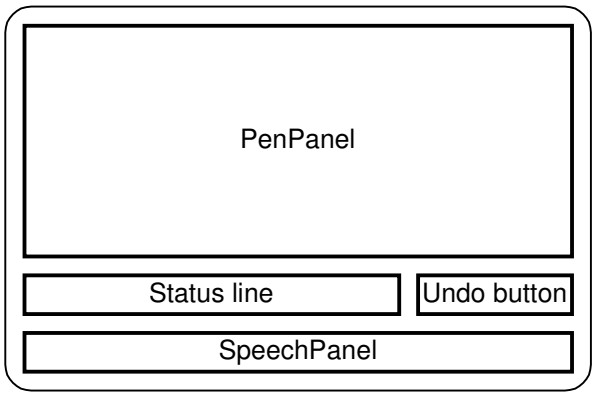


Figure 17. Multimodal Applet User Interface

4.7.1 SpeechPanel

The SpeechPanel is a visual interface component that streamlines access to speech recording/recognition capabilities. It occupies the bottom of the applet window in Figure 17 and consists of a SpeechButton on the left of a hypothesis display. The SpeechPanel is configured with a SpeechRecorder at object construction time and thereafter performs speech recording operations through the SpeechRecorder abstract interface, no matter what concrete implementation of SpeechRecorder is actually used.

The speech recording/recognition process cycles through four states: *inactive*, *listening*, *recording*, and *waiting for result* (see Table 4). The current state is reflected by the icon on the SpeechButton, which provides a visual indicator of the progress of speech input processing. Clicking the SpeechButton normally puts the speech subsystem in *listening* mode, waiting for the user to speak. The speech subsystem enters *recording* mode when the audio signal level indicates the user has started speaking, and stops the recording when a silence of a certain length is detected, as described in section 4.4.2. At this time the SpeechRecognizer is still working on the recorded data, so *waiting* mode lasts until the final recognition hypothesis is produced. During the recording and waiting periods, partial hypotheses reported by the SpeechRecognizer are displayed in the text field beside the SpeechButton, as is the final hypothesis produced at the end.

Icon	State	Description
	inactive	Speech recording is disabled.
	listening	The SpeechRecorder is activated and waiting for the user to speak.
blink	*recording*	The user has started speaking and speech data is being recorded for recognition.
	waiting for result	The user has stopped speaking and the SpeechRecognizer is working to produce the final hypothesis.

Table 4. Speech Recording/Recognition States

The default behavior described above can be changed by configuring threshold and silence time-out parameters. For convenience, the SpeechButton includes an *Auto-stop* property that enables the default behavior when set. If *Auto-stop* is not set, the SpeechButton manipulates SpeechRecorder parameters internally such that pushing the button immediately starts *recording* mode which lasts until the button is pushed again, regardless of the audio signal level. Another SpeechButton property, *Auto-reactivate*, enables continuous operation of the speech recording/recognition functionality by automatically reactivating *listening* mode after the previous utterance has been recognized, instead of going back to the *inactive* state and waiting for the button to be pushed again.

Clicking on the SpeechButton using the right mouse button pops up a visual property editor that allows the user to configure the *Auto-stop* and *Auto-reactivate* properties as well as SpeechRecorder properties (via an implementation-specific SRecPropEditor object as described in section 4.4.2).

SpeechEvent and InputEventCallback

Each time *recording* mode starts, the SpeechPanel generates a SpeechEvent, a subclass of InputEvent. An InputEvent is an abstraction of the data produced by a period of activity in an input channel. Once an InputEvent is generated, it goes through a *recording* state during which data from the input channel is retrieved and processed, and a *waiting for result* state during which the input channel has become inactive but the recorded data is still being processed. Objects holding a reference to the InputEvent can examine its state or retrieve the data processing result. The result of a SpeechEvent is a SpeechResult object containing the text of the final recognition hypothesis.

InputEventCallback is an abstract notification interface that binds the SpeechPanel (and the PenPanel described below) to the rest of the application. Client objects requiring access to the speech recording/recognition facility simply call a SpeechPanel method to register an InputEventCallback to be invoked whenever a SpeechEvent is generated. This is done by the InputCoordinator described in section 4.7.3.

4.7.2 PenPanel

The PenPanel class implements the functionality of JavaPanelPenRecorder (see section 4.5.3) in a subclass of the standard Java Panel container class. Figure 17 on page 74 shows a circling gesture on the PenPanel that contains the map display.

The pen recording/recognition process cycles through four states that parallel the speech processing states: *inactive*, *waiting for input*, *recording*, *waiting for result*. When active, the PenPanel intercepts mouse events and interprets them as pen input data. The first mouse button press in the *waiting for input* state triggers *recording* mode; thereafter button-press events indicate “pen down,” button-release events indicate “pen up,” and mouse drags are recorded as pen strokes. If the pen-up state lasts for longer than a preset interval, the pen gesture is considered complete and the PenPanel stays in *waiting for result* mode until the PenRecognizer finishes processing the gesture. In normal operation, the PenPanel is configured to accept input continuously by going back to *waiting for input* after the final recognition hypothesis has been received.

As a full-fledged container, the PenPanel can hold any Java AWT components, thereby endowing them with pen input capabilities. During *recording* mode, the pen strokes being recorded are drawn on the PenPanel, tracking the mouse movements. Simply drawing on the PenPanel window itself would not work because a child window occupying an area inside a parent window would obscure that area and hide any graphics painted on that part of the parent window. The pen strokes are made visible on top of everything inside the PenPanel by the expedient of recursively drawing them on all the

child components as well as the parent container. Partial recognition hypotheses are also shown by drawing appropriate shapes in a different color from the "electronic ink" color used to track pen strokes. For instance, if a sequence of pen strokes is recognized as a *rectangle*, the hypothesis is shown as a rectangle superimposed on the corresponding strokes. Both the input strokes and the hypotheses are erased after recognition completes.

PenEvent

Similar to the way the SpeechPanel produces SpeechEvents, the PenPanel generates a PenEvent whenever *recording* mode starts. PenEvent is a subclass of InputEvent that produces a PenResult after the input data has been processed by the PenRecognizer. The result of the recognition process is a sequence of GestureComponent objects, each of which carries a shape label (e.g., *circle* or *rectangle*), a list of coordinates that specify the geometry of the shape, and some optional data (e.g., the text of the handwriting recognition hypothesis if applicable).

Client objects requiring access to the pen recording/recognition facility have to register an InputEventCallback to be invoked whenever a PenEvent is generated. This is normally handled by the InputCoordinator described below.

4.7.3 InputCoordinator

The InputCoordinator is a non-visual component responsible for implementing a policy of grouping input events from different modalities (see section 3.2.1). The default policy is the temporal proximity model shown in Figure 2 on page 34.

The InputCoordinator interface includes methods to announce new InputEvents and retrieve combined multimodal input events. The InputCoordinator normally receives notification of newly generated InputEvents via the InputEventCallback interface. If several InputEvents satisfy the criteria for grouping (e.g., if they overlap or occur close together in time, assuming the default InputCoordinator policy is in effect), a combined multimodal event—an array containing the grouped InputEvents—is created and placed on an output queue. Clients of the InputCoordinator retrieve multimodal input events from this queue.

4.7.4 MultimodalApplet

The MultimodalApplet class extends the Java Applet class and provides a structure that binds the previously described components together in a default graphical user interface. The easiest way to construct a graphical user interface for a new multimodal application is to create a subclass of MultimodalApplet and insert application-specific components into it. However, application developers can also assemble the components offered by MMApp into a custom user interface if the default implementation does not suit their needs.

User Interface Layout

The MultimodalApplet has the general layout shown in Figure 17 on page 74. The applet window consists of a SpeechPanel along the bottom edge, a status line and an "Undo" button just above the SpeechPanel, and a PenPanel that occupies the top area of the window. The only application-specific component in the depicted application is the map display inside the PenPanel. Applications constructed using the MultimodalApplet template will have essentially the same layout, with pen-enabled application-specific components nested inside the PenPanel. The insertion of application-specific components should be done in the init() method of the applet.

Interpretation of Multimodal Inputs

For most applications, the process of interpreting user input conforms to the following general outline:

1) Record input events from each of the available modalities;
2) Decide which input events should be grouped to form a combined multimodal input event;
3) Determine the parameterized action that best matches the multimodal input event;
4) Perform the action or report an error if no good match can be found;
5) Undo the effects of the action if the user indicates that it was incorrect.

Although the sequence of steps remains the same across applications, each of the above steps may have to be customized for each application. This situation is ideal for applying the Template Method design pattern (see Appendix C.5). The MultimodalApplet implements the sequence of steps and declares abstract methods for the customizable steps. A MultimodalApplet subclass in a particular application must override the following abstract methods to supply application-specific implementations:

- getInterpretation() accepts a combined multimodal input event (in the form of an array of InputEvents) and return an Object representing the action to perform;
- executeCommand() interprets the Object returned by getInterpretation() as an action command in an application-specific way and carries out the action;
- undoCommand() reverses the effects of a previously executed action.

The customization of input retrieval and grouping policies is delegated to the input recorders/recognizers and the InputCoordinator.

The normal control flow is as follows (see Figure 18 on page 79):

1) The user speaks and/or draws on the screen;
2) The SpeechPanel and/or the PenPanel generate InputEvents;

3) The InputCoordinator groups one or more InputEvents into a multimodal input event (an array of InputEvents);

4) The multimodal input event is passed to getInterpretation(), and the result is relayed to executeCommand();

5) undoCommand() is called if the user clicks on the "Undo" button; otherwise the application is ready to process the next multimodal input event.

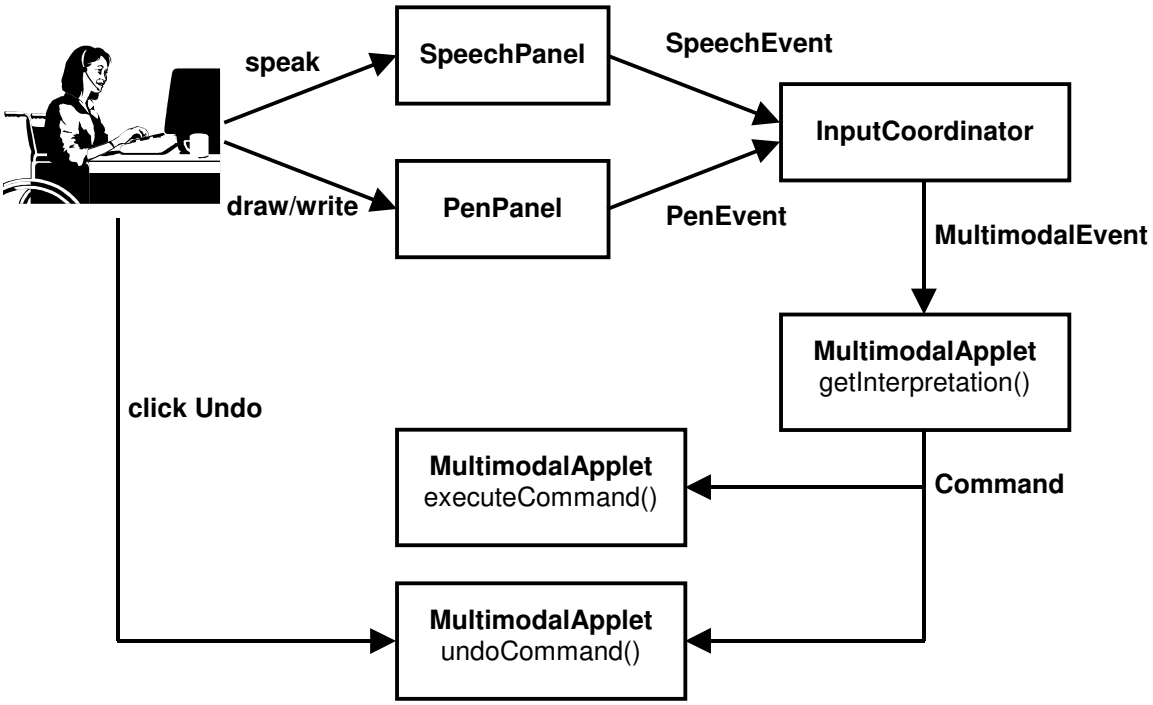


Figure 18. Control Flow for Multimodal Input Interpretation

Any of the above steps may raise an exception if an error occurs, in which case an error message is displayed on the status line and the application is again ready to process the next multimodal input event.

An implementation of getInterpretation() may use the interpretation components described in section 5.6 to preprocess the input data (section 5.6.1), send them to a multimodal semantic integrator (section 5.6.2), and postprocess the result (section 5.6.3) to produce the output action and its parameters. Application developers are also free to introduce other multimodal interpretation schemes.

Customizable Component Instantiation

The MultimodalApplet needs to instantiate several subordinate objects, among them a SpeechRecorder/SpeechRecognizer pair, a PenRecorder/PenRecognizer pair, and an InputCoordinator. Because application developers may decide to use different concrete implementations of these abstract interfaces, the code that instantiates these objects must not be hardwired into the framework. Instead, the actual instantiations are delegated to

MultimodalApplet subclasses using the Factory Method design pattern (see Appendix C.3). For instance, a subclass may override the createSpeechRecorder() method to substitute a different SpeechRecorder implementation for the default PluginSpeechRecorder.

Wizard-of-Oz Support

The Wizard-of-Oz user study technique [Salber93] is widely used to collect user data when a user interface prototype is available for an application but the complete system is not yet functional. In a typical Wizard-of-Oz experiment, a user is asked to perform some tasks using the interface prototype, but the actual operations are controlled by a hidden human operator.

The MultimodalApplet contains logic to connect a controlling applet to the target applet, relay multimodal input events from the target applet to the controlling applet (so that the wizard operator can see what the user draws and hear what the user says), and send operator commands to the target applet to simulate system operations. Because the control flow remains the same but the implementation details vary from application to application, the Wizard-of-Oz support is implemented using the Template Method design pattern (see Appendix C.5). The skeleton algorithm invokes abstract methods that are overridden in MultimodalApplet subclasses to provide application-specific implementations.

Chapter 5
MULTIMODAL DESIGN AND RAPID PROTOTYPING

The software components described in Chapter 4 can be instantiated and interconnected to create a multimodal application instance. To complete the construction of the application, a multimodal interpreter must be instantiated for the target domain so that the application can determine the correct action to perform in response to an input event.

This chapter describes a design process that can be followed to create a working multimodal application within the MMApp framework (see Chapter 4), and the *Multimodal Toolkit* (MMTk), a collection of tools that automate many steps in the proposed design process. For the most part, the tools in the MMTk workbench are implemented in the Java language to maximize platform independence and to make it possible to deploy them on the World Wide Web as applets running inside Web browsers.

The MMTk tools require an MMGL user input model for the target application (see Chapter 3); thus, we need a multimodal grammar implementation that would allow MMGL input models to be created, modified, and stored, as well as an editing tool to design such models. If real data is scarce, it would be useful to be able to generate random samples from the input model following the probability distribution built into it. A tool capable of generating a statistical language model for speech recognition would abate the need to collect enormous amounts of training data in the application prototyping stage. Finally, the components needed to instantiate the multimodal interpretation algorithm based on the MS-MIN are also good candidates for automatic generation.

There are many examples of grammar tools in the literature [Firth91][Erbach92] [Brown94][Shimazu95]. However, several characteristics distinguish MMTk: the integrated handling of multiple input modalities (this is also supported by the MM-DCG translator in [Shimazu95]), the exploitation of the multimodal semantic model built into the foundation of the tools, and the systematic use of the tools in the application design process.

5.1 Design Process

A working application constructed within the MMApp framework has to

- Partition the multimodal input streams into unimodal input events and group them into combined multimodal input events;
- Convert raw signals in individual input events to more convenient symbolic representations (e.g., using speech, gesture, and handwriting recognition);

- Align and jointly segment the token streams in each multimodal input event to construct an action frame composed of parameter slots;
- Extract parameter values from the input tokens inside the parameter slots and construct the complete parameterized action to perform in response to the multimodal input event.

These requirements suggest the following steps to design the application:

1) Determine the *set of commands* to be supported by the application and derive a set of *action frames* and their associated *parameter slots*;
2) Create a multimodal grammar that serves as an *MMGL input model* for the application;
3) Use the MMGL input model to generate *language models* for recognizers in relevant modalities (e.g., a trigram language model for speech recognition);
4) Train an *MS-MIN* to perform *multimodal semantic integration* in the task domain of the application;
5) Implement a *postprocessor* to *extract parameter values* from the result of the multimodal semantic integration process;

Each step in this design process is explored in more detail below.

5.1.1 Selecting Action Frames and Parameter Slots

The set of relevant action frames and their associated parameter slots depends on the set of commands that the application must support. This must be ascertained in the requirement analysis stage of the application design. The parameters needed to execute each command must also be explicitly specified.

Given a command structure for the application, the most straightforward way to proceed is to create one action frame for each supported command. Each parameter of the command then corresponds to a parameter slot in the action frame. It is usually advantageous to combine very closely related operations (e.g., differing in only one parameter) into a single command; however, for the sake of improved interpretation accuracy, it may be necessary to use separate action frames for such operations so that the action frames and their parameter slots are more tightly bound. This was discussed in section 3.2.2, and an example of this will be seen in the map application design presented in Chapter 6.

5.1.2 Designing the Input Model

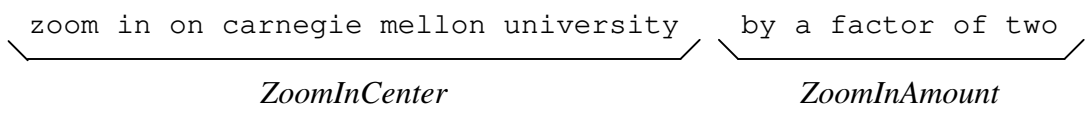
Given the set of action frames produced in the previous step, a skeleton MMGL model can be created with a Toplevel node containing one sequence per action frames and a single AFrame node in each sequence. The weights of the sequences should be the relative frequencies of the actions represented by the corresponding AFrames. This can be estimated from real user data if available (e.g., from simulation studies); otherwise the

application developer must make guesses or keep the default uniform distribution until more data is available.

The next step is to fill the AFrame nodes. The easiest way to proceed is to construct examples of multimodal input events, manually segment and align corresponding parts of the input streams according to the semantics of the application commands, create PSlots and UnimodalNodes from the input segmentation, and generalize the input fragments using a context-free grammar (CFG). PSlot nodes representing instances of the same parameter slot that result from different input messages should be given the same name (the name of the parameter slot) but distinguished by different tags.

This process is best illustrated by an example. Suppose we are building a map application that supports a "zoom in" command among others. The command requires at least one of the following two parameters: a center point for the zoom, and a zoom factor. Accordingly, we create a *ZoomIn* action frame with two parameter slots, *ZoomInCenter* and *ZoomInAmount*. The action frame is represented by a *ZoomIn* AFrame node.

An example of a (speech-only) input message that should be mapped to *ZoomIn* is "Zoom in on Carnegie Mellon University by a factor of two." The input segmentation is



We create a sequence in the *ZoomIn* AFrame and insert two PSlots into it: *ZoomInCenter(0)* and *ZoomInAmount(0)*. The "(0)" tags indicate that this is the first instance of each parameter slot that we have created. Inside *ZoomInCenter(0)* we can now place a UnimodalNode named *ZoomInCenter_Prefix(SPEECH)*; likewise, *ZoomInAmount(0)* contains a *ZoomInAmount_Suffix(SPEECH)* UnimodalNode.

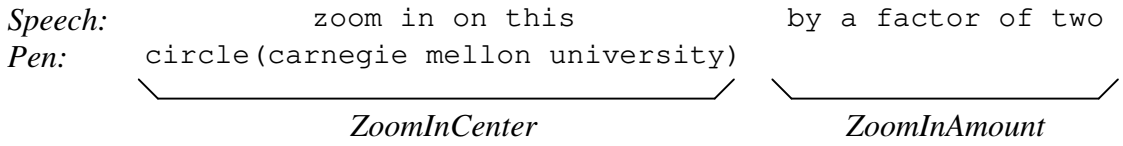
The sentence fragments in the above example can be generalized by generating synonyms or alternative phrasings. For instance, "zoom in on Carnegie Mellon University" suggests the following CFG:

```
ZoomInCenter_Prefix(SPEECH) ::= ZoomInVerb Preposition Place
ZoomInVerb ::= "zoom in" | "magnify" | "enlarge"
Preposition ::= "on" | "at" | "around"
Place ::= "carnegie mellon university" |
          "university of pittsburgh" |
          "duquesne university"
```

Non-terminal symbols (`ZoomInVerb`, `Preposition`, and `Place`) become NonTerm nodes, while the terminal text strings become Literal nodes in the MMGL model. Alternative sequences in NonTerms should be assigned weights proportional to the relative frequencies (estimated from data or simply guessed) of the sequences. For instance, if

"zoom in" is twice as likely to occur as the other two synonyms, it can be assigned a weight of 0.5 while "magnify" and "enlarge" have 0.25 weights[†].

An additional example of speech and pen combination is a spoken utterance "Zoom in on this by a factor of two" accompanied by a circling gesture around the icon for Carnegie Mellon University on the map. The segmentation is



This prompts us to create a new PSlot node, *ZoomInCenter(1)*, containing two UnimodalNodes: *ZoomInCenter_Deictic_Prefix(SPEECH)* and *ZoomInCenter(PEN)*. The *ZoomInAmount(0)* PSlot is reused in a new sequence in the *ZoomIn* AFrame, together with the new *ZoomInCenter(1)* PSlot.

The same generalizing process described previously can be used to derive CFGs for the new UnimodalNodes. We can reuse the NonTerms and Literals already created. For instance, a CFG for the speech fragment in the above example might be

```
ZoomInCenter_Deictic_Prefix(SPEECH) ::=
    ZoomInVerb Preposition Deictic
Deictic ::= "this" | "that" | "here" | "there"
```

The existing `ZoomInVerb` and `Preposition` are reused here.

For pen input, the circling gesture is a deictic expression that has other equivalents, e.g. a pointing gesture or a cross. This could serve as a basis for generalization in a CFG that may look like this:

```
ZoomInCenter(PEN) ::= Pen_Deictic
Pen_Deictic ::= Pen_CircleDeictic | Pen_PointDeictic |
                Pen_CrossDeictic
Pen_CircleDeictic ::= "circle(<ObjectID>)"
Pen_PointDeictic ::= "point(<ObjectID>)"
Pen_CrossDeictic ::= "cross(<ObjectID>)"
```

The encoding of pen gestures together with their spatial contexts (e.g. `<ObjectID>` in the above example) is application-specific and must be selected on a per-application basis with the input model in mind, so that relevant parameters can be extracted later.

[†] Actually the weights do not have to sum to 1; they will be automatically normalized in all probability calculations. Thus a weight of 2 for "zoom in" and 1 for the other synonyms will do as well.

5.1.3 Generating Unimodal Language Models

Because an MMGL grammar is a multimodal input model, it also implicitly defines unimodal language models that cover input messages from each modality. The MMApp framework strongly supports speech and pen modalities. However, pen gesture input is too application-specific and difficult to generalize, while speech language models are better understood. For this reason, the MMTk workbench strongly supports only the automatic generation of N-gram statistical language models for speech recognition, although in theory all unimodal language models are supported by the MMGL formulation.

As discussed in section 2.1.3, an N-gram statistical language model helps guide the search for the correct speech-to-text mapping by predicting the likelihood of encountering a word based on preceding words. A bigram model uses a single preceding word whereas a trigram model uses two preceding words.

N-gram probabilities are normally computed by counting word combinations in a training corpus of input sentences. If real sentences collected from users are not available, we can generate random sentences from an MMGL input model as described in section 5.4. However, this can be avoided by extracting the required probabilistic information directly from the input model.

An MMGL model specifies how non-terminal nodes are expanded and what the probability of each alternative expansion is; therefore, the model also implicitly determines bigram/trigram probabilities for the speech tokens. The Language Model Generator described in section 5.5 computes these bigram/trigram probabilities from an MMGL model, avoiding the need to collect a large amount of real user data or generate an equivalent amount of artificial data to calculate a language model for speech recognition.

It should also be possible to generate a finite-state-grammar language model instead of a statistical one, if such a language model is more suitable for an application. However, the current version of MMTk does not include any tool for this purpose.

5.1.4 Instantiating the Multimodal Semantic Integrator

The multimodal semantic integrator is responsible for aligning and jointly segmenting the token streams in a multimodal input event to produce an action frame—a sequence of parameter slots—as specified in the semantic model (see section 3.2.2).

A domain-independent semantic integration algorithm based on the MS-MIN was described in section 3.4. The MS-MIN can be trained from examples of multimodal input events presented together with their joint segmentation into parameter slots. One obvious way to do this (if real data is lacking) is to generate random input samples from an MMGL model following the probability distribution built into it, then use these samples to train the MS-MIN. It is very convenient (and intentionally so) that MMGL models have built-in semantic information in exactly the form required by MS-MIN training: the

joint segmentation of input streams into parameter slots in an action frame, as modeled by PSlot and AFrame grammar nodes. The Random Sample Generator described in section 5.4 can be used to produce training data for the MS-MIN semantic integrator.

A better way to produce an MS-MIN semantic integrator from an MMGL input model entails computing the connection weights in the MS-MIN from occurrence probabilities of input tokens and output classes (parameter slots). These probabilities are implicitly defined by the probability distribution in an MMGL input model, in the same way that N-gram language model probabilities are. It is thus possible to compute the connection weights directly from the MMGL model without having to generate random samples.

Because the MS-MIN is able to learn incrementally during actual use, an MS-MIN semantic integrator generated from an input model can be further trained using real input data when available.

Input Preprocessing

The input integration process produces a labeled segmentation of multimodal input streams that breaks the input into a sequence of parameter slots. Actual parameter values still have to be extracted from the parameter slots in a postprocessing step as discussed in the next section. This postprocessing could be much simplified if the most basic concepts in the input domain were represented by equivalent classes rather than simple words. For instance, if a parameter slot may contain a number that must be extracted, all the input phrases that represent a number ("one", "two", "twenty three" etc.) could be preprocessed and converted to a single *Number* token with an attached data packet containing the actual number phrase. The advantages of this preprocessing are twofold:

- The MS-MIN semantic integrator can achieve higher accuracy because it only has to learn associations for the *Number* token instead of all the possible number phrases;
- The parameter extraction postprocessing can readily identify which parts of the segmented input contain relevant, extractable information.

The degree of preprocessing may vary depending on the task domain, but there is a tradeoff between complexity and flexibility. If the preprocessing identifies high-level concepts, the complexity of the input space is reduced at the detriment of flexibility because many input/output associations have to be covered by the preprocessor instead of being learned by the MS-MIN. One way to increase flexibility in this case is to base the preprocessor itself on an trainable approach. For instance, Gorin [Gorin96] describes a method to learn salient grammar fragments from data. The current implementation of MMTk does not include a flexible, trainable preprocessor.

MMTk does include a simple state-machine-based preprocessor that can parse input fragments using a context-free grammar (see section 5.6.1). This approach was chosen because it is easy to implement and meshes well with the CFG foundation of MMGL. As this preprocessor can only parse fragments that match the grammar exactly, it should be

used to parse only the most basic concepts of the task domain needed for parameter extraction, e.g., *DayOfTheWeek* in an appointment scheduling task or *CityName* in a map navigation task. The job of matching higher-level constructs in a flexible manner should be left to the MS-MIN semantic integrator which can be trained from actual user input.

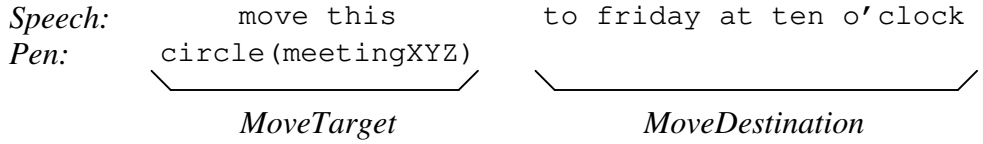
The above discussion pertains mostly to speech input. The encoding of gesture data is too application-specific for a generic treatment. However, the underlying principle still applies: the gesture input streams should contain tokens marked as relevant for parameter extraction. The design example presented in Chapter 6 offers some cogent examples.

Collectively, the input tokens (from any modality) representing combinations of input data (e.g., sentence fragments or groups of gesture symbols) that are relevant to the parameter extraction process are termed *macro concept tokens*.

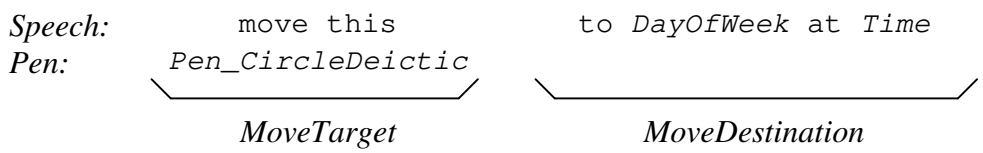
5.1.5 Implementing the Parameter Extraction Postprocessing

The multimodal semantic integrator only segments and labels the input to identify the action frame and parameter slots. The actual parameter values still have to be extracted in a postprocessing phase to complete the multimodal interpretation process. Some part of this postprocessing must necessarily be domain-dependent, but the rest usually consists of repetitive code that branches based on the names of action frames, parameter slots, or preprocessed macro concept tokens.

Consider an illustrative example. Suppose the design of a multimodal appointment scheduler specifies a *MoveMeeting* action frame with two parameter slots, *MoveTarget* and *MoveDestination*. The utterance "Move this to Friday at ten o'clock" and the accompanying circling gesture around a meeting displayed on the calendar interface are semantically integrated as follows:



Assume that the preprocessed macro concepts in this application include the tokens *Pen_CircleDeictic* (indicating that a circling gesture is drawn around an object on the calendar interface), *DayOfWeek* (representing the phrases "Monday" through "Sunday"), and *Time* (representing phrases such as "ten o'clock," "noon," "four in the afternoon," and so on). The actual output of the semantic integrator is



The postprocessor can now traverse this output and produce actual parameter values. Inside the *MoveTarget* parameter slot, the postprocessor expects macro concept tokens

that identify attributes of a meeting; it checks for those and, upon encountering *Pen_CircleDeictic*, branches to the code that extracts the meeting ID "`meetingXYZ`" from the data attached to the macro concept token. Inside the *MoveDestination* parameter slot, the postprocessor again checks for expected macro concept tokens that identify the date and time of the move destination. The presence of *DayOfWeek* and *Time* invokes the appropriate code that extracts "`friday`" and "`ten o'clock`" which are transformed to internal date and time representations.

Given an MMGL input model, the list of macro concept tokens to expect inside each parameter slot can be determined. Based on this information, it is possible to generate a code skeleton that traverses the output from the semantic integrator, examines macro concept tokens, and branches to appropriate subroutines that actually compute parameter values. Only the lowest level of processing that depends on application-specific data representations (e.g., meeting ID, date, time, etc.) must be customized.

The process of generating the code skeleton is automated by the Postprocessor Generator described in section 5.6.3.

5.2 Multimodal Grammar Implementation

The multimodal grammar structure described in section 3.3 is object-oriented in nature and therefore can be easily implemented using an object-oriented programming language such as Java. Figure 19 shows the resulting class hierarchy (see Appendix B for a summary of the UML symbols in the class diagram). The concrete classes Toplevel, AFrame, PSlot, UnimodalNode, NonTerm, Literal, and Sequence directly implement the corresponding grammar components defined in section 3.3.

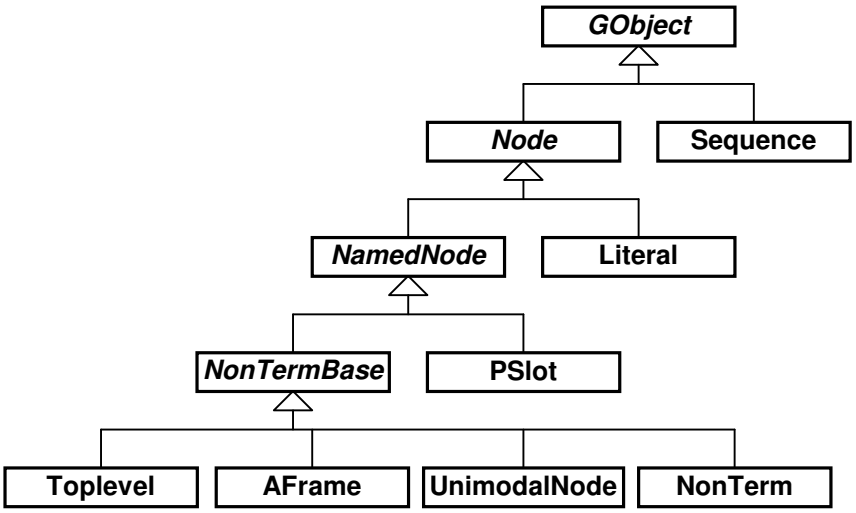


Figure 19. Class Hierarchy for Multimodal Grammar Components

The abstract base classes (indicated by italicized names) introduced into the class hierarchy permit a large amount of common code to be factored out and reused. The

common functionality of all node types is factored out and put into a common root class, GObject, which presents a uniform interface to all external entities that have to manipulate grammar objects. Toplevel, AFrame, UnimodalNode, and NonTerm are the only nodes that contain sequences; accordingly, the shared code for manipulating sequences is inherited from the abstract base class NonTermBase. The Sequence class manipulates Node objects without having to know their exact types; the correct code is selected at runtime via polymorphism.

A user input model is a grammar created by instantiating the nodes and sequences comprising it. The result is a graph of object instances connected by references representing *contains* and *is-part-of* relationships; for instance, in a map navigation application, a *ZoomInAmount* PSlot may be part of a Sequence contained in a *ZoomIn* AFrame. The expansion of a grammar object into its constituents corresponds to a grammar production rule. The grammar can be written to mass storage by serializing its object graph, i.e., by traversing the graph and writing out the content of each object encountered during the traversal.

Optional and Repeating Elements

There exist syntactic descriptions of context-free grammars that include notations for optional or repeating elements. These are only "syntactic sugar" notations that are convenient but not necessary, as they can be expressed using only the basic notations. For example, a node in the multimodal grammar formulation can be made optional by replacing it with a NonTerm containing an empty sequence and a second sequence that includes the original node. Similarly, a repeating node is equivalent to a NonTerm that recursively references itself. The current implementation of the multimodal grammar structure does not include shortcut notations; instead, the grammar writer must explicitly use the equivalent constructs shown in Figure 20.

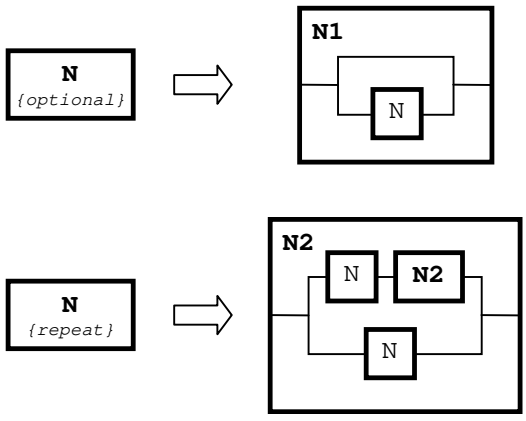


Figure 20. Implementing Optional and Repeating Grammar Nodes

Grammar Traversal

Most of the grammar-based tools described in later sections do their work by traversing the grammar graph and performing some operation on each component. If this

were implemented as a polymorphic recursive method† on the GObject class as shown in Figure 21, it would be necessary to add one method for each supported operation.

```
public class GObject {
   // ...
   public abstract void doOperation();
}
public class Sequence extends GObject {
   // ...
   public void doOperation() {
      // perform operation on this sequence
      // ...
      for (int i=0; i<getNodeCnt(); i++) {
         // traverse each node in this sequence
         getNodeAt(i).doOperation();
      }
   }
}
public class Node extends GObject {
   // ...
}
public class NonTermBase extends Node {
   // ...
   public void doOperation() {
      // perform operation on this node
      // ...
      for (int i=0; i<getSequenceCnt(); i++) {
         // traverse each sequence in this node
         getSequenceAt(i).doOperation();
      }
   }
}
// ...
```

Figure 21. Grammar Traversal with Polymorphic Recursive Method

A better alternative is to apply the Visitor design pattern (see Appendix C.6), which is ideal for this situation because the number of grammar object types is fixed whereas the number of possible operations is unlimited. The list of operations that need to be supported includes loading and saving grammar objects as well as generating various types of information from a grammar: language models, random samples, preprocessors, integration networks, etc.

In the Visitor pattern, the only necessary modification to the target object hierarchy (GObject and its subclasses in this case) is a polymorphic method that accepts an instance of an abstract Visitor class. Using a well-know object-oriented technique called *double-dispatch*, the same method invocation ends up calling different functions depending on

† *Polymorphic* means that the method is declared at the top of the class hierarchy and overridden in subclasses, so that the same call may invoke different methods depending on the actual object type. *Recursive* means that the method calls itself on the constituents of an object to traverse the object graph.

the actual types of two objects: the target of the visit and the Visitor instance. Different operations are implemented by different subclasses of the Visitor base class. The Visitor pattern forms the basis for the implementation of many algorithms that work on multi-modal grammars, without requiring *ad hoc* modifications of the GObject class hierarchy.

```
public abstract class GObject {
   // ...
   public abstract void accept(Visitor v);
}
public class Sequence extends GObject {
   // ...
   public void accept(Visitor v) {
      v.visitSequence(this);
   }
}
public class NonTerm extends NonTermBase {
   // ...
   public void accept(Visitor v) {
      v.visitNonTerm(this);
   }
}
// ...
public abstract class Visitor {
   public abstract void visitSequence(Sequence s);
   public abstract void visitNonTerm(NonTerm n);
   // ...
}
public class Operator extends Visitor {
   public void visitSequence(Sequence s) {
      // perform operation on this sequence
      // ...
      for (int i=0; i<s.getNodeCnt(); i++) {
         // traverse each node in this sequence
         s.getNodeAt(i).accept(this);
      }
   }
   public void visitNonTerm(NonTerm n) {
      // perform operation on this node
      // ...
      for (int i=0; i<n.getSequenceCnt(); i++) {
         // traverse each sequence in this node
         n.getSequenceAt(i).accept(this);
      }
   }
   // ...
}
```

Figure 22. Grammar Traversal with Visitor

Figure 22 shows a Visitor implementation that performs the same operation as the code in Figure 21. The difference is, to implement a second operation using the approach in Figure 21, we would have to declare another abstract method in GObject and implement the method in GObject subclasses; with the Visitor approach, it would only be necessary

to create another Visitor implementation without changing the GObject hierarchy at all. The same thing applies if a different traversal algorithm is required, e.g., one that traverses sub-objects first before performing the operation on the containing object.

5.3 Visual Grammar Designer

Traditional grammars are usually represented textually in some descriptive language such as Backus Normal Form (BNF) or Phoenix [Ward91]. It is rather difficult to follow these textual descriptions at a glance and keep track of grammar production rules, especially as the size of the grammar increases. A graphical display that represents the grammar components visually as in Figure 4 on page 42 makes it much easier to understand the grammar by visual examination. Furthermore, creating or editing a large grammar in textual form usually requires the skills of a computer programmer; in contrast, designing a grammar visually by dragging and dropping graphical components is much more intuitive and requires less training. This section describes an object-oriented, drag-and-drop grammar editor that employs exactly this visual construction paradigm.

5.3.1 Graphical Display of Grammar Components

Following a well-established design rule, the graphical user interface (GUI) elements are cleanly separated from the underlying grammar representation using the Observer design pattern (see Appendix C.4), similar to the model-view approach in the Smalltalk programming environment. Each GObject is an "observable" or "model" having one or more associated "observers" or "views", represented by subclasses of a GObjectView root class (Figure 4 on page 42 shows some views captured from a computer display). The views know how to update themselves whenever the "observable" or "model" object broadcasts a change in its data. External entities do not need to know about views; when they manipulate the underlying grammar structure the screen will be automatically updated.

The views for Node objects can be expanded to show the internal structure or collapsed to display only the node labels. This way the overall grammar structure can be grasped instantly while still allowing for detailed examination of any section, down to the level of Literals. When connected nodes are expanded, phrases modeled by the grammar can be easily read off the display as in part d) and e) of Figure 4.

5.3.2 Drag-and-Drop Editing

The view objects provide convenient handles to manipulate grammar entities visually. It is relatively easy to implement a drag-and-drop GUI in which the handles may be moved around by moving the mouse while holding down a button ("dragging"), and inserted into other objects by letting go of the mouse button when the cursor is over the desired location ("dropping"). This kind of direct manipulation is ubiquitous in modern GUIs and familiar to most computer users. It permits the rapid construction of a grammar with convenient, continuous visualization of various grammar parts and their relationships.

The Multimodal Grammar Designer program supports exactly this kind of grammar construction and editing. Figure 23 is a screen capture of the Designer interface.

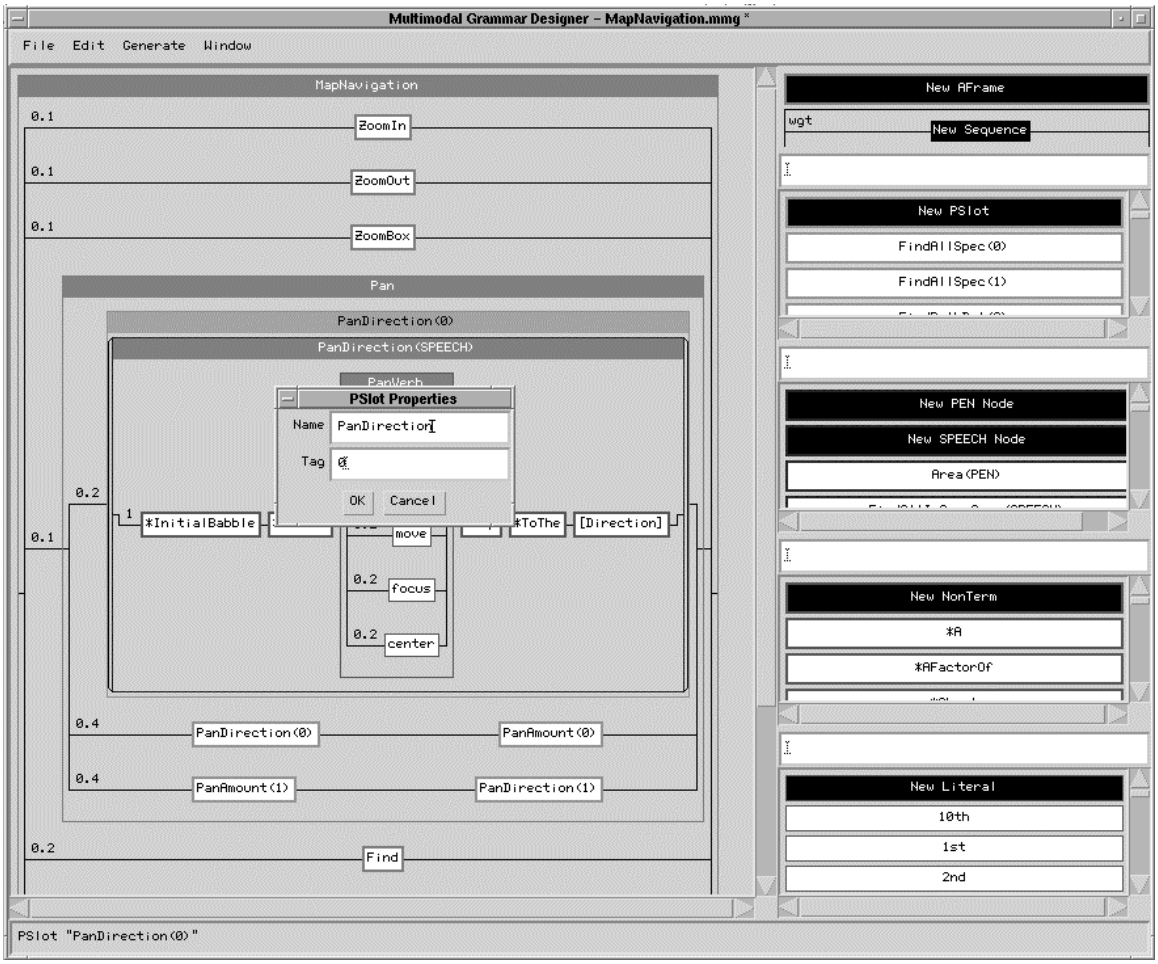


Figure 23. Multimodal Grammar Designer

The main window shows a graphical grammar display. Views of different node types are color coded so that the type of any grammar entity can be grasped instantly. Expanded and collapsed states of grammar nodes are also distinguishable visually. Double-clicking on any node expands the node or collapses it if the node is already expanded. Clicking with the right mouse button displays a context-sensitive popup menu tailored to the object beneath the cursor. Certain menu items are common across all objects, such as a “Properties” menu item that pops up a dialog allowing the user to modify object attributes (the Properties dialog for a PSlot is shown in the center of Figure 23).

On the right side of the Designer window is a palette of grammar object “prototypes” that can be used to construct a grammar visually. The prototypes are also color coded and grouped by object types. The prototypes at the top of each group allow the creation of new objects, while the other prototypes correspond to existing objects and permit their reuse in difference places. The prototypes can be “grasped” and dragged with the mouse. The cursor changes shape when a dragged prototype passes over potential drop sites to

indicate whether a drop at that location will be allowed. For example, AFrame can only be dropped into Toplevel, PSlot can only be dropped in AFrame, and so on. The object-oriented nature of the program makes it easy to enforce this kind of behavior with very simply code. Visual feedback in the form of a dashed line indicates where the new object (or reference to an existing object) would be inserted in the drop target. When a newly created object is dropped, a Properties dialog pops up to allow the user to change object attributes from the default values. After the object has been successfully added to the grammar, this dialog (see Figure 23) can be accessed again at any moment by clicking with the right mouse button and choosing "Properties" from the context menu, as previously described.

5.4 Random Sample Generator

Components of multimodal applications usually need to be validated by computing certain evaluation functions over a set of test input data. If little or no actual data is available, as may be the case during the construction of an application prototype, it is still possible to obtain a set of artificial data that reflects a user input model constructed by the application developer. This kind of model is precisely what MMGL grammars are supposed to encode. Using the probability distribution represented by sequence weights, we can generate random input samples that follow such a distribution.

The sample generator is a Visitor subclass (see section 5.2) that traverses the grammar graph of an MMGL input model and selects sequences at random. For each non-terminal node, the weight of each sequence is divided by the total weights of all sequences in the node to produce the selection probability. The literal tokens from selected sequences are concatenated to form a token stream for each modality. The output is encoded in a format that retains the parameter slot segmentation information in case this may be useful to the evaluation procedures that process the generated data.

5.5 N-gram Language Model Generator

As mentioned in section 2.1.3, a large vocabulary, continuous speech recognizer is usually customized for a particular task domain using a statistical language model, normally an N-gram model with N=2 (bigram) or N=3 (trigram). This section concerns the generation of trigram language models, which subsume bigram models.

Trigram language models are normally generated by counting trigrams[†] (sequences of 3 words) in a training corpus and computing bigram/trigram probabilities from the trigram count table. Generating the language model directly from an MMGL input model means performing the equivalent of generating a very large number of random samples from the input model to form the training corpus. Because generating random samples

[†] Counting trigrams automatically gives unigram and bigram counts as well.

involves traversing the grammar structure, we can also count the trigrams during the traversal and avoid the need to store any sample.

If we imagined generating an enormous number of random samples and dividing the number of times a certain trigram occurs by the number of samples, the limit of this ratio as the corpus size tends to infinity is called the *trigram weight*. Using the sequence weight distribution built into the grammar, we can compute an exact trigram weight for each trigram permitted by the grammar without having to generate any random samples. The N-gram probabilities can be computed directly from the trigram weights instead of trigram counts because we are basically scaling the counts by the number of samples.

5.5.1 Basic N-gram Counting Algorithm

For each object (node or sequence) in an MMGL grammar, we want to compute the weights of all the trigrams that can occur if we were to generate random samples from the grammar object. The list of all trigrams for the grammar object and their associated trigram weights forms a *trigram weight table*.

It is possible to compute a trigram weight table for each grammar object by recursively combining the trigram weight tables of its sub-objects. Because each node is composed of weighted alternative sequences, and each sequence is a series of nodes, three basic operations on trigram weight tables can be defined:

1) *Scaling*. This operation multiplies all trigram weights in a table by a scaling factor representing the probability that a certain sequence in a node would be randomly selected if we were to generate random samples from the grammar. This probability is simply the weight of the sequence divided by the total weight of all sequences in the containing node.

2) *Merge*. This operation takes the trigram weight tables for two sequences in a node and combines them to form a new trigram weight table that contains all the trigrams in the original tables. Weights for trigrams that occur in both component tables are added because if we were to generate random samples from the grammar, the trigram counts for samples that contain either of the two alternative sequences would accumulate additively.

3) *Concatenation*. This operation produces the trigram weight table for two nodes in a series by concatenating trigrams from the two nodes to form new trigrams. Two trigrams can be concatenated if one occurs at the end of the first node and the other occurs at the beginning of the second node. The weights of the concatenated trigrams are multiplied together. To see why this is the case, let us assume N_1 samples are generated from the first table, so that a trigram with weight w_1 can be expected to occur $N_1 w_1$ times. Similarly, a trigram with weight w_2 can be expected to occur $N_2 w_2$ times in N_2 samples. The concatenation produces $(N_1 w_1)(N_2 w_2)$ occurrences of the resulting trigram in $N_1 N_2$ samples, hence the trigram weight in the new table is $w_1 w_2$.

The language model generator is a Visitor subclass (see section 5.2) that does the following for each grammar object it visits:

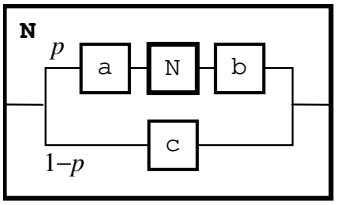
- If the object is a Literal, count the trigrams that occur in the Literal's text;
- If the object is a PSlot, visit the UnimodalNode for the designated modality (usually speech);
- If the object is any other non-terminal node, visit each sequence of the node in turn and merge the resulting trigram weight tables;
- If the object is a Sequence, visit each node in the Sequence and concatenate the resulting trigram weight tables, then scale the result by the normalized weight of the Sequence.

Nodes can be reused in different parts of the grammar; therefore, the trigram weight table for each node is computed only once, then cached and reused it if the node is visited again.

5.5.2 Handling Recursive Grammar References

The above algorithm works if the grammar is finite-state (i.e., any grammar traversal will terminate in a finite number of steps), but goes into an infinite loop as soon as there is a recursive reference to a node. The only practical way of avoiding this infinite loop is to stop the recursion at some depth. In that case the computed trigram weights are no longer exact, but if the recursion were allowed to go deep enough, the multiplicative effects of scaling would reduce the probabilities to such small values that the depth-limited computation could produce results with any desired accuracy.

Consider the following grammar node:



This node represents sentences of the form $\texttt{a}^n\texttt{cb}^n$ for all integers $n \geq 0$. The possible trigrams are: <s><s>a, <s><s>c, <s>aa, <s>ac, aaa, aac, acb, cbb, bbb, cb</s>, bb</s>, and <s>c</s>, where <s> and </s> denote the beginning- and end-of-sentence markers. (Imagining that each sentence starts with two beginning markers and ends with one end marker makes trigram counting more regular.) It is easy to see that the trigram weight for <s><s>a must be p and the weight for <s><s>c must be $1-p$, for example, but the trigram weight for aaa is distributed across all sentences $\texttt{a}^n\texttt{cb}^n$ for $n \geq 3$. For a specific n the trigram aaa occurs $n-2$ times in the sentence, and the sentence has an occurrence probability of $p^n(1-p)$, hence the exact trigram weight is

the sum of the infinite series

$$w = \sum_{n=3}^{\infty} (n-2)\, p^n (1-p) = \frac{p^3}{1-p} \qquad (15)^{\dagger}$$

If we stop the recursive expansion at depth $n_{\max}$, we get the partial sum for $n = 3 \ldots n_{\max}$. One more level of expansion would add $(n_{\max} - 1)\, p^{n_{\max}+1}(1-p)$ to the partial sum. We can stop if this new contribution divided by the new partial sum is smaller than an accuracy threshold, say 10^{-6}.

In general recursive references can be arbitrarily complex (e.g., more than one object can recursively reference each other, or one object can contain multiple recursive references), hence it is not possible to reduce all cases to compact formulas such as the one in Equation (15). To derive a general stopping heuristic, we observe that the summation in Equation (15) simply accumulates sentence probabilities. In the general case the sentence probabilities is not as simple as $p^n(1-p)$; however, we can keep track of the traversal path and compute the sentence probabilities using a running product of normalized sequence weights. In the above example we first traverse a sequence of weight $1-p$ before encountering n times a sequence of weight p, hence the running product is $p^n(1-p)$. Using the sentence probabilities to estimate the contribution of a recursive expansion to the trigram weights, we can stop the recursion when the relative contribution becomes smaller than the accuracy threshold.

The above heuristic will not break the infinite loop if the recursion does not "bottom out" (e.g., if the above grammar example has no `c` but only the single sequence `aNb` with unit probability). In this case the trigram weights are not well defined anyway, so it is reasonable to supplement the stopping heuristic by imposing an absolute maximum recursion depth and raising an exception if the recursion unwinds completely without producing any trigrams.

---

† The sum of this infinite series is derived from the series $\sum_{n=1}^{\infty} np^n = \frac{p}{(1-p)^2}$, which is derived in turn from the geometric series $\sum_{n=0}^{\infty} p^n = \frac{1}{1-p}$ for $0 < p < 1$.

5.5.3 Computing N-gram Probabilities

A trigram language model contains unigram, bigram, and trigram penalties, which are $-\log_{10}$ of probabilities. Given a trigram count table, the various N-gram probabilities can be computed as follows:

$$P_{unigram}(a_n) = \frac{count(a_n)}{\sum_i count(a_i)}$$

$$P_{bigram}(a_m a_n) = \frac{count(a_m a_n)}{\sum_i count(a_m a_i)}$$

$$P_{trigram}(a_m a_n a_p) = \frac{count(a_m a_n a_p)}{\sum_i count(a_m a_n a_i)}$$

Because the N-gram weights computed by traversing the grammar are equivalent to the ratios of N-gram counts to the training corpus size, the weights can be directly substituted for the counts in the above equations (the effect of this substitution is simply to divide both the numerator and the denominator by the same factor, the corpus size).

Each unigram also has a back-off value used to estimate the probabilities of bigrams that did not occur in the training corpus. Similarly unknown trigram probabilities are estimated from bigram back-off values. This back-off scheme smoothes out the N-gram distributions and lets the speech recognizer accept slight variations of sentences permitted by the grammar. The back-off values are computed using an absolute discount scheme in which a fixed discount (0.5 in the current implementation) is subtracted from each N-gram count to form a count for the unseen N-grams. Because this operation requires an actual count, we have to supply an *equivalent corpus size* which is then multiplied with the N-gram weights to produce the equivalent counts. A reasonable equivalent corpus size can be automatically computed to give the smallest trigram weight an equivalent count of 1, i.e., to produce an equivalent training corpus in which each trigram allowed by the grammar appears at least once.

5.6 Interpretation Engine Builders

The design process described in section 5.1 outlines a three-step process to interpret a multimodal event. Speech and pen recognition results are first preprocessed to reduce the complexity of the input space. A semantic integrator segments and aligns the preprocessed streams to produce parameter slots. A postprocessing step extracts actual parameter values from the parameter slots and constructs the action to be performed by the application interface. The three steps in the multimodal interpretation process require application-specific instantiations of the components described below. Given an MMGL input model for the target application, these instantiations can be largely automated, although some amount of domain-dependent manual customization is unavoidable.

5.6.1 Input Preprocessor Generator

The input preprocessor's responsibility is to identify input fragments that should be converted to macro concept tokens (see section 5.1.4). MMTk includes an input preprocessor implementation based on a simple state-machine parser that identifies particular word groups in the target input stream (this is usually the speech stream but the same algorithm works for any modality that needs preprocessing in this manner). Figure 24 shows a state machine that accepts either "zoom in" or "zoom out" optionally preceded by "please."

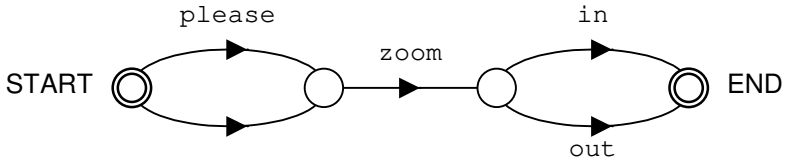


Figure 24. Example of State Machine for Input Preprocessing

To generate a preprocessor from an MMGL input model, NonTerm nodes corresponding to macro concept tokens must be marked as "parseable." The Input Preprocessor Generator is a Visitor subclass (see section 5.2) that traverses the MMGL grammar structure to generate a state-machine-based matcher for each "parseable" node. The resulting collection of state machines can be used to match fragments of the input stream and convert them to macro concept tokens.

The input preprocessor generates a parse tree for each macro concept token it finds. Figure 25 shows an example of a parse tree for a *Number* macro concept, generated from the fragment "nineteen ninety eight." The parse tree is encoded within the macro concept token to facilitate the extraction of the parameter value in the postprocessing phase.

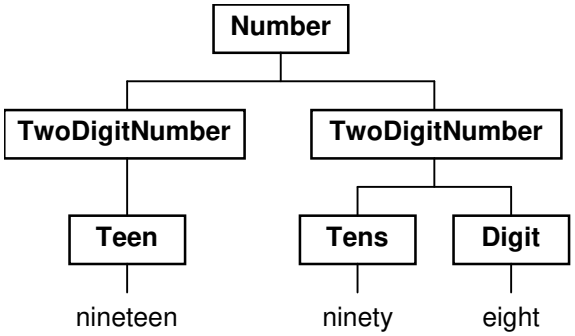


Figure 25. Preprocessed Parse Tree for a Macro Concept

5.6.2 Integration Network Generator

The Multi-State Mutual Information Network described in section 3.4.2 has connection weights that can be computed from input-output occurrence probabilities in a training corpus. The input-output occurrence probabilities are estimated by counting occurrences

of input tokens or fragments and output classes in the training corpus. The similarity with trigram counting immediately suggests a grammar traversal algorithm that computes the probabilities directly without generating any random samples, as with the N-gram Language Model Generator in section 5.5.

The limit of the ratio between an occurrence count and the number of samples, as the number of samples tends to infinity, is an *occurrence weight*. For each grammar object in the MMGL input model, we define an *occurrence weight table* similar to the trigram weight table for language model generation. The occurrence weight table for a grammar object can be recursively computed from the weight tables of its sub-objects, using an algorithm very similar to the N-gram counting algorithm in section 5.5.1. One major difference is we do not have to descend inside NonTerm nodes marked as "parseable" because each of these nodes corresponds to a macro concept that will appear in the input stream as a single token. A second difference is the treatment of PSlot nodes: because the parameter slots correspond to the output classes in the MS-MIN, input-output co-occurrences must be counted at the PSlot level.

The above algorithm is implemented in a Visitor subclass (see section 5.2) that traverses the MMGL grammar structure and recursively computes the occurrence weight tables, then generates an MS-MIN by calculating the connection weights from the occurrence weight tables.

In one experiment, this network generator took 2 minutes to produce a network which would have required the equivalent of 6 million training examples and more than 11 hours of training time to cover all possible input combinations.

5.6.3 Postprocessor Generator

As explained in section 5.1.5, the postprocessor is responsible for extracting parameter values from the output of the semantic integrator. A skeleton of the postprocessing algorithm can be automatically generated from the MMGL grammar structure. Another Visitor subclass (see section 5.2) accomplishes this by traversing the grammar and writing out template code customized by the grammar context at each node.

The output of the Postprocessor Generator is a Java class that contains methods to identify the interpretation context (e.g., the current action frame and parameter slot) and branch to the appropriate postprocessing code. The application developer only has to fill in the domain-dependent parts of the postprocessing template (the parts that handle application-specific data representations). To avoid losing the modifications if the template code is regenerated, the class produced by the Postprocessor Generator should be retained unchanged and the modifications should be put in a subclass by overriding the appropriate methods.

Figure 26 on page 101 shows a code excerpt from a generated postprocessor class. The code comes from the QUICKTOUR application described in Chapter 6.

```
public abstract class StreetMapInterpBase
extends MMInterp
{
  protected void interpretAction(String action, ParsedPSlot[] psSeq,
                                 MMFrameSet fs)
    throws MMInterpException
  {
    if (action.equals("Find"))
      actionFind(psSeq, fs);
    else if (action.equals("FindAll"))
      actionFindAll(psSeq, fs);
    //...
  }
  protected void actionFind(ParsedPSlot[] psSeq,
                            MMFrameSet fs)
    throws MMInterpException
  {
    for (int i=0; i<psSeq.length; i++)
      {
      String psName = psSeq[i].getName();
      if (psName.equals("FindSpec"))
        psFind_FindSpec(psSeq, i, fs);
      //...
      }
  }
  protected void psFind_FindSpec(ParsedPSlot[] psSeq, int index,
                                 MMFrameSet fs)
    throws MMInterpException
  {
    Modality[] modList = psSeq[index].getModalityList();
    for (int i=0; i<modList.length; i++) {
      if (modList[i].equals("SPEECH")) {
        for (j=0; j<psSeq[index].getChildrenCnt(modList[i]); j++) {
          ParsedNonTerm nt = psSeq[index].getChildAt(modList[i], j);
          String ntName = nt.getName();
          if (ntName.equals("PlaceName"))
            paramFindSpec_PlaceName(nt, fs);
          //...
        }
      } else if (modList[i].equals("PEN")) {
        //...
      }
    }
  }
  protected void paramFindSpec_PlaceName(ParsedNonTerm nt,
                                         MMFrameSet fs)
  {
    // application-specific code to be filled in by subclass
  }
  //...
}
```

Figure 26. Postprocessor Skeleton for Parameter Extraction

Chapter 6
DESIGN EXAMPLE: A MAP SYSTEM

This chapter describes the design and implementation of a multimodal application using the MMApp framework and the MMTk workbench. The construction of this application revealed many insights into the requirements of multimodal applications and helped document many design choices that had not been made explicit during the implementation of MMApp and MMTk. The target application, QUICKTOUR, is a map navigation system that allows the user to manipulate a map display and issue multimodal queries.

The QUICKTOUR application described here has been deployed in user observation sessions in which participants interacted with the program to carry out assigned tasks by speaking and drawing. These sessions demonstrated that an application designed and implemented with MMApp and MMTk does indeed work as expected, letting real users accomplish useful tasks using multimodal interaction. Detailed evaluation results are reported in the next chapter.

6.1 Requirement Analysis

QUICKTOUR should display a map of a certain geographical area and allow the user to perform the following operations using speech and/or pen input:

1) *Adjust the map view*. The map display should support the two fundamental map manipulation operations:

   - *Zoom* — change the magnification factor of the map view. It should be possible to zoom in (increase the magnification) or zoom out (decrease the magnification) by a given factor. The user can optionally specify a point on the map as the zoom center, i.e., the point that will be displayed at the center of the map view after the zoom operation. The change in magnification factor can be alternatively specified as a rectangular area of a certain size surrounding the zoom center.
   - *Pan* — shift the map view while retaining the same magnification factor. The user should be able to pan the map along the basic compass directions (north, east, northeast, etc.) by an amount given in units of distance (miles or kilometers) or as a percentage of the screen[†].

[†] Precise zoom and pan parameters may seem out of place in a tourist map application. However, this application was based on a military map application in a previous project. The ability to zoom and pan by precise amounts was required for that project and the requirement was transferred to this project even though the application domain now concerns travel information for tourists.

2) *Search for places on the map*. The possible search criteria include street addresses, place names, establishment types (bank, restaurant, theater, etc.), and other attributes (price range, quality of food, etc.). The search may be restricted to a given rectangular area on the map. The user can specify that all matching locations should be found or only the one closest to the current position marker. The map program should display the matching locations and adjust the map view if necessary to bring those locations into view.

3) *Find the best route from one place to another*. For this application the best route is defined to be the shortest path along the street segments between the two places. The user can also ask for the length of this path or the time needed to traverse the path.

Places on the map can be specified verbally by name or visually by pointing or circling (if they are visible in the current view). Zoom target areas as well as areas used to restrict search operations can be visually delimited by drawing a contour on the screen. A line or an arrow joining two places indicates a request for the best route, optionally accompanied by a verbal qualifier asking for the distance or the travel time. Handwriting support is not required.

Map is a popular choice of application domain for multimodal systems, being studied in [Neal91], [Matsu'ura94], [Cheyer95], [Oviatt96], [Martin97], and possibly many other research projects. The most important motivating factor is the strong visual component inherent in map manipulation. Oviatt et al. [Oviatt97a] showed that in a map task the commands most likely to involve more than one modality are *spatial location commands* which require specifying a spatial location description. [Oviatt97b] identifies performance difficulties with speech-only interaction in a map task and reports a strong user preference to interact multimodally for this task.

The map application domain is strongly supported by MMApp and MMTk according to the criteria outlined in section 1.2 because the speech and pen modalities are strongly supported and the map state can be manipulated by parameterized actions.

The following are some examples of multimodal commands for QUICKTOUR (pen input elements are in italics surrounded by square brackets):

```
Please locate Carnegie Mellon University for me.

Can you show me the fastest route from here [point at a
place] to University of Pittsburgh?

Where are the Chinese restaurants in this area [rectangle]?

How long does it take to go from here to here [arrow from
one place to another]?

Zoom in five times [circle the center of the zoom].
```

6.2 Design

The system architecture of QUICKTOUR follows the default MMApp architecture (Figure 9 on page 54) without any significant changes.

Step-by-step application of the design process in section 5.1 is outlined below.

Selecting Action Frames and Parameter Slots

The action frames and parameter slots follow directly from the supported commands listed in the above requirement analysis. The resulting list is summarized in Table 5.

Action Frame	Parameter Slot	Description
ZoomIn	*ZoomInCenter*	Center point of the zoom
	ZoomInAmount	Whole-number zoom factor
ZoomOut	*ZoomOutCenter*	Center point of the zoom
	ZoomOutAmount	Whole-number zoom factor
ZoomBox	*ZoomBoxCenter*	Center point of the zoom
	ZoomBoxAmount	Size of square zoom area
	ZoomBoxArea	Contour of zoom area
Pan	*PanDirection*	Direction of the map shift
	PanAmount	Distance of the map shift
Find	*FindSpec*	Search criteria
FindAll	*FindAllSpec*	Search criteria
	FindAllArea	Contour of the search area
FindPath	*FindPathSrc*	Path starting point
	FindPathDst	Path end point
FindPathLen	*FindPathLenSrc*	Path starting point
	FindPathLenDst	Path end point
FindPathTime	*FindPathTimeSrc*	Path starting point
	FindPathTimeDst	Path end point

Table 5. Action Frames and Parameter Slots for QUICKTOUR

It is possible to combine *FindPath*, *FindPathLen*, and *FindPathTime* into a single action frame with an additional parameter slot specifying the required path information (distance or travel time). However, as pointed out in section 3.2.2, experience has shown

that the MS-MIN-based semantic integrator can achieve higher accuracy if separate action frames are used so that the action frames and their parameter slots are more tightly bound. This is reflected in the above table.

Designing the Input Model

The MMGL input model is constructed using the Multimodal Grammar Editor described in section 5.3.

It is straightforward to create one AFrame node per action frame listed in Table 5. For each AFrame, the procedure to design the PSlots is as follows:

- Construct speech-only commands for the given action, segment them into parameter slots, abstract the segments into context-free grammars, and create corresponding PSlots each containing one Speech UnimodalNode;
- Do the same for pen-only commands to obtain PSlots each containing one Pen UnimodalNode;
- Do the same for speech-and-pen combinations to obtain PSlots each containing Speech and/or Pen UnimodalNodes.

As an example, consider the *ZoomIn* action frame. Typical speech-only commands for this action follow a few main patterns:

(a) `Zoom in (magnify, enlarge) around (at, on) <place> <number> times (by a factor of <number>).`

(b) `Zoom in (magnify, enlarge) <number> times (by a factor of <number>) around (at, on) <place>.`

(c) `Show (display) <place> <number> times more detailed.`

(Synonyms and alternative phrasings are in parentheses.)

The above three patterns translate into three PSlot sequences:

(a) *ZoomInCenter(0) — ZoomInAmount(0)*

(b) *ZoomInAmount(1) — ZoomInCenter(1)*

(c) *ZoomInCenter(2) — ZoomInAmount(2)*

The numerical tags in parentheses distinguish PSlot nodes for the same parameter slot. It is straightforward to create context-free grammars for the contents of the PSlots.

Either the zoom center or the zoom factor may be omitted, in which case default values will be used. Thus we have two more PSlot sequences consisting of only *ZoomInCenter(0)* or *ZoomInAmount(1)*. Examples of utterances for these sequences are "Zoom in around Carnegie Mellon University" and "Magnify two times."

There are no pen-only commands for *ZoomIn*. Combined speech and pen commands use a deictic gesture instead of a verbal specification to indicate the zoom center. Thus pattern (a) above becomes

(a') `Zoom in (magnify, enlarge) around (at, on) here`
`[pen deictic]` `<number> times (by a factor of <number>).`

The phrase "around here" may be omitted. The corresponding PSlot sequence is

(a') *ZoomInCenter(3) — ZoomInAmount(0)*

ZoomInCenter(3) contains both Speech and Pen UnimodalNodes. Note that it makes sense to reuse *ZoomInAmount(0)* here because the second part of the spoken utterance remains the same. A significant part of the Speech UnimodalNode in *ZoomInCenter(0)* can also be reused in *ZoomInCenter(3)*.

Pen deictics can be a point, a circle, a rectangle, or a cross. The Pen UnimodalNode in *ZoomInCenter(3)* is filled in accordingly.

At this point is becomes important to decide on an encoding for pen input. Besides the gesture shape, the *spatial context* of a gesture plays an important role in its semantics. In this application, it is important to know whether the gesture refers to a map object (an icon for a place displayed on the map); therefore, a *point* gesture is encoded as *point* if it falls on an empty spot of the map and as *point_object* if it falls on an object. Each encoded gesture also carries an attach data packet; for *point* this packet specifies the coordinates of the point and for *point_object* the packet specifies the object ID. Gesture shapes such as *arrow* and *line* carry significant information in their start and end points, hence they are encoded as sequences of the form a*rrow_start* followed by a*rrow_end*. An *_object* suffix is appended if the start or end point falls on an object.

The design of other PSlot sequences proceeds in the same fashion.

The resulting MMGL input model contains 9 AFrames, 84 PSlots, 59 UnimodalNodes, 207 NonTerms, 435 Literals, and 451 distinct words.

Generating Unimodal Language Models

This step is straightforward once the input model has been created. Running the N-gram Language Model Generator (see section 5.5) on the input model produces a trigram language model file suitable for loading into JANUS or SPHINX. The language model contains 454 unigrams, 36,322 bigrams, and 2,118,083 trigrams.

Instantiating the Multimodal Semantic Integrator

This step requires the identification of macro concepts for input preprocessing (see section 5.6.1). The macro concepts should be significant in the extraction of action parameters.

For the speech modality, we need to extract six types of parameters:

- Zoom factor (for *ZoomIn* and *ZoomOut*)
- Zoom box size (for *ZoomBox*)
- Pan direction (for *Pan*)
- Pan amount (for *Pan*)
- Map location (for *ZoomIn/Out/BoxCenter* and *FindPath* commands)
- Search criteria (for *Find* and *FindAll*)

The zoom factor, the zoom box size, and the pan amount are whole numbers, hence we need to parse number phrases. For the zoom box size and the pan amount we also need to specify a unit of distance, either mile or kilometer. The pan direction requires specifying compass directions. Verbal specification of map locations should indicate street addresses and/or place names. Search criteria include object types and attributes in addition to addresses and names.

From the above considerations it is obvious that the macro concepts for this application should include the following:

- *Number*: This should include all number phrases that are likely to occur for this application. Zoom factors and pan amounts are small numbers, but street numbers in addresses may have up to 4 or 5 digits, hence the grammar for the *Number* macro concept should cover all phrases from "one" to "ninety nine thousand nine hundred and ninety nine." Alternatives such as "nineteen ninety eight," "nineteen hundred and ninety eight," and "one thousand nine hundred ninety eight" should be included.
- *DistanceUnit*: This should cover "mile(s)" and "kilometer(s)."
- *Direction*: This should include all eight compass directions ("north," "south," "east," "west," as well as "northeast," "southwest," etc.) plus "left," "right," "up," and "down."
- *Street*: This should include all the street names of interest. The grammar for this can take advantage of the fact that a street name may have up to three parts: a direction ("North," "South," etc.), a name, and a type ("Avenue," "Boulevard," etc.). Only the name is required.
- *PlaceName*: This should list the names of all the places of interest. Examples include "Carnegie Mellon University," "Pittsburgh Zoo," etc.
- Type concepts: The type of an object consists of a mandatory *ObjectClass* and an optional *ObjectSubtype*. For instance, the class "Restaurant" can have subtypes such "McDonald's" or "Burger King."
- Attribute concepts: These include *Nationality* such as "Chinese" or "French," *Quality* such as "good" or "excellent," and *Price* such as "inexpensive."

For the pen modality, it is important to be able to extract coordinates or object IDs from the pen tokens. The simplest solution is to represent each gesture token as a macro concept. For instance, a gesture encoded as *point* becomes the macro concept *P_Point* while *point_object* becomes the macro concept *P_PointObject*[†]. Unlike for speech, the pen macro concepts are not parsed from the input stream but must be constructed explicitly by the program. In the MMGL input model we can use placeholders for data that will be filled in by the program; for instance, *P_Point* contains the Literal "?x ?y" which indicates that x-y coordinates will be filled in at runtime.

Macro concepts must be marked in the MMGL input model as "parseable" NonTerm. The Grammar Designer includes "Parseable" in the Properties dialog for NonTerms, accessible via the context-sensitive menu that pops up when a node is clicked with the right mouse button (see section 5.3).

Once macro concept nodes are properly marked, appropriate MMTk tools generate an input preprocessor that parses the speech stream and an MS-MIN integrator (see sections 5.6.1 and 5.6.2).

Implementing the Parameter Extraction Postprocessing

Running the Postprocessor Generator (see section 5.6.3) on the MMGL input model creates a MapInterpBase Java class containing a skeleton algorithm to traverse the output of the semantic integrator and construct the output action and its parameters. Domain-specific parts of the algorithm must be implemented in a MapInterp class that extends MapInterpBase and overrides appropriate methods. The implementation of MapInterp is described in the next section.

6.3 Implementation

The user interface is a MapApplet class that extends MMTk's MultimodalApplet. MapApplet uses the same layout as MultimodalApplet, hence the only graphical element that needs to be added is a MapView object that displays the map inside the applet's PenPanel.

MapView is implemented as an Observer for a Map object (see the Observer design pattern in Appendix C). The map is represented as a graph; the vertices are the street intersections and the edges are the street segments. This graph representation facilitates the computation of shortest paths using Dijkstra's algorithm [Knuth73]. The Map object also contains a list of places with attributes such as name, address, icon image, type, and so forth. MapView can display these places as icons; in addition, MapView has methods to adjust the view by zooming and panning. The current state of MapView can be captured in a snapshot and restored later; this is the mechanism for undoing actions.

[†] Naming pen nodes with a *P_* prefix avoids potential conflicts with the names of speech nodes.

MapApplet overrides three abstract methods in MultimodalApplet as specified in section 4.7.4. The executeCommand() and undoCommand() methods work on a MapCommand object that encapsulates a command action and associated parameters. MapCommand is an abstract base class that declares execute() and undo() methods to be defined by subclasses. There is a MapCommand subclass for each action frame or group of related action frames:

- ZoomCommand implements the *ZoomIn* and *ZoomOut* actions;
- ZoomBoxCommand implements the *ZoomBox* action;
- PanCommand implements the *Pan* action;
- FindCommand implements the *Find* action;
- FindAllCommand implements the *FindAll* action;
- FindPathCommand implements *FindPath*, *FindPathLen*, and *FindPathTime*.

The remaining abstract method, getInterpretation(), receives a group of InputEvents and returns a MapCommand object. Internally, getInterpretation() transforms the InputEvents to a form suitable for semantic integration and invokes a method on a MapIntegrator object to perform the integration (the request is transparently forwarded to a remote server process via the Switchboard as described in section 4.6.3). Speech InputEvents are parsed using the generated input preprocessor to produce a sequence of tokens, each of which may be a simple word or a macro concept. Pen InputEvents are encoded and transformed into a sequence of macro concept tokens as described in section 6.2 above.

The output of the semantic integrator is postprocessed by MapInterp, which extends the MapInterpBase postprocessing skeleton generated from the input model. The generated code in MapInterpBase branches to appropriate methods based on the names of the AFrame and PSlot nodes encountered during a traversal of the integrator output. MapInterp customizes those methods to extract the action name and parameter values needed to instantiate a subclass of MapCommand. There are also helper methods to convert the text in macro concepts to values of the appropriate types; for instance, a getNumber() method receives a NonTerm representing the *Number* macro concept and computes an integer value from the words in the parse tree of the NonTerm.

Besides the MapApplet user interface which runs in a Web browser, QUICKTOUR includes a JANUS speech recognition server, a TmplGRec gesture recognition server, and an MS-MIN semantic integration server. The servers can be distributed on fast workstations and accessed via the central Switchboard running on the Web server machine. Services are accessed through the interfaces defined in MMApp, hence any compatible implementations of the servers can be transparently substituted for the default versions.

Figure 27 on page 110 shows the QUICKTOUR user interface with gesture and speech input events in progress.

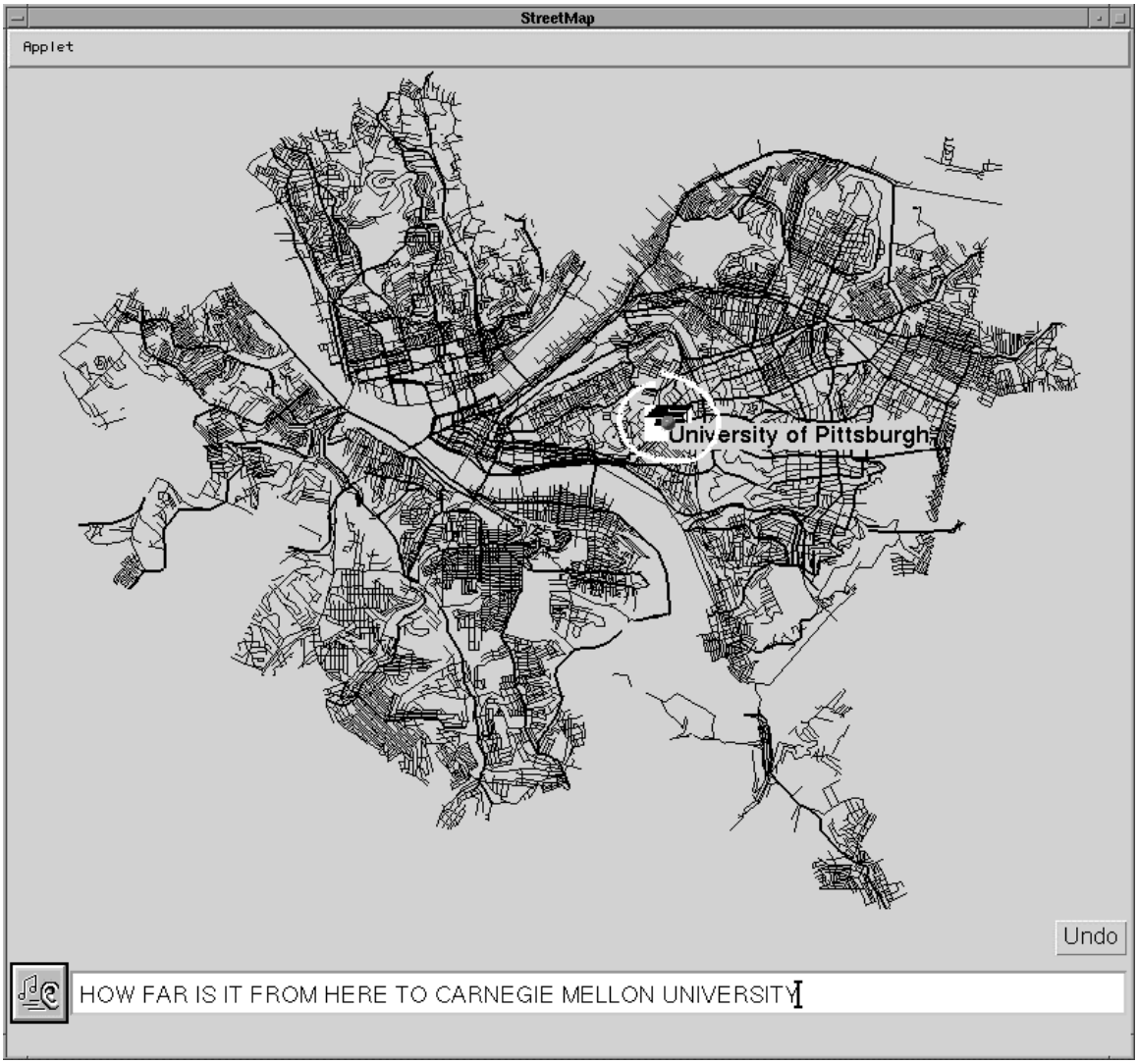


Figure 27. Multimodal Map Application

Chapter 7
EVALUATION

This chapter offers some validation of the work presented in this dissertation by examining the application of the proposed framework and design process to real systems. The first section describes a validation of the development procedure (i.e., the process of using MMApp and MMTk in application development) in terms of development effort and portability across different task domains. The second section focuses on one multimodal system constructed using the development procedure, namely the QUICKTOUR application described in the previous chapter, and demonstrates that the resulting system is a working application that enables real users to perform useful tasks in a multimodal interaction style.

7.1 Evaluation of the Development Procedure

This section describes two other applications that demonstrate the portability of the framework to different task domains and presents an assessment of the development effort for the QUICKTOUR design example in Chapter 6 to illustrate the benefits of developing the application using the MMApp framework and the MMTk workbench.

7.1.1 Portability

This dissertation presents a multimodal semantic model, an application framework, and a tool-assisted design process to facilitate the construction of a broad class of multimodal applications. Chapter 6 described how a complete application was constructed using the above development framework. This section demonstrates the applicability of the framework to more than one domain by presenting two other multimodal applications that have been developed using MMApp and MMTk.

JEANIE-II — A Multimodal Appointment Scheduler

[Vo96] describes a multimodal calendar application called JEANIE. Its successor, JEANIE-II, is a complete re-implementation using MMApp and MMTk.

Figure 28 on page 112 shows the JEANIE-II user interface. The layout inherited from MultimodalApplet is essentially the same as that of QUICKTOUR (see Figure 27 on page 110), except that a calendar display replaces the map display inside the PenPanel.

A user can schedule new meetings, cancel existing meetings, or changing them in various ways by speaking, drawing gestures, writing words, or using any combination thereof. In Figure 28, an arrow accompanied by spoken words moves the indicated meeting to the location at the tip of the arrow. This is a case of redundant information

from two modalities because the start of the arrow indicates the same meeting specified by speech. Other ways to specify the same operation include

- Using the arrow gesture alone without speaking;
- Circling or pointing to the board meeting and saying "Move this to Wednesday at 1 PM;"
- Drawing a rectangle covering the 1PM-2PM time slot on Wednesday and saying "Move the board meeting here."

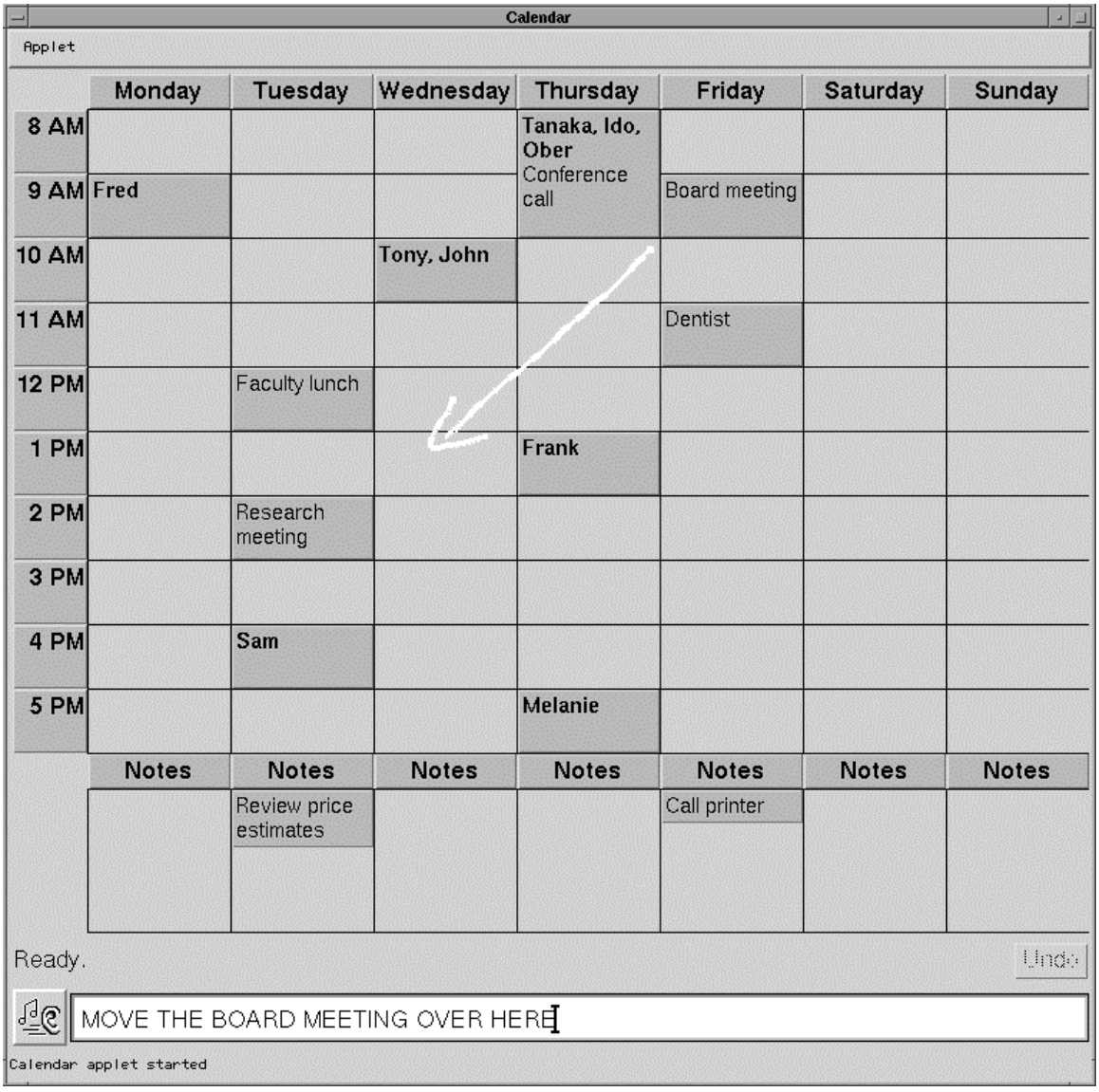


Figure 28. Multimodal Appointment Scheduler Application

JEANIE-II also supports handwriting in addition to speech and gestures. Attendee names and meeting topics can be spoken or written. Writing is useful if the speech recognizer has difficulty with certain words, especially proper names.

QUARTERBACK — A Multimodal Football Simulation

QUARTERBACK is an application written by Gregory Cortis Clark using MMApp and MMTk. The user interface depicted in Figure 29 shows a representation of a football field populated by football players in formation.

A user can draw gestures on the rectangle on the right hand side to describe a play. Circles indicate player positions and arrows specify their movements. Saying “Ball carrier” while drawing a play assigns the ball to the indicated player. Crossing out previously assigned players removes them from the play description. Stored plays can also be recalled verbally by speaking the play number and name, e.g. “620 Belly Counter.” Once a play description has been completed, saying “Execute play” animates the player icons on the field to simulate the play.

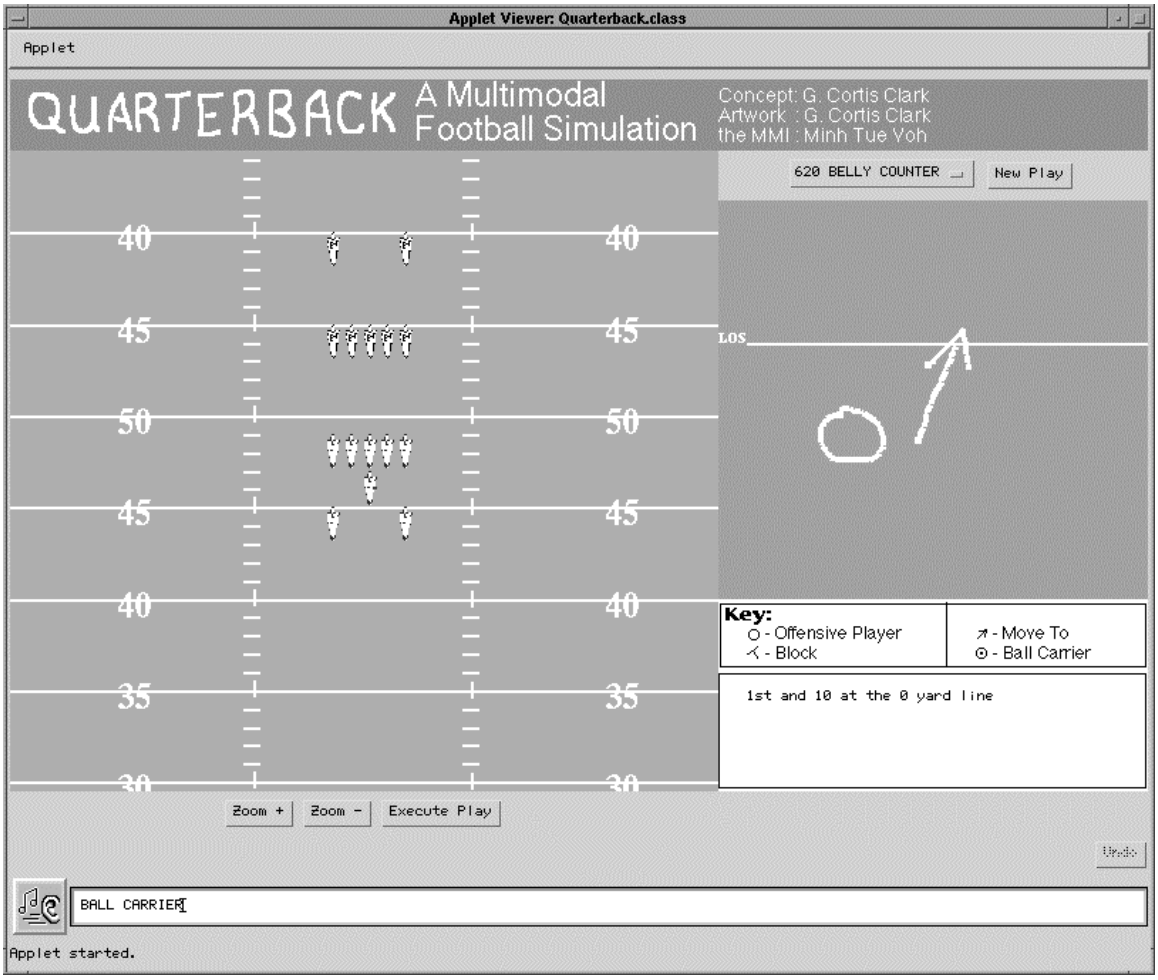


Figure 29. Multimodal Football Application

The implementations of JEANIE-II and QUARTERBACK both accept the default options in MMApp to minimize the amount of required customizations.

7.1.2 Ease of Development

It is quite difficult to quantify the benefits of using MMApp and MMTk for application development without any point of reference for comparison. A thorough quantitative analysis would require something equivalent to implementing the same application with and without MMApp/MMTk support and comparing the development efforts in the two implementations.

This section approaches the matter from a more subjective point of view with the intention of showing that an application implementation reused a large amount of code compared to the amount of application-specific customization, thereby benefiting from the framework by implication. Another consideration that demonstrates quantifiable benefits is the performance improvements measured against the incremental efforts to achieve those improvements after testing the application on real users.

Code Reuse in Application Development

	Subsystem	Lines of code	Public classes	Public methods
MMApp	Networking	1834	23	121
	Speech	2190	12	114
	Pen	1163	8	88
	Multimodal coordination	297	5	31
	User interface	770	1	20
	Miscellaneous	891	8	86
	Subtotal	7145	57	460
MMTk	Grammar	2534	30	208
	Designer	5356	41	388
	Generator tools	7669	28	294
	Miscellaneous	2353	17	104
	Subtotal	17912	116	994
Servers	SRecServer	8277	N/A	N/A
	TmplGRec	2214	N/A	N/A
	MS-MIN	4219	N/A	N/A
	Subtotal	176520	N/A	N/A
	Total	315394	173	1454

Table 6. Size of Software Component Libraries

Table 6 on page 114 lists the software subsystems used for multimodal application development. The size of each subsystem is measured by three variables: the total number of lines of code (excluding comment lines), the number of publicly accessible classes (i.e., classes that application developers can use), and the number of publicly accessible methods (i.e., methods that application code can call) in those classes[†]. The number of code lines (the shaded column in Table 6) is the main measurement for the size of the system, while the number of public classes and methods represents the size of the API that application developers must learn.

Table 7 quantifies the development effort for the QUICKTOUR application described in Chapter 6. The application program code is divided into three parts:

- *Map display*. This consists of classes that encapsulate the map data, display a map view on the screen, and implement map operations such as zooming, panning, searching for and displaying places on the map, etc. The size of this part of the program represents the application-specific development effort that is independent of the multimodal infrastructure.
- *Interpretation*. This consists of classes that employ MMApp facilities to capture, coordinate, and interpret multimodal inputs for QUICKTOUR. This part of the program represents application-specific customizations of MMApp facilities.
- *Generated*. This is the postprocessor skeleton generated by MMTk.

In addition to the number of code lines, classes, and methods, the table lists the number of man-hours spent on the development of each part of the program and on the MMGL input model for the application.

Subsystem	Lines of code	Public classes	Public methods	Man-hrs
Map display	2958	13	132	250
Interpretation	4193	43	258	360
Generated	2738	1	1	N/A
Input model	N/A	N/A	N/A	160
Total	9889	57	391	770

Table 7. QUICKTOUR Application Development Effort

The figures in Table 7 show that 30% of the program code and 32% of the time spent belong to the implementation of the map functions, which have nothing to do with handling multimodal inputs. About 42% of the code and 47% of the time can be attributed to hand-coded customizations of the MMApp components used in input interpretation. Automatically generated code accounts for 28% of the program. 21% of

[†] The class and method information is only available for Java-based modules.

the time went into the creation of the MMGL input model. The whole project took about 3 man-month (assuming 8 hours of work per day, or 1 man-month of continuous 24-hour work days) to complete.

The 4,193 lines of code[†] for handling multimodal interpretation represent 19% of the total size of MMApp and the associated servers (MMTk components are intended for input model work and thus are not directly incorporated into the application program). Thus we can infer that reusable components from the framework account for more than 80% of the code for capturing and interpreting multimodal inputs.

The MMGL input model was partly based on a (speech-only) context-free grammar written for another map project. Without this head-start, it would have taken a larger fraction of the development time to create the input model. The 21% of development time spent on the input model paid off handsomely in the form of the automatic generation of 28% of the program code, the language model for speech recognition, and the MS-MIN for semantic integration.

Development Effort Coupled with Learning the Framework

Table 8 summarizes the development effort that Gregory Cortis Clark spent on the QUARTERBACK application. This data is interesting because in this case the application developer did not have any prior experience with MMApp and MMTk.

Subsystem	Lines of code	Public classes	Public methods	Man-hrs
Football	852	13	134	121
Interpretation	333	4	27	100
Input model	N/A	N/A	N/A	8
Total	1185	17	161	229

Table 8. QUARTERBACK Application Development Effort

72% of the program code and 53% of the development time went into the implementation of the football simulation. Multimodal interpretation accounts for only 28% of the code, but nearly 44% of the time. The extra time was needed to learn how to use the framework and customize the components.

Little time was spent on input modeling because the input model for this application is very simple (compared to the QUICKTOUR model) and handles only speech. Pen gestures in QUARTERBACK are specialized commands for play description, hence it

[†] Most of this code (3347 lines, or 80%) is for extracting parameter values and implementing the MapCommands for the actions. The complexity comes from handling a large number of parameter-carrying token types and decoding them from textual form to the appropriate data types.

was easier to handle them separately in custom code rather than using MS-MIN semantic integration.

The application developer, G. Cortis Clark, had the following comments about his experience in using MMApp and MMTk for multimodal application development:

- The largest obstacle to learning the framework is the lack of documentation. This dissertation would have significantly remedied this deficiency; unfortunately it was still a work in progress at the time the QUARTERBACK application was written. Cortis had to spend a significant amount of time reading part of the MMApp source code and consulting me. Work on a complete set of documentation for the APIs and the design process is currently in progress with input from Cortis.
- Some aspects of MMApp, especially the user interface, are still not very flexible. It is harder than necessary (perhaps partly due to the lack of documentation) to make small customizations if the default options are slightly inappropriate. A visual interface builder would be a welcome improvement.
- Knowledge of the multimodal semantic model and MMGL formulation is necessary to the effective use of MMTk tools, including the Grammar Designer. Cortis had to spend 4 hours in consultation with me for a tutorial on input modeling and parsing, as a consequence of the lack of documentation.
- Once enough experience is gained during the course of development, it is rather easy to apply the framework and design process to other applications. Cortis is currently offering his newfound expertise for consultation in a project involving fitting an existing application into the MMApp framework.

7.1.3 Incremental Improvement

The design process outlined in section 5.1 enables the rapid prototyping of an application even if little or no data on user input patterns for the application domain is available. However, once a working prototype of the application is deployed, it becomes feasible to collect user data by letting real users interact with the system. The collected data could serve as a basis for improving the performance of the application.

A set of user data was collected for the QUICKTOUR application during user observation sessions, as described in section 7.2.2 below. The data was manually labeled and used to test the interpretation accuracy of the MS-MIN semantic integrator. (The tests in this section used transcribed speech and gestures to eliminate the effects of recognition errors on the interpretation accuracy.) The question is, given this data, how much effort would be needed to discover and implement modifications that would result in performance improvement. Two modifications documented in Table 9 illustrate an answer to this question.

It should be noted that the results reported in Table 9 only serve to illustrate the ease with which certain flaws in the application can be discovered and corrected. The improved performance figures were measured on the same data set used to achieve the improvements, hence it is very likely that the figures include some optimistic estimation bias, similar to the way performance measured on the training set is almost always higher than that measured on an independent test set. Even so, the flaw-correction process described below illustrates how easy it is to adapt an application prototype to fit real user data and quickly obtain better interpretation accuracy.

The MS-MIN can produce several hypotheses ranked by score. In Table 9 the top three hypotheses are considered for each input sample.

Modification	**Man-hrs**	**Interpretation accuracy (%)**		
		Top 1	**Top 2**	**Top 3**
(Initial system)	N/A	80.2	89.5	90.1
Adjust sequence weights	0.25	81.6	89.8	90.9
Add synonyms & alternatives	0.75	83.0	92.8	93.4

Table 9. Incremental Improvement in QUICKTOUR

The MMTk workbench includes a program that tests an MS-MIN on a set of labeled data. The output of the tester program contains a confusion matrix which tabulates the number of times an action A is incorrectly classified as another action B. Upon examination of this confusion matrix, it was apparent that many *Find* actions were classified as *Zoom* actions.

Further analysis of the hypotheses generated by the MS-MIN revealed that the errors were mostly caused by spoken utterances of the form "Show/display <something>." One look at the MMGL input model for QUICKTOUR showed that the *Zoom* commands also included "show/display" phrases, as in "Show the area around CMU with more detail." This resulted in strong associations that favored the *Zoom* commands in the output.

The above discovery revealed a flaw in the QUICKTOUR input model. The "show with more detail" phrases were intended as a supplement for the "main" phrases such as "zoom in," but the two alternatives were given equal weights. One obvious remedy is to reduce the weight of the "show" phrases relative to the "zoom" phrases to give less emphasis on the "show" phrases in the context of the *Zoom* commands. The "show" phrases will then favor the *Find* commands instead. This modification should not adversely affect performance on the *Zoom* commands because the "more detail" part of the phrases is still strongly associated with *Zoom*.

Minor weight adjustments similar to the above were necessary in a few other parts of the input model. After the weight adjustments, the interpretation accuracy increased from 80.2% to 81.6% (the second data row of Table 9). It is interesting to note that the accu-

racy based on the top 2 or 3 hypotheses did not increase as much, indicating that the correct *Find* commands had been in the top 2 or 3 choices even though the *Zoom* commands took the top slots. The effect of the weight adjustments is to change the scores enough so that the correct output rose 1 or 2 notches in the ranking.

It took only 15 minutes to discover and implement the weight adjustments.

Running the tester program again after adjusting the weights produced another confusion matrix. This time a number of misclassified *ZoomBox* commands yielded two discoveries:

- The input model included many synonyms referring to a square area on the map ("square," "rectangle," "box," "window," etc.) but it turned out that the participants also used other terms such as "radius," "perimeter," "region," etc.;
- The input model covered constructs of the form "a two-mile area" but quite a few participants used variants of "an area two miles on each side" instead.

A few other synonyms or equivalent terms were missing from other parts of the input model (e.g., "expand" was not on the list of "zoom in" verbs).

It took 45 minutes to go through the erroneous hypotheses, extract the missing synonyms and alternative phrasings, and add them to the input model (after generalizing from them in the context-free grammar formulation of MMGL). The result was an increase in interpretation accuracy from 81.6% to 83% (the third data row of Table 9). This time there was a corresponding improvement in the accuracy based on the top 2 or 3 hypotheses because the missing terms that were added had been very important to the classification of the affected commands. Without those terms the correct commands received scores too low to put them in the top 3 choices.

The data in Table 9 shows that one additional hour of development time was enough to improve the interpretation performance by almost 3%. As noted above, this performance increase is probably optimistically biased because it was measured on the "training set" for the improvements.

It should be possible to increase the interpretation accuracy on the collected data set by incrementally training the MS-MIN on the data. However, there is no guarantee that such performance improvements would generalize well to unseen data. The modifications described above are more general because they cover *patterns* extracted from the collected data rather than just the samples in the data set. The only way to confirm the above conjecture is to measure the performance on an additional independent test set; however, no experiment along that line has been performed because of a lack of data.

7.2 Evaluation of the Product

This section describes experimental results that provide an assessment of how well the QUICKTOUR application works in practice. The first subsection describes some tests

that demonstrate the feasibility of using the MS-MIN for multimodal interpretation. The next subsection presents results obtained from real data collected during user observation sessions.

7.2.1 Performance of the MS-MIN

Ideally, we should collect a large amount of user data using a Wizard-of-Oz study [Salber93], manually label each multimodal input event in the data set with the correct action and parameters, use a portion of the labeled data to train an MS-MIN, then test the network on the rest of the labeled data. Doing this properly requires a large scale, very time-consuming effort. However, it is possible to use an MMGL input model instead of real data in the tests. This is what application developers should do in the application prototyping stage when real data is scarce.

The input model constructed for QUICKTOUR is a non-trivial model that covers a wide range of input variations; therefore, using data generated from the model to measure the interpretation accuracy of the MS-MIN will at least provide a useful indication of whether the semantic integration algorithm works at all.

Interpretation Accuracy

In the first test, 11,000 labeled multimodal input events were generated from the MMGL input model for the QUICKTOUR application, using the Random Sample Generator described in section 5.4. An MS-MIN was trained on 10,000 of the samples by presenting each sample in turn together with the correct segmentation into parameter slots as specified by the MMGL model (the Sample Generator retains this information as it constructs the samples).

The trained network was tested on both the training set and a test set composed of the remaining 1,000 samples. For each sample to be tested, the input data was segmented into parameter slots by the MS-MIN, then an action and its parameters are extracted from the network output. These are compared to the action and parameters specified by the correct semantic labeling from the MMGL model, and the interpretation is deemed correct if the action and all the parameters match perfectly. The network can produce several hypotheses ranked by score, and the top three hypotheses are considered for each input sample.

To demonstrate the usefulness of the Integration Network Generator described in section 5.6.2, another MS-MIN was generated directly from the MMGL model and tested using the same 11,000 samples.

Table 10 on page 121 summarizes the test results. The MS-MIN trained on 10,000 samples achieves an interpretation accuracy of over 90% on the training set. As expected, the performance on the test set is slightly lower, but still not far behind at almost 88%. About 95-96% of the time the correct interpretation is within the top two or three hypotheses.

	Trained network			Generated network		
	Top 1	**Top 2**	**Top 3**	**Top 1**	**Top 2**	**Top 3**
Training set (%)	90.4	95.8	96.6	92.0	96.2	96.8
Test set (%)	87.9	95.0	96.1	91.3	96.1	96.3

Table 10. MS-MIN Interpretation Accuracy on Artificial Data

As the test input samples were drawn randomly and independently according to the implicit probability distribution in the MMGL input model, the error rate measured on the test set is a random variable that follows a binomial distribution. We can approximate this distribution with a normal distribution and derive a confidence interval for the interpretation accuracy figures reported in Table 10 [Mitchell97]. For an observed error rate *e* on a test set of size *n* (i.e., the interpretation algorithm makes *en* mistakes), the *N*% confidence interval for the true error rate is given in [Mitchell97] as

$$e \pm z_N \sqrt{\frac{e(1-e)}{n}} \tag{16}$$

where z_N is given in Table 11 and $n = 1000$ for this experiment.

Confidence interval *N*%	50%	68%	80%	90%	95%	98%	99%
Constant z_N	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Table 11. Values of z_N for Two-Sided $N\%$ Confidence Intervals

Thus, the 95% confidence intervals for the interpretation accuracy of the trained and generated networks are $87.9 \pm 2.0\%$ and $91.3 \pm 1.7\%$, respectively.

Because the connection weights in the generated network were computed from the true probability distribution built into the MMGL input model, it is as if the network were trained from an infinitely large number of samples generated from the input model. As such, it is reasonable to expect that the generated network would be able to generalize much better than the network trained on just 10,000 samples. Indeed, Table 10 shows that the generated network achieves higher interpretation accuracy on both data sets.

The above test results, although obtained on artificially generated data, clearly indicate that the MS-MIN can be trained to perform semantic integration of multimodal inputs according to the action frame/parameter slot semantic model described in Chapter 3. The network exhibits strong generalization power as attested by the performance on an independent test set. The network generated directly from the input model outperforms the network trained on 10,000 input samples, demonstrating the advantage of network generation over time-consuming training.

Incremental Learning

As explained in section 3.4.3, the MS-MIN is capable of learning incrementally; i.e., it can improve its performance gradually as it is presented with more and more examples. To demonstrate this fact, an MS-MIN was incrementally trained on data generated from the MMGL input model for QUICKTOUR (see above) using the following procedure:

The network initially contains no input units and no connections. For each input sample, the new sample is first interpreted by the network, then the correct parameter slot segmentation is presented to train the network, and finally the sample is interpreted again after the network connections have been updated by the single training step. The cumulative number of correct interpretations divided by the number of samples presented up to that point is the interpretation accuracy. The before-training evaluations approximate the performance measured on an independent test set because at evaluation time the samples had never been seen before. The after-training evaluations correspond to the "training set" performance.

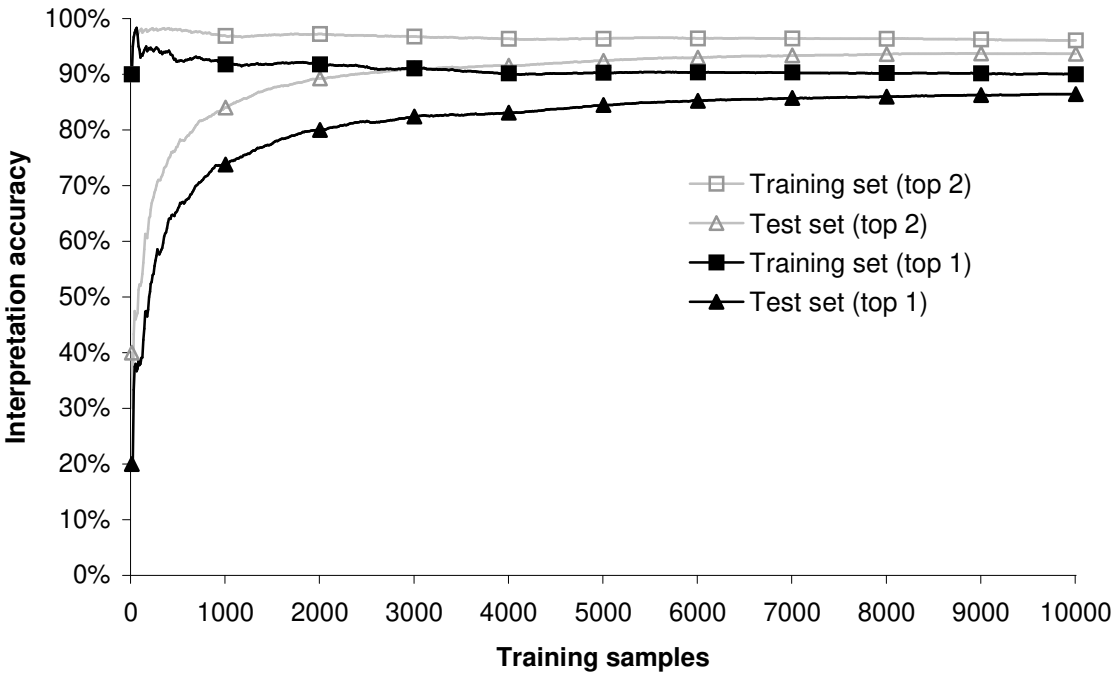


Figure 30. MS-MIN Incremental Learning

Figure 30 summarizes the test results. The "training set" curve indicates that about 90% of the time the MS-MIN was able to learn the interpretation of an input sample after a single presentation of the example. This ability was sustained at more or less the same level even when a large number of examples had been presented and the network had grown considerably. Furthermore, the network was also able to generalize from the examples and steadily improved its performance on new input samples, as attested by the "test set" curve. After 10,000 examples, the interpretation accuracy on new inputs has

risen above 86%. If the top two hypotheses are considered, the rate of correct interpretation increases to 96% on the training set and almost 94% on the test set. Note that these results closely parallel the batch-training results reported in Table 10.

As examples are presented, the network incorporates the input tokens in the training samples into its vocabulary. The MMGL input model for QUICKTOUR contains 451 unique words; however, only 161 unique tokens are visible in the input streams the network receives, because certain groups of words (e.g., street names) are folded into macro concept nodes by the preprocessor (see section 5.1.4). All these 161 tokens are present in the first 2,000 samples. Figure 31 shows the rate of vocabulary acquisition.

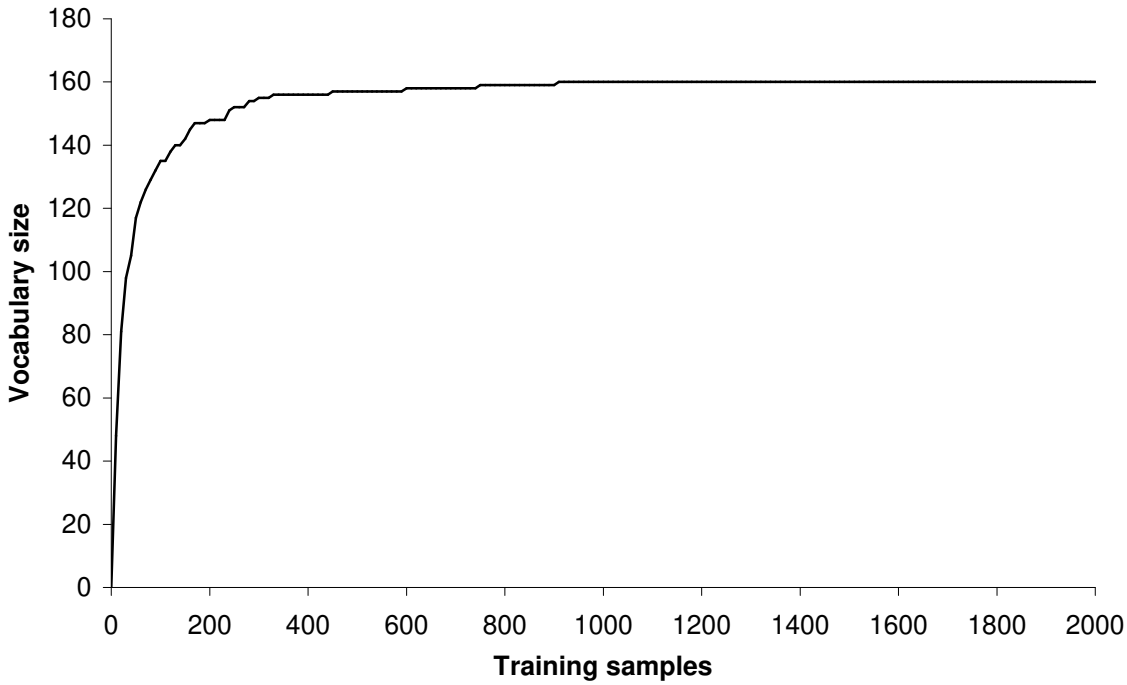


Figure 31. Vocabulary Acquisition During MS-MIN Training

After 10,000 samples, the network contains 8,215 input units and 139,655 connections. Among the input units, 161 are single-token units that make up the acquired vocabulary, and the remaining 8,054 are fragment units corresponding to combinations or two or more input tokens. The gradual growth of the network as training samples are presented is documented in Figure 32 on page 124.

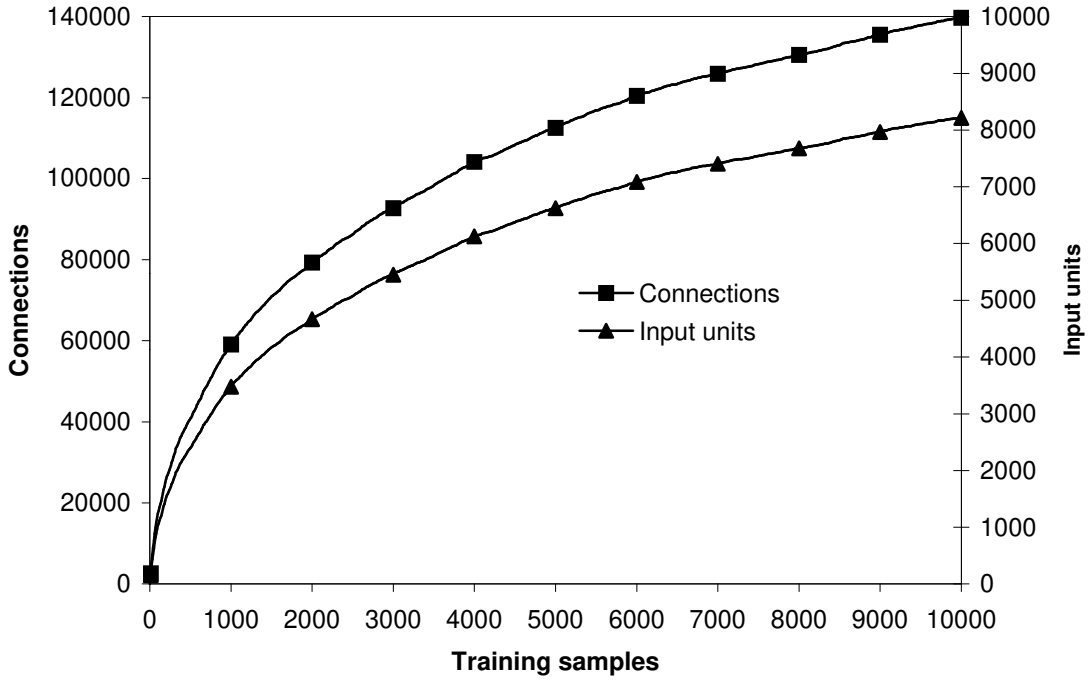


Figure 32. Network Growth During MS-MIN Training

7.2.2 User Observation Results

A number of user observation sessions were conducted to provide some real data for performance assessment of the QUICKTOUR application. The test results described below demonstrate that QUICKTOUR, a multimodal application constructed using the framework, toolkit, and design process described in this dissertation, does work in practice and enables real users to accomplish useful tasks using multimodal commands.

Protocol

Ten paid participants (4 males and 6 females, comprising mostly students and office workers at a pharmacy school) were asked to use QUICKTOUR to perform a series of 12 tasks involving most of the operations supported by the map (see Chapter 6). The participants were encouraged to speak naturally as they would to an intelligent assistant who would manipulate the map display according to their wishes, and to draw on the screen as well if they thought it would be easier to express certain ideas that way. A participant could go to the next task only after successfully completing the current task or giving up because the system made too many mistakes. The speech and pen data were recorded together with the responses from the application. At the end of each session, the participants filled out a simple questionnaire aimed to find out whether they liked the system or not. Each participant was also asked to read a set of 100 sentences (generated from the MMGL input model) to provide data for speech recognition improvement later.

A detailed description of the user observation protocol, including the list of tasks and the questionnaire, can be found in Appendix D.

User Preference

The first three questions on the questionnaire (page 171) aim to find out whether the participants liked the multimodal interaction style in QUICKTOUR. 7 out of 10 participants indicated that they liked being able to speak and draw to communicate with the computer very much; 2 people said they liked it quite well; the remaining 1 person said she liked it somewhat. None of the participants indicated that they did not like it at all. Unsurprisingly, in all 10 cases the ratings for the combination of speaking and drawing equaled or exceeded the corresponding individual ratings for speaking and for drawing.

When asked what they found the most annoying about the program, whether it was too slow or did not understand them well enough, or because of some other deficiencies, 7 out of 10 participants agreed that the program needed improvements to make it understand them better. However, perhaps surprisingly, the other 3 people stated that they did not find the program's failures annoying at all because they understood it was still experimental and they really enjoyed interacting with the program. Still, these results suggest that the first order of priority in a schedule of improvements for the application should be to increase the speech and gesture recognition performance as well as the interpretation accuracy.

Data Collection

A total of 430 multimodal input events were collected. Among these, 65 were not interpretable even for a human because of various reasons (see Table 12 on page 126). These reasons include:

- *All-noise speech*. The speech/silence detection algorithm was fooled by a noise (e.g., coughing, clearing the throat, or adjusting the headset, etc.) and stopped recording without getting any real speech.
- *Cut-off speech*. The participant spoke too softly or hesitated too long in the middle of a sentence, causing the speech recorder to stop recording prematurely, thereby cutting off the last few words.
- *Noisy speech*. This was caused by a sound driver bug that crashed the system a few times when it unaccountably ran out of buffer memory. Some recordings made just before the crashes turned out to be fragmented and noisy.
- *Accidental pen touch*. The participant inadvertently touched the screen, causing the system to process a false gesture.
- *Ambiguous or meaningless gesture*. A rectangle or a deictic gesture without disambiguating speech may have no discernable meaning or more than one possible meaning. This usually happened when a participant drew something and waited too long before speaking, causing the system to start processing the gesture prematurely.

- *Aborted input*. In the middle of a sentence, the participant became aware of a mistake and floundered or started to laugh.
- *Incomplete or incorrect input*. Either the input did not contain enough information for even a human to decide what it meant, or certain information was erroneous. For example, one participant said "University of Pennsylvania" instead of "University of Pittsburgh."

Reason	Number discarded
All-noise speech	10
Cut-off speech	13
Noisy speech (driver bug)	12
Accidental pen touch	5
Ambiguous or meaningless gesture	11
Aborted input	8
Incomplete or incorrect input	6
Total	65

Table 12. Discarded Input Events in User Data

The 65 non-interpretable inputs events were discarded, leaving a total of 365 multimodal input events. This is still 3 times the minimum number of input events required by the tasks (10 users × 12 tasks = 120 needed input events) because of several factors. Sometimes a participant had to redo a task several times before the application succeeded in producing the correct interpretation. In a number of cases, the application correctly interpreted the input; however, the result was still incorrect because the participant asked the wrong question or left out certain required information specified by the task[†]. Three participants had to start over from the beginning because of system crashes involving an obscure sound driver bug in the operating system; the data collected during their first (partial) sessions remained valid.

The recorded utterances and gestures were manually transcribed. Based on the transcriptions, the combined multimodal input events were manually labeled with semantic information, enough to produce the desired action and parameters. This semantic labeling serves as a reference against which to judge the correctness of the interpretations generated by the network.

[†] The most frequently occurring example was the failure to ask for the distance or travel time when this was required in conjunction with the shortest path between two places.

Among the 365 collected input events, 280 involve only speech, 84 are composed of both speech and pen inputs, and 1 consists of pen input only. Figure 33 depicts this modality distribution.

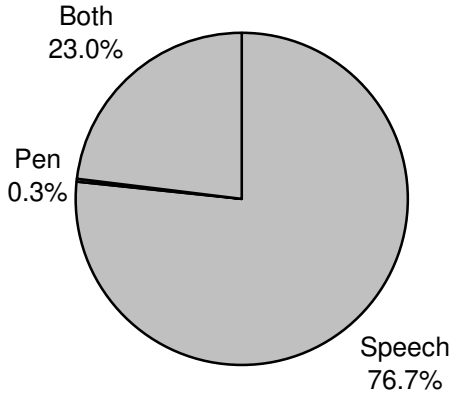


Figure 33. Modality Distribution in User Data

The modality distribution is not uniform across the 12 tasks. According to Figure 34, some tasks were exclusively performed verbally by all participants (tasks 1, 3, 4, 6, 10, and 12); others were carried out multimodally in all cases (tasks 5 and 9); the remaining tasks involved both unimodal and multimodal commands in different proportions. This uneven distribution stems in part from certain biases in the input space of the application domain: some pieces of information (e.g., addresses, place names, zoom/pan amounts, etc.) are most easily expressed verbally, while others (e.g., a rectangular area on the map) can only be specified via gestures.

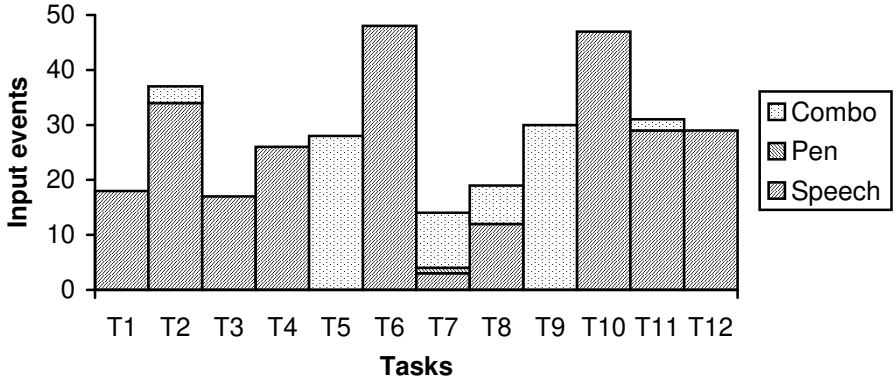


Figure 34. Modality Distribution for Each Task

Speech Recognition Accuracy

The collected utterances exhibit many characteristics of spontaneous speech: false starts, hesitations, variable speaking rates (especially when the speaker is also drawing on the screen, as people have a tendency to stretch their sentences to match the drawing

movements), sloppy articulations, etc. In addition, the utterances do not perfectly match the language model generated from the MMGL grammar. As a result, the speech recognition accuracy is rather low, as shown in Table 13. There exists many known techniques to improve speech recognition accuracy in the presence of spontaneous speech phenomena, but no attempt was made to incorporate these techniques in this experiment.

	Spontaneous speech	Read speech
Sentence accuracy (%)	21.2	39.4
Word correct rate (%)	66.6	85.3
Word accuracy (%)	61.4	83.6

Table 13. Speech Recognition Accuracy

As noted above, each participants also recorded 100 read sentences. For reference, the recognition performance on this data set is reported in the "Read speech" column above. The table reports 3 performance measures: sentence accuracy, which is the fraction of correct sentences; word correct rate, which is the percentage of correctly recognized words; and word accuracy, which takes into account the number of substitution, insertion, and deletion errors (see section 2.1.4 for detail). Word accuracy is typically the best measure of speech recognition performance.

The speech recognizer was originally based on a JANUS recognizer for the Wall Street Journal task. Using the recorded read sentences, the recognizer was adapted to the map task by running one pass of acoustic adaptation which adjusted the acoustic models to match the acoustics of the experimental settings better. In addition, certain recognition parameters (language model weight and search beam width among others) were adjusted to maximize the recognition accuracy on the set of read sentences. Table 14 shows the improved accuracy after these adaptation procedures.

	Spontaneous speech	Read speech
Sentence accuracy (%)	20.9	47.6
Word correct rate (%)	76.5	90.5
Word accuracy (%)	68.0	87.9

Table 14. Speech Recognition Accuracy After Adaptation

The speech recognition accuracy is actually much lower than what could be expected for this task. Given a vocabulary in the order of 500 words and a read-speech corpus that conforms perfectly to the language model, the speech recognizer should be able to achieve a word accuracy in the high 90% for read speech and much higher than 68% for the kind of spontaneous speech encountered in the collected data. There are several possible reasons for the mediocre performance:

- The acoustic models come from a rather old recognizer for the Wall Street Journal task and do not incorporate recent advances which have dramatically improved the performance of JANUS on spontaneous speech;
- The audio input system (a sound card on a PC) produces speech of lower quality than that normally used in speech recognizer evaluation[†];
- The language model generated by MMTk does not incorporate noise models that improve performance on spontaneous speech.

Gesture Recognition Accuracy

Because the gesture recognition model in MMApp employs sequences of gesture shapes rather than single shapes, in the same way spoken utterances are considered sequences of words, the same transcription/hypothesis alignment procedure used for speech was applied to the assessment of gesture recognition performance. In Table 15, "sentence" refers to "sequence of gesture shapes" and "word" refers to a single shape within the sequence.

	Before adjustment	**After adjustment**
Sentence accuracy (%)	49.0	60.2
Word correct rate (%)	52.4	63.1
Word accuracy (%)	35.0	48.5

Table 15. Gesture Recognition Accuracy

The gesture recognizer used in the user observation sessions was configured with a basic set of templates that cover, among other gestures, various ways of drawing a rectangle. Upon examining the collected data, it became apparent that quite a few recognition errors were due to other rectangle configurations that had been overlooked (in the mistaken belief that nobody would use these "strange" ways of drawing a rectangle). Gesture recognition was attempted again after the addition of these templates, and the improved recognition performance is reported in the "After adjustment" column above.

The majority of the collected gestures are single-shape gestures. All the exceptions are instances of pointing at or circling two separate objects in the same multimodal input event. The low word accuracy is mostly due to misrecognition of a single shape as two or more separate shapes (e.g., separate lines and arcs that were supposed to form a rectangle), resulting in insertion error rates in the order of 15%.

Because single-shape gestures seem to be predominant in this application, sentence accuracy is a better performance measure than word accuracy.

[†] In fact, a low background hum is discernable in most of the recordings.

It appears that gesture recognition accuracy is not very high because most of the gestures in the test data are rectangles, which the TmplGRec gesture recognizer tends to label as circles. The test data is thus precisely biased towards one of the weakest areas of the recognizer (see Table 3 on page 68).

However, in practice the participants did not notice the low gesture recognition accuracy because most of the rectangles served to delimit an area on the map (e.g., to accompany utterances such as "Show me the Chinese restaurants in this area"). Even when the rectangles were misrecognized as circles, the program seemed to respond correctly because circles also serve to delimit map areas, and it was not possible for the participants to detect the slight discrepancy in the exact area selected. In other words, although technically the program did not derive the exact action parameters, pragmatically speaking it correctly deduced the user's intention.

Interpretation Accuracy

An MS-MIN generated from the map MMGL input model (the same network that was used for multimodal interpretation during the user observation sessions) was tested on the collected data in three stages. In the first stage, the actual recognition hypotheses generated during the user observation sessions were used in the input events. The speech and gesture recognizers were improved using the methods described above, and newly generated hypotheses were then substituted into the input events. Next, the hypotheses were replaced by the manual transcriptions to simulate perfect recognition.

Table 16 summarizes the recognition accuracy for each modality and the overall interpretation accuracy. The performance measure is word accuracy (WA) for speech and "sentence" (or sequence) accuracy (SA) for gesture.

	Speech (% WA)	**Gesture (% SA)**	**Interpretation (%)**		
			Top 1	**Top 2**	**Top 3**
Initial recognition	61.4	49.0	35.8	41.3	41.9
Improved recognition	68.0	60.2	46.8	53.1	54.2
Transcription	N/A	N/A	80.2	89.5	90.1

Table 16. MS-MIN Interpretation Accuracy on Real Data

The last row indicates that the MS-MIN-based semantic integration algorithm performs respectably even on real data, provided all the words and gestures are recognized correctly. This is also an indication that the MMGL input model (from which the network was generated) achieves a pretty good coverage of real user inputs. Unfortunately, mediocre speech and gesture recognition performance significantly degrades interpretation accuracy.

To put the performance figures in Table 16 in perspective, recall from Table 13 and Table 14 that the sentence accuracy for speech is only in the order of 20%, yet the

interpretation accuracy is 35% to 47% and seems to increase rapidly with improved word accuracy. This means that in many cases, the MS-MIN is capable of deriving the correct interpretation even if some of the less important words (not keywords like street names etc.) are misrecognized. The recognized sentences do not even have to be grammatical, as long as it contains enough words and fragments that, as the network has learned, are strongly associated with certain output classes.

As shown in Figure 33 on page 127, more than three quarters of the input events involve only speech, and most of the rest contain speech/pen combinations. Table 17 shows how the interpretation accuracy differs across these two groups of input events. Pen-only results are not shown because a single pen-only input event was available.

		Interpretation accuracy (%)		
		Top 1	**Top 2**	**Top 3**
Speech only	Initial recognition	34.5	41.6	42.8
	Improved recognition	32.1	40.4	41.6
	Transcription	72.6	86.9	88.0
Bimodal combo	Initial recognition	36.4	41.4	41.7
	Improved recognition	51.0	56.7	57.8
	Transcription	82.5	90.3	90.7

Table 17. Interpretation Accuracy for Each Input Type

It is important to note that the comparison between speech-only and speech-pen bimodal input events may be misleading because the two input groups cover different tasks (see Figure 34 on page 127).

To assess the impact of recognition errors in a single modality, the network was tested again with transcribed speech and recognized gestures, and vice versa. The new tests use the improved recognizers. Table 18 summarizes the results.

Source of recognition errors	Interpretation accuracy (%)		
	Top 1	**Top 2**	**Top 3**
None (transcriptions)	80.2	89.5	90.1
Gesture errors only	75.0	84.1	84.3
Speech errors only	48.7	56.4	57.5
Speech & gesture errors	46.8	53.1	54.2

Table 18. Effect of Recognition Errors on Interpretation Accuracy

As indicated in the table, gesture recognition errors (100–60.2=39.8%) cause only a 5% decrease in interpretation accuracy, whereas speech recognition errors (100–68=32%) are responsible for a performance degradation of over 30%. The impact of gesture recognition errors is much less than that of speech recognition errors because only 23% of the input events include gestures, whereas 99.9% involve speech (see Figure 33 on page 127).

If a multimodal input event contains some information redundantly specified in both modalities, it is possible that the multimodal interpreter will produce the correct output despite recognition errors in either or both modalities. However, it turns out that the incidence of redundancy is rather low in the collected data. In contrast, 77 out of 84 input events that involved both speech and gestures exhibited *contrastive functionality*; i.e., the two modalities provided complementary rather than redundant information, such that it became impossible for the multimodal interpreter to overcome recognition errors by exploiting cross-modal synergy. Although the collected data represents too small a sample for statistically significant results, the data appears to confirm the observation made by Oviatt et al. [Oviatt97a] that contrastive functionality is much more prevalent than redundancy in certain types of multimodal interaction.

Error Sensitivity Analysis

It is important to discover what factors influence the interpretation accuracy. For this purpose it is necessary to consider only the accuracy measured on the transcribed data to eliminate the effects of recognition errors.

Although the collected data samples were not, strictly speaking, drawn independently according to a fixed probability distribution, it is still useful to apply Equation (16) on page 121 to estimate how much confidence we should put in the accuracy figures reported in Table 16. Using $n = 365$ and $N = 95$, the 95% confidence interval for the interpretation accuracy on transcribed data turns out to be $80.2 \pm 4.1\%$. Thus the true accuracy could be anywhere from 76% to 84%.

The interpretation accuracy on data generated from the input model was $91.3 \pm 1.7\%$. This can be viewed as the level of performance that can be attained when the data fits the input model extremely well. The performance for the collected data is about 10% below this level, indicating that the collected data does not fit the input model perfectly.

Table 19 on page 133 breaks down the interpretation accuracy for each of the 10 participants.

As shown in the third column of Table 19, the overall interpretation accuracy is 80.2, but the accuracy for individual participants ranges from 71.4 to 89.2, indicating that the interpretation algorithm works much better for some users than for others. For two participants the accuracy approaches the level attained on artificially generated data. An analysis of the input transcriptions reveals that, not surprisingly, the algorithm achieves higher accuracy when a participant's verbal and gestural expressions match the input model more closely. This observation underlines the importance of anticipating all likely

input variations during the construction of the input model and collecting real user data to refine the input model appropriately.

Participant	Samples	Interpretation accuracy (%)		
		Top 1	Top 2	Top 3
f.4	35	71.4	88.5	88.5
m.1	47	74.4	87.2	89.3
m.2	43	76.7	86.0	86.0
m.4	37	78.3	86.4	86.4
f.5	35	80.0	97.1	97.1
f.6	34	82.3	94.1	94.1
f.3	43	83.7	88.3	88.3
m.3	31	83.8	87.0	90.3
f.1	32	87.5	87.5	87.5
f.7	28	89.2	96.4	96.4
Overall	365	80.2	89.5	90.1

Table 19. Interpretation Accuracy for Each Participant

The fourth column of Table 19 reveals an interesting fact. For some participants, most notably f.5 and f.6, the best hypothesis is correct only 80-82% of the time, but the top two hypotheses contain the right answer 94-97% of the time. In fact, among the cases where the best hypothesis is incorrect but the second best hypothesis is correct, in almost half of them the scores of the top two hypothesis are very close and the correct hypothesis receives a lower score only because of a slight discrepancy compared to the input model. In these cases, a slight adjustment to the input model (e.g., adding a synonym or an alternative phrasing, as described in section 7.1.3) is enough to add some strong associations that increase the score of the correct hypothesis, moving it from the second best to the best position.

The causes of the interpretation errors can be classified into the following categories:

- *Missing keywords or salient fragments.* The input event contains words or fragments that must be very important for classification, but the input model does not cover these salient tokens for the action frame in question. For example, the grammar for *ZoomIn* is missing a few synonyms for "zoom in," such as "expand."
- *Unanticipated phrasings.* The input event does not include any really important keywords or fragments that are missing from the input model; however, the words are arranged somewhat differently compared to the input model, in

enough to weaken certain associations and lower the score of the correct output. For example, the input model covers "Display CMU using a two-mile square" but not "Display two miles on each side of CMU."

- *Cross-associations.* The input event contains words or fragments that are associated with more than one output classes, and the sequence weights in the input model do not fit the actual input distribution, so that an incorrect output receives stronger associations than the correct output. For example, the word "show" and its synonyms should be mainly associated with *Find* and *FindAll*; however, *ZoomIn* also has a "show" phrase that receives too high a weight in the input model, causing many *Find* commands to be misclassified as *Zoom*.

- *False salient fragments.* The input event contains words or fragments that happen to occur in only certain parameter slots in the input model, although from a common sense point of view those words or fragments should not have such biased associations. For example, "Zoom out ten times" is easily recognized as a *ZoomOut* command, but "Zoom out the map ten times" is always misclassified! It turns out that in the input model "the map" occurs in *ZoomIn* and *ZoomBox* but never in *ZoomOut*. This oversight causes "the map" to become a false salient fragment that significantly reduces the score of *ZoomOut*.

- *Non-monotonic dependency*. The mathematical foundation of the MS-MIN includes an assumption that the label of a segment depends only on past and present segments, not on subsequent segments. A few input events in the collected data are incorrectly classified because of a violation of this assumption. For example, "Show the route from CMU to University of Pittsburgh and the distance" is supposed to be segmented as *FindPathLenSrc–FindPathLenDst*; however, at the time "Show the route from CMU" is assigned a label, the keyword "distance" has not yet been seen, causing *FindPathLenSrc* to receive a much lower score than *FindPathSrc*. As a result, the whole input event is classified as *FindPath* instead of *FindPathLen*, and the executed action does not display the distance information for the path.

- *Bad gesture encoding*. A few input events revealed errors in the code that converts gesture recognition results to pen input tokens. For example, a small line or arc on top of a map object should have the same effect as a simple pointing action (the user simply moved the finger a little by accident while pointing). Instead, the encoded input stream contains two tokens that refer to the same object, causing the object to appear in two different parameter slots. Such programming errors are easily repaired once they have been identified.

- *Unknown causes*. The input event appears to conform to the input model but the result is still incorrect. This is not really surprising because the interpretation accuracy is not 100% even with data generated from the input model.

Figure 35 on page 135 shows the distribution of the 19.8% error rate among the listed causes.

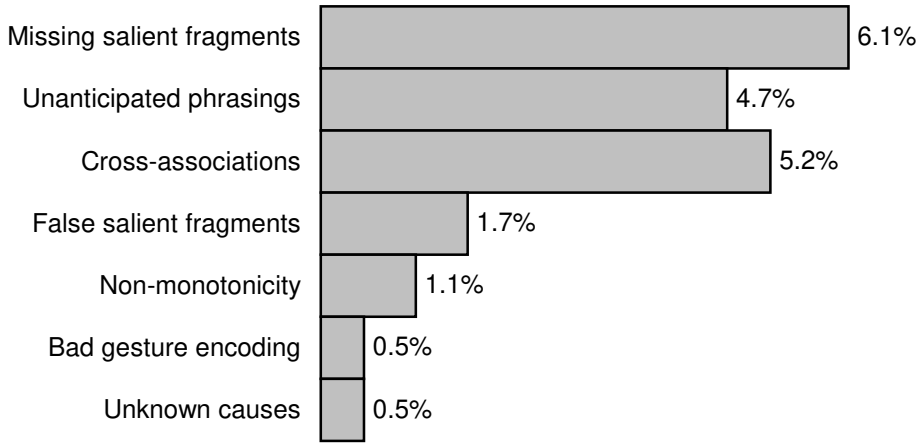


Figure 35. Distribution of Interpretation Error Categories

More than four fifths of the errors (16% out of a total of 19.8%) belong to the first three categories. The errors in the first two categories, as well as errors due to false salient fragments, can be reduced or eliminated by adding synonyms and alternative phrasings to the input model. Eliminating cross-association errors requires adjusting the probabilities in the input model. If the incorrect associations are obvious upon inspection, they can sometimes be fixed by manual adjustment of the weights; otherwise a large amount of real data may be necessary for probability estimation.

Non-monotonic dependencies, which violate an assumption in the derivation of the MS-MIN, are actually rather rare, occurring in only 1.1% of the collect data. Moreover, all the observed instances resulted from self-repairs by the participants. For example, a participant began a command with "Show me the route from CMU to University of Pittsburgh," then suddenly remembered that the task called for requesting the travel distance and thus added "and the distance" after a short pause.

Impact of Interpretation Errors on Task Completion

Measuring performance by calculating the interpretation accuracy does not provide a complete picture of how interpretation errors impact the successful completion of the tasks. One way to remedy this is to determine how many times the participants had to repeat or rephrase their requests before the application successfully interpreted their intentions. Figure 36 on page 136 shows the distribution of this measure.

The charts in Figure 36 were compiled using 344 out of the 365 collected input events. 21 input events were excluded because their manually labeled semantics did not correspond to the action and parameters required to accomplish the corresponding tasks. In these cases the participants either asked the wrong question or failed to supply all the information needed to complete the tasks, hence the application could not be expected to produce the desired response. The excluded input events cause a discrepancy between the

previously reported interpretation accuracy (35.8%) and the relative frequency of task completion with one single request (42.1%).

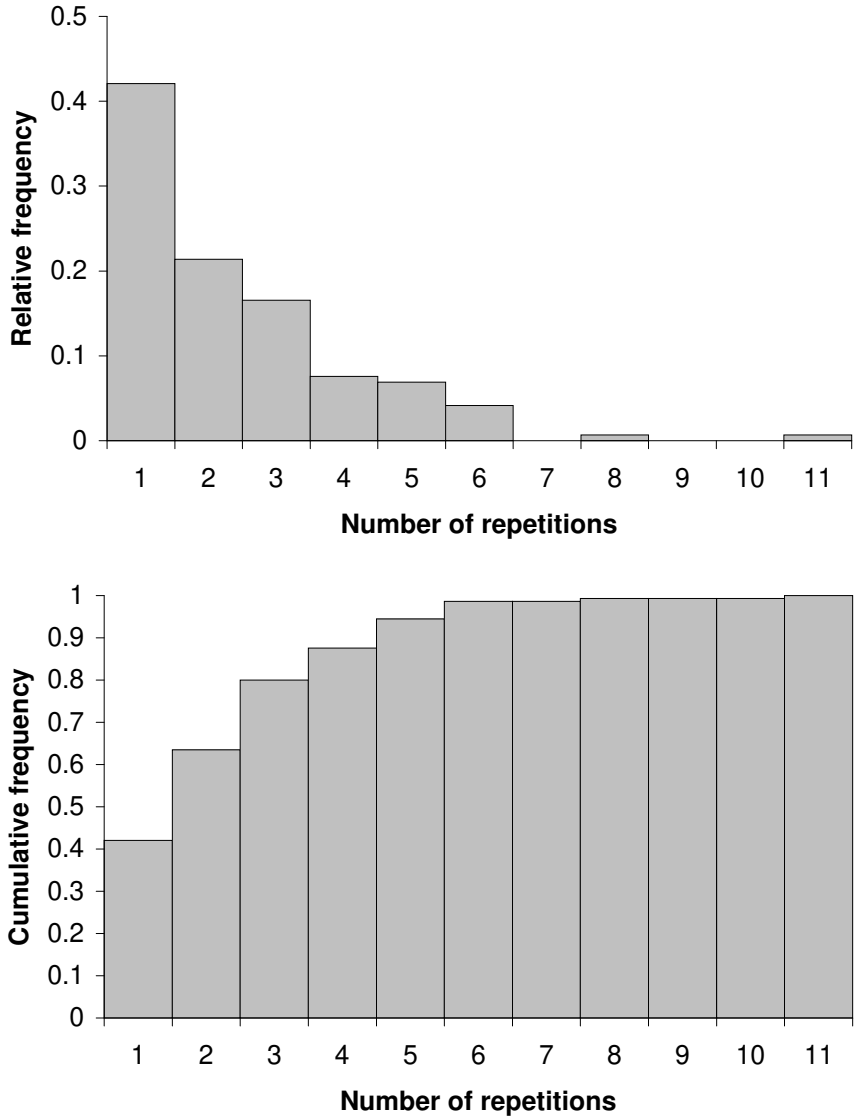


Figure 36. Number of Multimodal Commands to Complete a Task

According to the charts, more than 40% of the time the application responded correctly the first time. Over 60% of the tasks are completed within two requests, and the first three requests cover 80% of the tasks. The average number of repetitions is 2.4, and the median is 1.5. In two extreme cases, it took the participants up to 8 and 11 requests, respectively, to complete the tasks.

Reducing the number of times the users are forced to correct the system is crucial to achieving user acceptance. The improved speech and gesture recognition rates after adaptation raised the interpretation accuracy by 11%; thus we can expect the single-

request task completion rate to increase above 50% with the new recognizers. With perfect recognition, the users could be expected to complete each task with a single request more than 80% of the time.

Incremental Learning

The incremental learning test described in section 7.2.1 was repeated with real data in place of the artificially generated data. Starting with an empty network, each input sample (using transcribed inputs to simulate perfect recognition) is evaluated before and after the network is incrementally trained on that sample. The samples from each participant were presented to the network in the order they were collected to simulate what would have happened if the users had taught the network the correct response after each multimodal command.

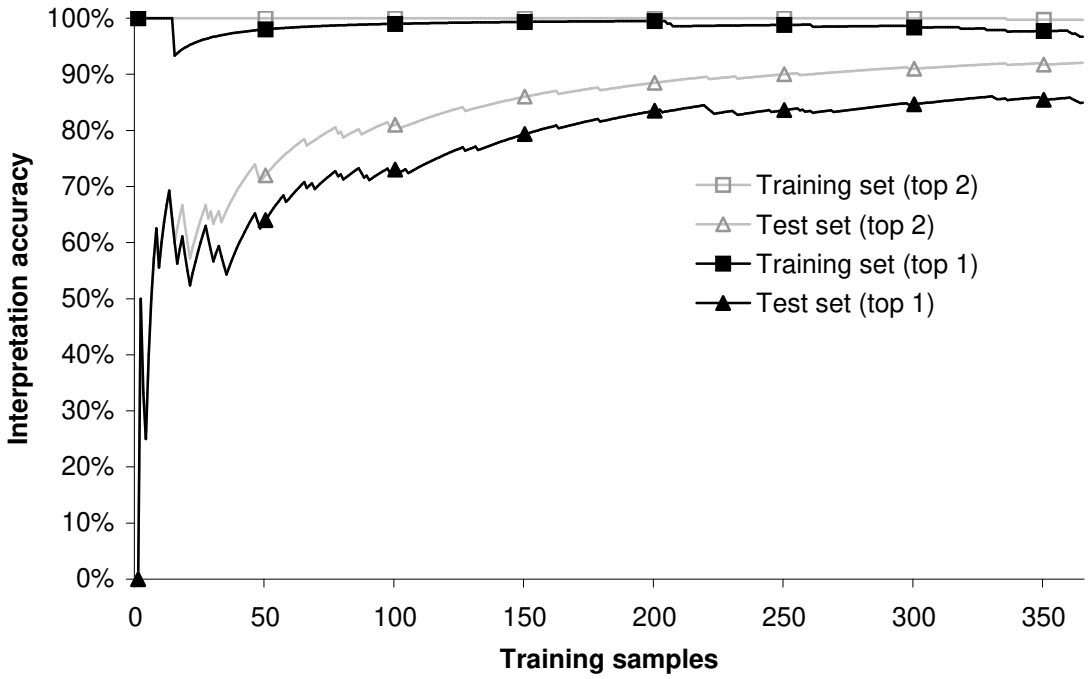


Figure 37. Incremental Learning with Real Data

Figure 37 shows the result of incremental training. Compared to the graph in Figure 30 on page 122 (obtained from artificially generated data), Figure 37 indicates a much higher learning rate: the network needed only about 300 examples, not thousands, to reach 85% accuracy. This is because the collected data represents only a small subset of the output action space, namely the subset containing the 12 tasks in the user observation protocol.

At any rate, the network exhibited much the same behavior with real data as with artificially generated data. Over 95% of the examples were immediately learned after their presentation to the network. The “test set” curves show steadily increasing performance on new data, indicating good generalization.

Figure 38 documents the vocabulary acquisition rate during training. The network growth in terms of input units and connections is depicted in Figure 39.

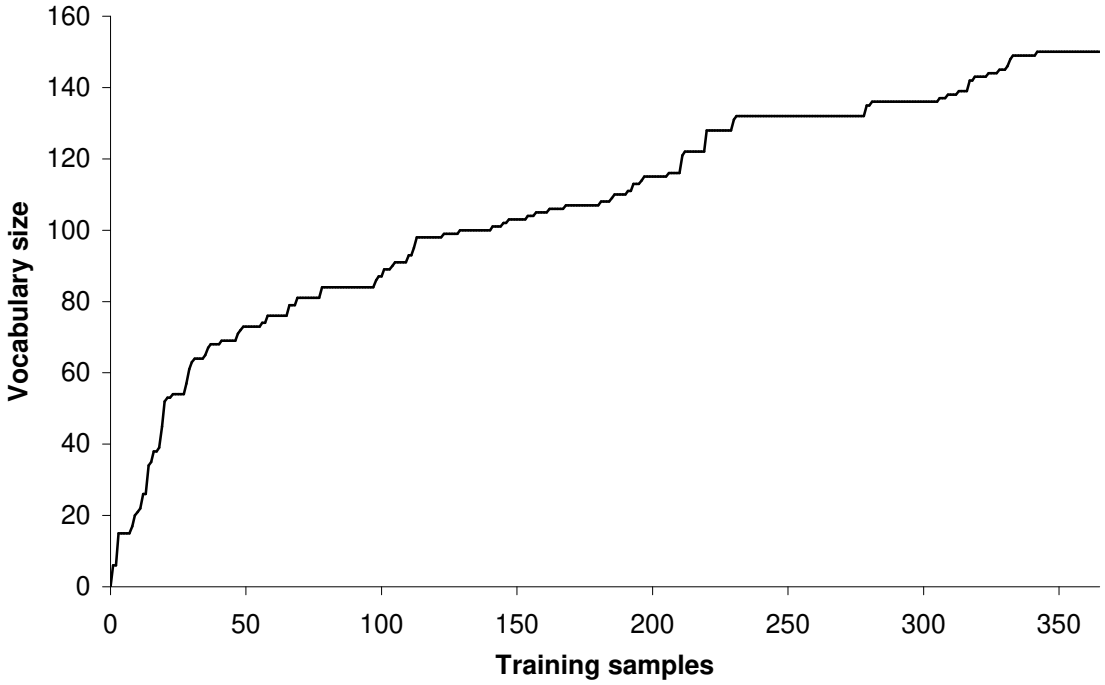


Figure 38. Vocabulary Acquisition with Real Data

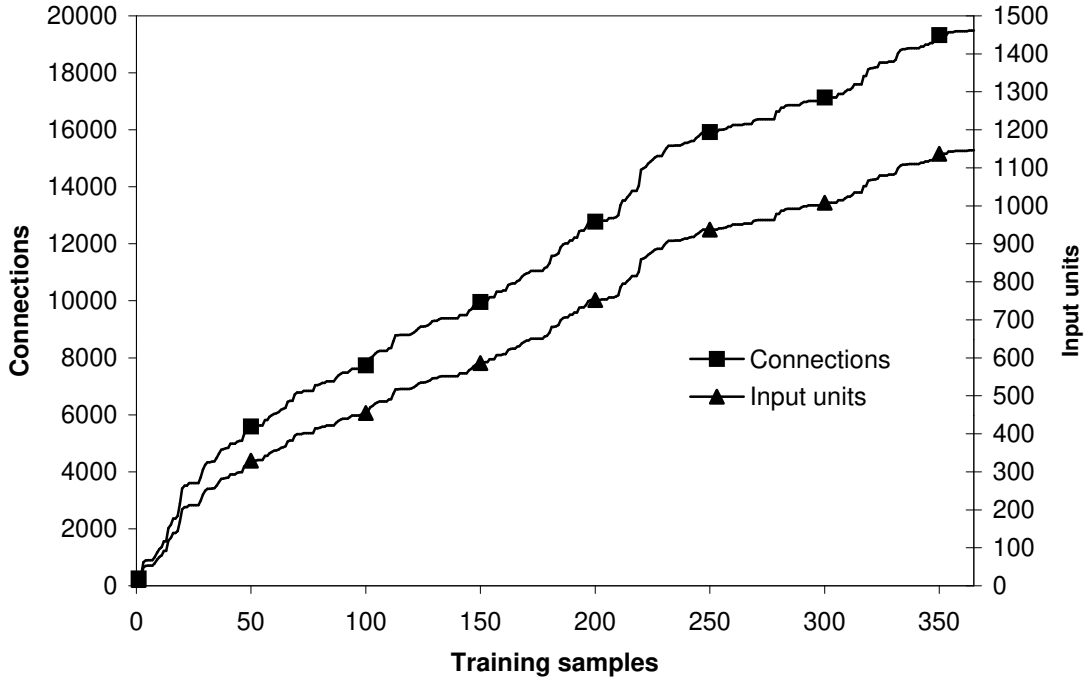


Figure 39. Network Growth with Real Data

Instead of starting from an empty network, we could generate a network from an MMGL input model and incrementally train it further with real user data for the application. Figure 40 charts the incremental learning progress of a generated network on the previously described set of collected input events.

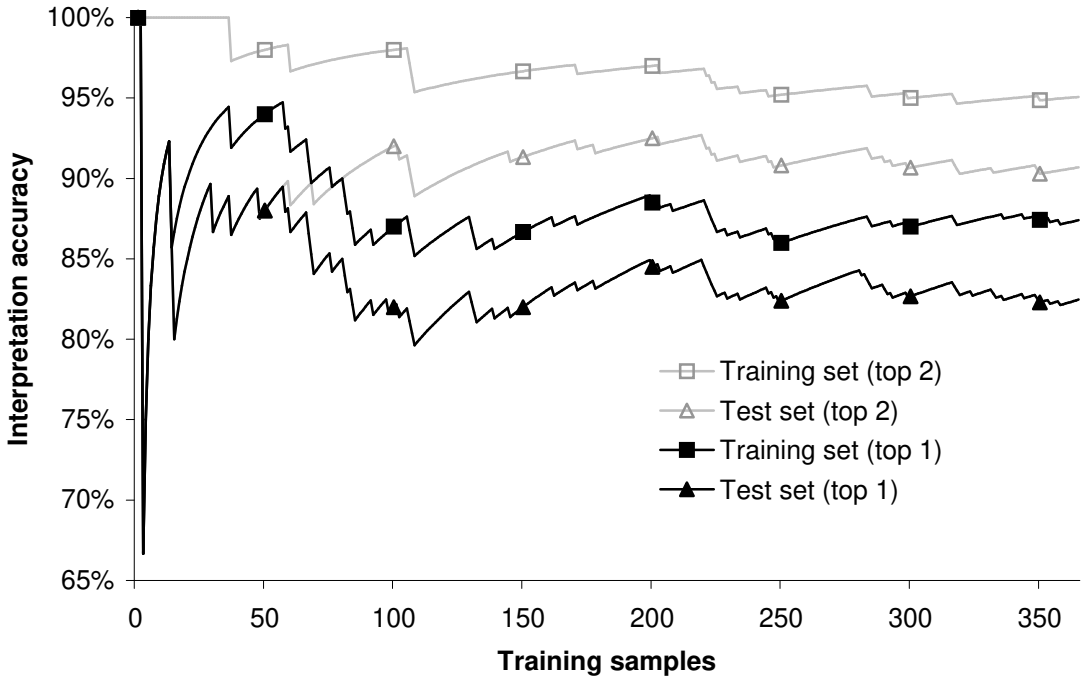


Figure 40. Additional Incremental Training of Generated Network

The graph shows that the interpretation performance dropped sharply at first as the network encountered input events not covered by the MMGL input model, but very quickly rose again to a more or less stable level through the rest of the data set as the network assimilated the new data patterns. At the end of the training session, the "test set" interpretation accuracy is 82.5%, slightly higher than the 80.2% accuracy obtained before incremental training (see Table 16 on page 130). The "training set" accuracy is 87.4%. These results show that it is feasible to bootstrap the system using an MMGL input model, then incrementally train the semantic integration network as user data becomes available.

Chapter 8
CONCLUSION AND FUTURE DIRECTIONS

This chapter summarizes the contributions of this dissertation and explores some directions for future research.

8.1 Contributions

The work in this dissertation was motivated by three challenges in constructing multimodal applications:

- The development of a domain-independent algorithm to interpret multimodal inputs;
- The design of a common infrastructure that supports a broad class of multimodal applications to avoid unnecessary development effort;
- The rapid prototyping of a multimodal application by instantiating the application structure and multimodal interpretation algorithm for a specific domain.

Accordingly, this dissertation makes contributions in three main areas:

- Theory of multimodal interaction;
- Software architecture and reusable framework for the construction of multimodal applications;
- Design process and supporting tools for the construction of multimodal applications.

The contributions in all three areas are intimately related to each other through the unifying theme of a multimodal semantic model that supports a broad class of applications, namely those that accept input data from multiple information streams and allow their states to be modified by parameterized actions.

The three areas of contributions are discussed in turn below.

8.1.1 Theory of Multimodal Interaction

The foundation of all the contributions in this dissertation is the model of multimodal interpretation based on the semantic integration of multiple information streams.

Chapter 3 defines multimodal interpretation as the mapping from multimodal input events to their semantic values. For the applications targeted by this work, the semantic values are the parameterized actions that should be performed in response to the input

events. Inputs from all modalities are considered information streams, or sequences of tokens that give information about the likely output actions.

The proposed multimodal semantic model requires a segmentation of each input stream and an alignment of cross-modal segments to form parameter slots in an action frame. Each parameter slot contains information that constrains the value of a single parameter of the action represented by the action frame.

Data fusion, or integration of information from multiple modalities, happens at the parameter slot level. The cross-modal alignment of segments to form parameter slots is based on semantic rather than temporal constraints. Timing and synchronization of inputs from different channels are relegated to the lower level of grouping unimodal input events into combined multimodal input events, outside the scope of the semantic model. The semantic model can therefore work with any input grouping policy.

An algorithm for multimodal semantic integration using the above model was developed based on a connectionist network architecture, the Multi-State Mutual Information Network (MS-MIN). The MS-MIN represents associations between input tokens and output parameter slots by connection weights that measure input-output mutual information. Based on these associations, the network computes an input segmentation and a corresponding label assignment to produce the most likely parameter slot sequence given the input.

The MS-MIN can be trained from examples much faster than traditional backpropagation neural networks. Given an input model that describes the distribution of input messages and their associated semantic values, it is possible to compute the connection weights directly, avoiding time-consuming training altogether. Furthermore, the MS-MIN can learn incrementally, thus improving its performance during actual use.

8.1.2 Software Architecture and Reusable Framework

A significant amount of work in the development of multimodal applications concerns the mechanics of assembling system components that remain the same across applications. Accordingly, the second area of contributions in this dissertation is the creation of a reusable infrastructure for multimodal application development.

Chapter 4 describes the Multimodal Application Framework (MMApp), a collection of reusable components and a system architecture that together form a modular, distributed, customizable infrastructure for multimodal applications.

The design of MMApp follows established principles of object-oriented software engineering and exploits design patterns to achieve a high degree of modularization. The major system components are structured as interfaces that expose a well-defined set of operations. Depending on the needs of a particular application, the application developers can select any suitable implementations of the interfaces without affecting other parts of the system. In other words, the modules are plug-replaceable; i.e., any module can be

replaced by an equivalent module that implements the same interface. MMApp defines interfaces for recording and recognizing speech and pen inputs as well as synchronizing and coordinating multimodal input events.

The modularity of MMApp allows system components to be distributed among multiple computers and processes. Computation-intensive components can run as servers on high-end workstations and export their services to user interface components running in Web browsers anywhere on the network. MMApp components communicate with one another via a communication layer that supports location-transparent access to services.

A philosophy that pervades the design of MMApp is to provide reasonable default implementations for everything and at the same time allow application developers to customize all aspects of the system to suit particular needs. Hiding components behind well-defined interfaces plays an important role in accomplishing this. Other techniques to promote customizability include class inheritance, object factories, and template methods to specify algorithm skeletons.

8.1.3 Design Process and Supporting Tools

The MMApp framework handles the assembly of reusable components within a system architecture; however, without a module that interprets multimodal inputs in the target application domain, the resulting application shell would be like a computer without an operating system, or a body without a brain. The third area of contributions in this dissertation is a design process backed by a set of tools to instantiate a multimodal application for a particular domain.

Chapter 5 describes the design process supported by the Multimodal Toolkit (MMTk) for rapid prototyping of multimodal applications. The basis of the design process is the creation of an input model based on the Multimodal Grammar Language (MMGL) to specify a set of multimodal input messages, their probability distribution, and their associated semantic values according to the action frame/parameter slot semantic model.

MMTk contains a visual grammar editor that allows application developers to create and modify MMGL input models using a drag-and-drop visual construction paradigm. This approach frees grammar writers from the complexity of language syntax and lets them concentrate on the contents of the grammar specification.

From an input model, the tools in MMTk can generate a speech recognition language model and instantiate a multimodal interpreter for the application. The multimodal interpreter consists of a preprocessor to parse macro concepts from the input streams, a semantic integrator based on the MS-MIN, and the skeleton of a postprocessor to extract parameter values. Application developers only have to customize domain-specific parts of the postprocessor to obtain a complete multimodal interpreter for the application.

8.2 Future Directions

This section discusses remedies to the deficiencies of the current work as well as extensions that will make good subjects for future research.

The semantic integration approach requires input streams to be partitioned into unimodal input events which are combined into multimodal input events before interpretation takes place. This scheme is flexible in that the input partitioning and grouping policies are completely unspecified and the selection of these policies has no impact on the semantic integration stage. However, as integrated segmentation and recognition has been shown to improve speech and handwriting recognition performance significantly [Haffner92][Manke95], future research should explore approaches that can determine the boundaries of multimodal input events and derive their semantic values at the same time. One possible avenue of exploration is the addition of a dynamic programming stage on top of a semantic integrator to compute the best path through the input space, similar to the state layer in the MS-MIN but possibly with different scoring criteria.

Semantic integration with action frames and parameter slots occurs at an intermediate symbolic representation level, after recognizers have transformed the raw input signal into a more convenient symbolic representation but before any semantic interpretation has been assigned to the unimodal input. [Nigay95], [Vo96], and [Johnston97] describe multimodal integration approaches at a high level of representation, after input from each modality has been parsed and assigned a (possibly incomplete) semantic interpretation. It is possible to handle unimodal recognition errors in integration at the symbolic or partial interpretation levels by generating multiple recognition hypotheses ranked by confidence. However, future research should also explored two other data fusion strategies:

- Fusion of low-level input signals to arrive directly at a semantic value without intermediate symbolic representations such as words or gesture shapes;
- Hybrid multi-level fusion that integrates recognition and semantic interpretation using a joint multimodal language model, such that the recognition and semantic interpretation of one modality can influence the recognition and semantic interpretation of another modality.

The work presented in this dissertation can accommodate many input modalities, but the current implementation of the framework strongly supports only speech and pen modalities. Future versions should define interfaces and provide implementations of components that strongly support other modalities, including lip-reading, 3D gestures, gaze tracking, face tracking, etc.

The MMApp framework contains components that can be assembled to construct a multimodal application. The MultimodalApplet component is a customizable user interface that integrates the MMApp components for speech, pen, and multimodal coordination. However, if the layout or the structure of this default user interface is not appropriate for an application, or if additional modalities must be accommodated, application developers

will have to construct a different user interface using basic MMApp components. This process would be much more convenient with the addition of a visual interface builder.

The MultimodalApplet employs a simple undo scheme that allows the cancellation of the last executed command. However, in many applications, some commands may be very costly or impossible to undo. In these cases, it would be better to replace undoing with a scheme that lets the user confirm the interpretation of a command before executing it. It is also possible to implement implicit confirmation by treating the issuance of the next command as silent confirmation of an uncorrected previous command.

Error recovery and repair techniques, especially multimodal approaches that allow cross-modal error repairs as in [Suhm97], would be a valuable addition to the MMApp framework.

The template-based gesture recognizer in the current implementation of MMApp was intended only as a concept demonstration. The recognition algorithm suffers from inherent weaknesses that prevent it from achieving high accuracy. Future implementations should include a better gesture recognizer. Furthermore, algorithms for distinguishing gestures from handwritten words in pen input should be explored to improve the integration of gesture and handwriting recognition.

The Grammar Designer in MMTk is functionally complete; i.e., any MMGL grammar can be created and modified in the Designer. However, there is still room for improvement in the Designer user interface to render some common operations faster and more convenient. Useful improvements include shortcut notations for optional and repeating grammar nodes, the ability to move or copy elements from one part of a grammar to another by dragging and dropping (the current version allows only prototypes to be dragged and dropped), the ability to edit multiple grammars at the same time and copy data among them, and a library of reusable MMGL nodes that can be incorporated into the input models of many applications.

The N-gram Language Model Generator could be extended to produce more compact class-based language models. The current version keeps all trigram weight tables in memory and thus cannot accommodate very large MMGL input models; this could be remedied with a disk-based caching scheme. A Finite-State-Grammar Language Model Generator would also be a good addition to the toolkit.

Appendix A
GLOSSARY

Action frame. Sequence of parameter slots specifying a parameterized action.

Alignment. One-to-one correspondence between elements of two or more groups. The multimodal semantic model in Chapter 3 aligns segments from different input modalities to form parameter slots.

Applet. Program that is downloaded via the network and runs inside a Web browser.

Application. Computer program.

Application framework. A system architecture and a collection of reusable components that facilitate the construction of certain types of applications.

Application Programming Interface (API). Specification of how to access the functionality of a system (usually in terms of function calls that the system supports).

Backpropagation. Neural network training algorithm which performs gradient descent by computing errors at the output and proceeding backward through the network to calculate weight updates.

Bigram model. N-gram model with N=2.

Clicking/dragging. Ways of using the mouse in a graphical user interface. Pressing then releasing a mouse button is called clicking; moving the mouse while holding down a button is called dragging.

Client/server. Architecture for distributed computing based on a request-response paradigm. A client accesses the functionality of a server by sending it a request and processing the corresponding response from the server.

Connectionist network or **Neural network.** Structure that relies on massively interconnected simple processors (liken to neurons in the brain) to perform complex computations.

Data fusion. *See* **Input integration**.

Deictic. Pointing or referring to something. Examples: “this,” “that,” etc. in speech; pointing with a finger or pen tip.

Design patterns. Object-oriented design solutions encapsulated in a general, reusable form.

Distributed processing. Style of computing that divides a computation into many parts, each running on a different machine.

Drag-and-drop. Style of interaction in a graphical user interface, consisting of clicking on a screen object and dragging the mouse (while holding down a button) to the target location, then releasing the mouse button to drop the object at the new location.

Dynamic programming. Programming technique to compute certain recursive functions by eliminating the recursion and recording intermediate values in a table. It is widely used in optimization problems such as optimal alignment, in which the path followed to reach the optimal value is as important as the value.

Dynamic Time Warping (DTW). Dynamic programming algorithm to align a temporal sequence with a series of labels. It is used in speech recognition to produce the best sequence of phonemes or other speech units given the acoustic evidence.

Face tracking. Technique of making a camera automatically follow the movement of a person's face and deliver a constant-sized image of the face.

Fragment. Group of tokens in an information stream.

Gaze tracking. Technique of automatically following a person's eye movements and determining the gaze direction.

Gesture. A pen-based gesture represents pointing, signs, symbols, picture sketches, etc. obtained from the pen modality; a 3D gesture conveys information with a movement of the hand.

Gradient descent. Minimization technique that improves the target function by moving along the direction of steepest decrease indicated by the gradient of the function.

Graphical user interface (GUI). User interface that presents information using a visual metaphor.

Grammar. Set of rules that encode a set of admissible sentences.

Implementation. (In application design) Code that performs the operations specified in an interface.

Incremental learning. Process of improving performance by additional training on new data without having to retrain on previously learned data.

Information streams. Sequence of tokens that carry information influencing the selection of an output value.

Inheritance. Object-oriented reuse mechanism in which a subclass that inherits from a superclass automatically possesses all the data and behavior of the superclass.

Input integration or **Data fusion.** Process of combining information from multiple input sources to arrive at an interpretation.

Input model. Encoding of a set of input messages together with their semantic values, used to characterize what kind of input messages an application can expect from the users.

Input synchronization/coordination. Process of determining the boundaries and combinations of multimodal input events.

Instantiation. Creation; construction from a template, a specification, or a model.

Interface. (In application design) Specification of operations supported by a system, without committing to any particular implementation of those operations.

Interpretation or **Understanding.** Mapping from an input message to a semantic value.

Language model. Encoded knowledge that guides the recognition process using the relationships between linguistic elements.

Lip-reading. Technique of recognizing speech by analyzing lip movements.

Multimedia. Supporting multiple channels of information.

Multimodal. Supporting multiple input modalities.

Multimodal input event. Group of unimodal input events that are interpreted together.

Multithreading. The use of multiple threads in a program for parallel processing.

Modality or **Mode.** Input channel.

Neural network. *See* **Connectionist network**.

N-gram model. Language model that predicts the likelihood of encountering a word based on N previous words in the sequence.

Object-orientation. Design and programming paradigm based on the use of objects and classes to represent entities with state and behavior in a design.

Parameter slot. Cross-modal segment of a multimodal input event, specifying a parameter in an action frame.

Parsing. Process of analyzing a sentence and labeling its constituents using a grammar.

Pen. Input modality in which information is conveyed by inputs from a digitizing device.

Polymorphism. Dynamic binding of methods that enables the same method call to resolve to different methods depending on the actual type of the target object at runtime.

Process. Computer program running in an independent address space.

Recognition. Process of mapping input signals to a symbolic representation.

Recording. Process of capturing input signals in a form the computer can manipulate.

Segmentation. Division of a sequence into contiguous segments based on some criteria. The multimodal semantic model in Chapter 3 segments the input streams and aligns them to form parameter slots.

Semantic model. Encoding of knowledge about interpretation. The multimodal semantic model in Chapter 3 assigns semantic values to multimodal input events using an action frame/parameter slot specification.

Semantic value. Output value assigned to an input message by the interpretation mapping.

Socket. Channel used to send data between processes.

Speech. Input modality in which information is conveyed by spoken utterances.

Temporal proximity. Input synchronization/coordination model based on how close in time the input event occurrences are.

Thread. Independent, parallel execution of program code within the same address space.

Token. Information-carrying unit in an information stream.

Toolkit or **Workbench.** Collection of software components that work together to assist programmers in some task.

Trigram model. N-gram model with N=3.

Understanding. *See* **Interpretation**.

Unimodal. Referring to a single input modality.

Unimodal input event. Contiguous segment from a single input stream.

Web browser. Program that presents information downloaded from the World Wide Web.

Wizard of Oz. Simulation paradigm in which a hidden operator controls the operations of the system under study.

Workbench. *See* **Toolkit**.

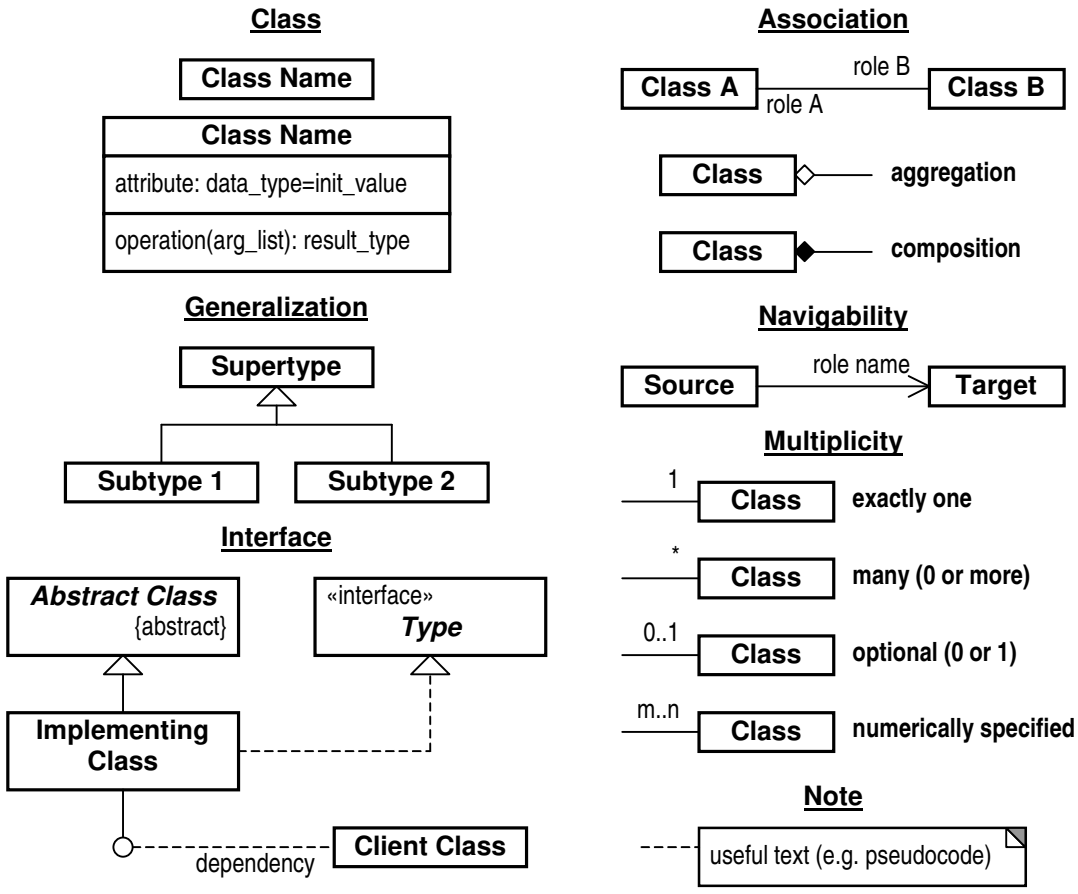
World Wide Web. Part of the Internet that relies on the Hypertext Transfer Protocol (HTTP) to transfer multimedia information.

Appendix B
SUMMARY OF THE UNIFIED MODELING LANGUAGE

The Unified Modeling Language (UML) [Fowler97] is a standard system of notation for specifying, visualizing, and constructing the artifacts of software systems. It unifies the methods of Booch [Booch93], Rumbaugh [Rumbaugh91], and Jacobson [Jacobson92] for object-oriented analysis and design.

UML notation is used in all the class diagrams in Chapter 4 and Chapter 5, as well as in the description of design patterns in Appendix C. The following is a summary of UML notation regarding class diagrams only; UML covers a much wider range of topics, from use cases to behavior and implementation diagrams.

For brevity, the {abstract} property indicator for abstract classes is omitted from the class diagrams in this dissertation; instead, italicized class names are used to indicate abstract classes. Likewise, the names of abstract operations are also shown in italics.



Appendix C
SUMMARY OF DESIGN PATTERNS

Object-oriented design patterns, as defined in [Gamma95], are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." The software components in the MMApp (see Chapter 4) and MMTk (see Chapter 5) libraries contain applications of several design patterns summarized below. The materials presented in this appendix, including the class diagrams, are largely adapted from [Gamma95].

C.1 Abstract Factory

The Abstract Factory pattern provides an interface for creating objects without specifying their concrete classes. This pattern is useful when a system should be independent of how its products are created, composed, and represented. Only the interfaces of the products are revealed, not their implementations. Systems using the Abstract Factory pattern can be configured with one of multiple families of products, and the way the products in a family are used together is enforced by the structure of the pattern.

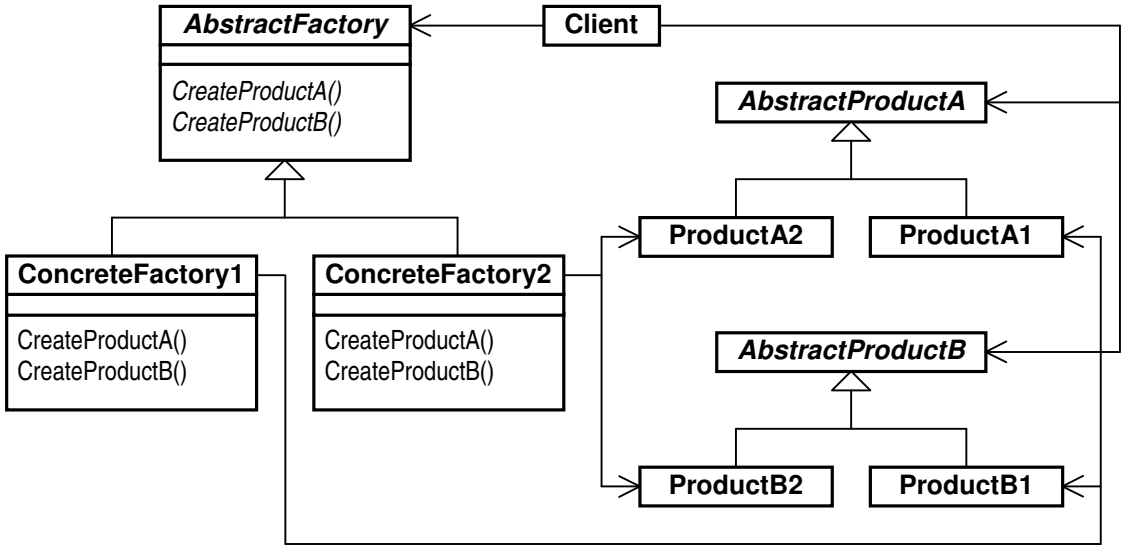


Figure 41. The Abstract Factory Design Pattern

In Figure 41, AbstractFactory declares an interface for operations that create AbstractProduct objects. Each AbstractProduct class declares an interface for a type of product objects which are implemented in Product classes. Each ConcreteFactory is an implementation of AbstractFactory that produces a different family of Products. The Client uses only interfaces declared by AbstractFactory and AbstractProduct, and never has to concern itself with the actual object types.

The Abstract Factory pattern is used in many user interface toolkits, including the Java Abstract Window Toolkit (AWT), to achieve portability across different window systems by using the same interface to produce different families of interface components (widgets). MMApp applies the Abstract Factory pattern to hide the implementation of the transport protocol used in interprocess communication (see section 4.6.2).

C.2 Adapter

The Adapter (also known as Wrapper) pattern converts the interface of a class into another interface clients expect. It is useful when an existing class to be reused does not have an interface compatible with a required interface. The pattern presented here is the object adapter version which relies on object composition; there is also a class adapter version which uses multiple inheritance to adapt one interface to another.

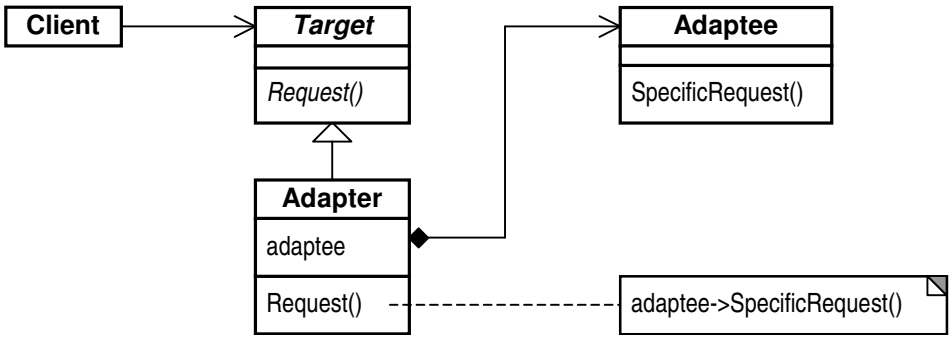


Figure 42. The Adapter Design Pattern

In Figure 42, Target is the (domain-specific) interface that Client uses, and Adaptee defines an existing interface that needs adapting. Adapter adapts the interface of Adaptee to the Target interface by calling on Adaptee methods to implement Target methods.

MMApp uses the Adapter pattern for wrappers around existing software components such as the SRecServer and NetscapeSRec speech recorders (section 4.4.2), the JANUS and SPHINX speech recognizers (section 4.4.3), the XPRecServer pen recorder (section 4.5.3), and the TmplGRec gesture recognizer (section 4.5.4).

C.3 Factory Method

The Factory Method pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate for the object. It is applicable when a class cannot anticipate the class of objects it must create, and therefore must delegate this decision to its subclasses.

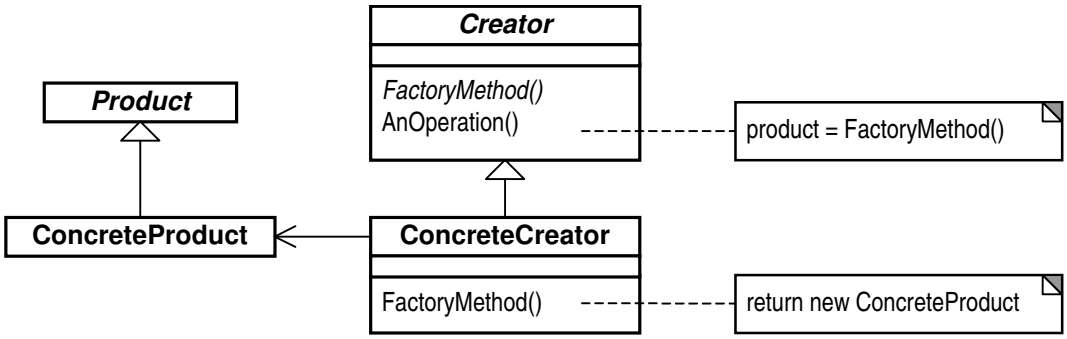


Figure 43. The Factory Method Design Pattern

In Figure 43, Product defines the interface of objects to be created, which are implemented by ConcreteProduct. Creator delegates the creation of Product objects to its FactoryMethod(), which is overridden in the ConcreteCreator subclass to instantiate the correct ConcreteProduct.

MMApp employs the Factory Method pattern in the MultimodalApplet class to permit customizable instantiation of SpeechRecorder, SpeechRecognizer, PenRecognizer, and other subordinate objects (see section 4.7.4).

C.4 Observer

The Observer pattern defines a one-to-many dependency between objects so that all the dependents of an object are automatically notified and updated when the object changes state. This approach avoids tight object coupling, so that the changed object does not have to know how many dependents need to be update or make any assumptions about what the dependents are.

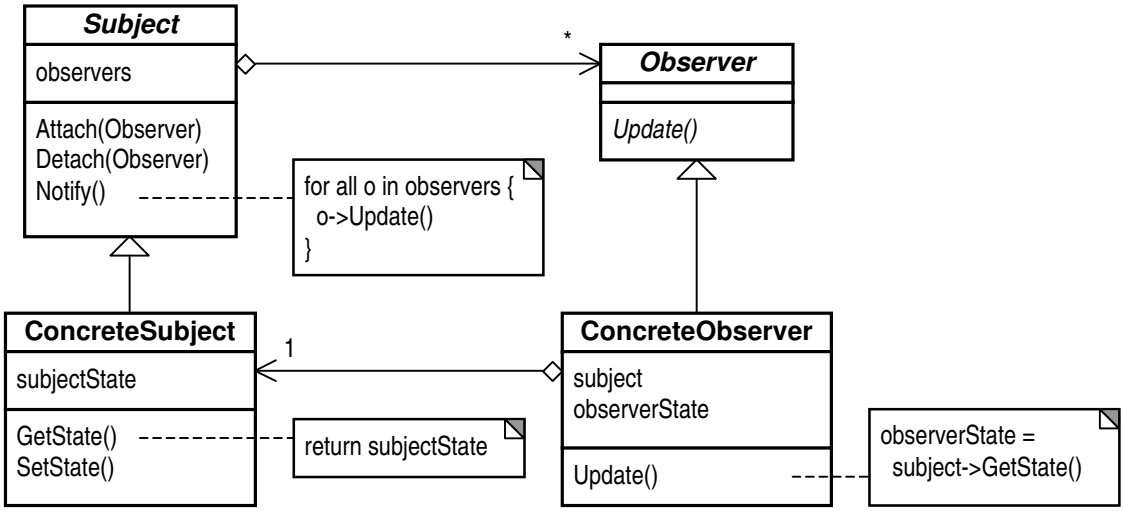


Figure 44. The Observer Design Pattern

In Figure 44, Subject (also called Observable) provides an interface for attaching and detaching Observer objects, which define an updating interface to receive notifications of state changes in Subject. ConcreteObserver implements the Observer updating interface and maintains a reference to a ConcreteSubject object so that its state can be retrieved.

The Model/View/Controller (MVC) user interface framework in Smalltalk is the best-known example of the Observer pattern. Other user interface toolkits such as InterViews, the Andrew Toolkit, and Unidraw, also employ this pattern. The standard Java class library defines an Observable class and an Observer interface. In MMTk, the Observer pattern helps decouple grammar objects from their screen representations in the Visual Grammar Designer (see section 5.3.1).

C.5 Template Method

The Template Method pattern defines the skeleton of an algorithm and defers some steps to subclasses so that these steps can be customized without changing the structure of the algorithm. The invariant parts of an algorithm are implemented once; only the parts that can vary have to be customized as needed. The Template Method controls the way subclasses can supply extensions; only "hook" operations at specific points can be extended.

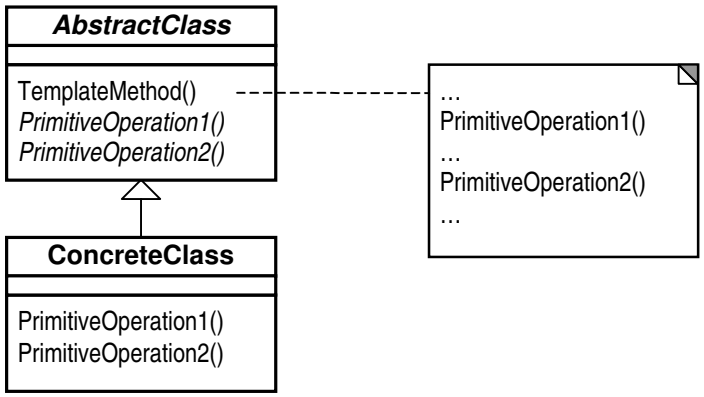


Figure 45. The Template Method Design Pattern

In Figure 45, AbstractClass defines *primitive operations* that concrete subclasses must define to implement steps of an algorithm. TemplateMethod() defines the skeleton of algorithm that makes use of primitive operations. ConcreteClass then implements the primitive operations to carry out subclass-specific steps of the algorithm.

The MultimodalApplet class in MMApp defines a template method for multimodal interpretation; subclasses only have to override three abstract methods (i.e., primitive operations) to implement application-specific behaviors (see section 4.7.4).

C.6 Visitor

The Visitor pattern represents an operation to be performed on the elements of an object structure, in such a way that new operations can be defined without changing the classes of those elements. It is applicable when the target object structure contains many classes of objects with differing interface, and the operations depend on their concrete classes. The Visitor pattern is useful when the classes defining the object structure rarely change, but there are many operations to be performed on the objects in the structure.

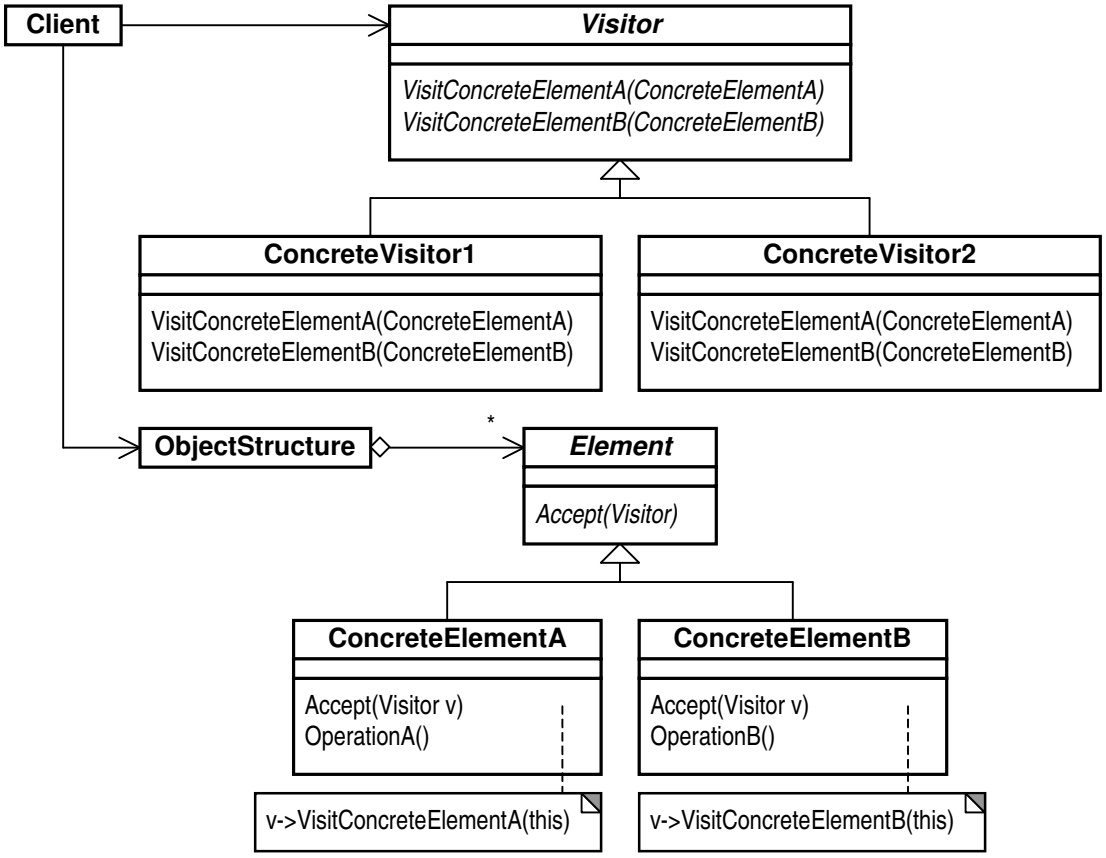


Figure 46. The Visitor Design Pattern

In Figure 46, Visitor declares a Visit operation for each class of ConcreteElement in the ObjectStructure. ConcreteVisitor implements each operation declared by Visitor to provide a fragment of the algorithm that traverses the object structure. Each ConcreteElement implements the Accept() operation declared by Element to call the appropriate Visit method in Visitor using a technique known as *double-dispatch* (which selects the right method to call based on the actual types of *two* objects, the Visitor and the Element).

The Visitor pattern forms the basis for all grammar traversal algorithms in MMTk. Operations defined as Visitors on MMGL input models include the automatic generation of random samples, N-gram language models, input preprocessors, semantic integration networks, and postprocessors for parameter extraction (see sections 5.4 to 5.6).

Appendix D
USER OBSERVATION PROTOCOL FOR THE MAP APPLICATION

This appendix describes the protocol for the user observation sessions reported in 7.2.2.

The participants were asked to read and sign an informed consent form. A sample of the form can be found on the next page.

Each participant was given two pages of instructions describing the session format and the kind of input one can expect the computer to understand. After reading the instructions, the participant sat down in front of a workstation, donned a Sennheiser headset with close-talking microphone, and proceeded down a list of twelve tasks itemized in an instruction packet.

The participants issued multimodal commands by speaking into the microphone and/or drawing gestures with a finger on the touch-sensitive screen. Speech input was configured for a "Click to talk" mode, so that the participants had to press a speech button before speaking.

The description of each task briefly states the goal of the task and gives an image of what the map display should look like after successful completion of the task. The participants were instructed to try to complete a task successfully before proceeding to the next task, and to press an "Undo" button and try again if the application responded incorrectly. The experimenter was always available for questions and clarifications.

After each session, the participant filled out a questionnaire about their subjective impression of the system.

The instructions, the task list, and the questionnaire are included in this appendix for reference.

[Gomoll90] and [Dillman78] were the main sources of consultation for the protocol and questionnaire design, with additional input from Bonnie John.

CARNEGIE MELLON UNIVERSITY

CONSENT FORM

Project Title: Multimodal Human-Computer Interaction
Conducted By: Minh Tue Vo

I agree to participate in the observational research conducted by Professor Alex Waibel or by students or staff under the supervision of Professor Waibel. I understand that the proposed research has been reviewed by the University's Institutional Review Board and that to the best of their ability they have determined that the observations involve no invasion of my rights of privacy, nor do they incorporate any procedure or requirements which may be found morally or ethically objectionable. If, however, at any time I wish to terminate my participation in this study I have the right to do so without penalty.

If you have any questions about this study, you should feel free to ask them now or anytime throughout the study by contacting:

Professor Alex Waibel
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-7676
E-mail: ahw+@cs.cmu.e du

You may report any objections to the study, either orally or in writing to:

Susan Burkett
Associate Provost
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-8746

Purpose of the Study: I understand I will be using a computer program to perform a list of tasks involving either finding information about places on a map or scheduling appointments on a calendar, using spoken commands and/or gestures drawn on the computer screen. I know that the researchers are studying how people would interact with computers using speech and gestures, in order to improve the way their computer program supports this kind of interaction. I am aware that what I say and draw on the screen during the experiment will be recorded so that the researchers can later analyze the data and use the result to correct mistakes and improve the performance of their computer program. I also realize that I will be asked to fill out a questionnaire about my subjective assessment of the program.

I understand that the following procedure will be used to maintain my anonymity in analysis and publication/presentation of any results. Each participant will be assigned a number, names will not be recorded. The researchers will save the data and answers to the questionnaire by participant number, not by name. The speech and gesture data as well as the completed questionnaires will be stored in locked files by Professor Waibel and his student, Minh Tue Vo. No other researchers will have access to these files.

I understand that in signing this consent form, I give Professor Waibel and his associates permission to present this work in written and oral form, without further permission from me.

_____ _____
Name (please print) Signature

_____ _____
Telephone Date

INSTRUCTIONS

You will sit in front of a computer screen displaying a street map of Pittsburgh, PA. You will wear a headset that allows you to speak to the computer. You can also draw directly on the computer screen using your fingertip. The computer will help you carry out a list of tasks that the experimenter will give to you. You can think of this computer as an intelligent assistant to whom you give verbal instructions (possibly accompanied by scribbles on the map if you find it easier to express certain things that way). The assistant can adjust the map display for you by

- zooming in to magnify certain areas and show more details; you can specify
    - a magnification factor
    - a center point and a square area of a certain size around it
    - an arbitrary area you specify visually on the map
- zooming out by a certain factor to show a wider view with less details
- panning to shift the view in any of 8 directions (north/up, south/down, east/right, west/left, northeast, southeast, northwest, southwest) by a certain percentage of the screen or a certain distance

The map assistant can locate places on the map by address, name, or certain characteristics, and tell the map to display those places as icons (small pictures with labels). The assistant can also find the best way (the shortest and fastest route) to go from one place to another, how far apart those places are, and how long it would take to traverse the best route.

The computer can understand a limited repertoire of verbal and visual cues. The purpose of this study is to find ways to improve the program, so please be patient if it does not understand everything perfectly yet. Remember, we are testing the computer program; we are *not* testing you! Even when the program fails to do what you want, the recorded data is valuable to us.

For each of the assigned tasks, study the instructions to make sure you know what the intended result should be. When you are ready, press the square button with a microphone icon at the bottom left corner to tell the computer to listen to you, then speak and/or draw on the screen. The computer will automatically stop listening when you finish speaking. *Turn to the next task on the list only if the result is correct according to the instructions.* If not, press the "Undo" button at the bottom right corner of the display and try again until the computer gets it right.

Thank you for your participation.

WHAT THE COMPUTER CAN UNDERSTAND

You can ask a question or tell the program to do something. For example, both *"Where is the Oriental Kitchen?"* and *"Find the Oriental Kitchen"* are OK. The program usually understands several synonyms, so "find," "locate," "show me," "display," etc. are all equivalent. You can also use natural, conversational sentences as if you were speaking to a person instead of issuing computer commands. *"Please, can you tell me where University of Pittsburgh is?"* should work as well as *"Find University of Pittsburgh."*

If a place is displayed on the map, you can refer to it by name or point it out visually by touching it or circling it on the map display. *"How to I go to 5000 Forbes Avenue from here?" + pointing at or circling Duquesne University* on the map is another way of expressing *"How do I go from 5000 Forbes Avenue to Duquesne University?"* If your question/command is about the fastest route or the distance between two places (both displayed on the map), you can also draw a line or an arrow from one place to the other on the map in conjunction with your spoken words, something like *"How far is it from here to there?" + line/arrow between the two places.*

When you need to specify a rectangular area on the map, do it visually by circling or drawing a box around the desired area. An approximation is fine; you don't have to get it right down to within millimeters or anything like that.

To increase the magnification, i.e., expand a small area to fill the whole display, you can use words like "zoom in," "magnify," "enlarge," "expand," "more detail," etc. To specify a center point for the zoom, add something like "around (or at/on/…) <placename>," or refer to it visually as described above. To specify a magnification factor (e.g., 2), say something like "2 times" or "by a factor of 2," etc. To specify an area of a specific size (e.g., 3 miles on each side), add something like "a 3-mile area (or square/box/…)." Phrase the whole thing in any way that seems natural to you.

To decrease the magnification, i.e., fit a larger area into the display, you can use words like "zoom out," "shrink," "less detail," etc. Additional information can be specify in a similar manner to the "zoom in" operation described above.

To show an area north/south/east/west etc. of the currently displayed area, you can use words like "pan," "shift," "move," etc. together with the desired direction (left/right/up/down should also work). You can also specify a distance for the shift, either in miles/kilometers or as a percentage of the screen.

Sometimes the computer does not recognize all of the words correctly, but it gets enough information so that it does exactly the right thing. As long as the result is what you intended, you can move on to the next task.

TASK LIST

You start with a map that displays the streets of Pittsburgh.

1. **Goal:** Get the map to locate Carnegie Mellon University and display it as an icon.

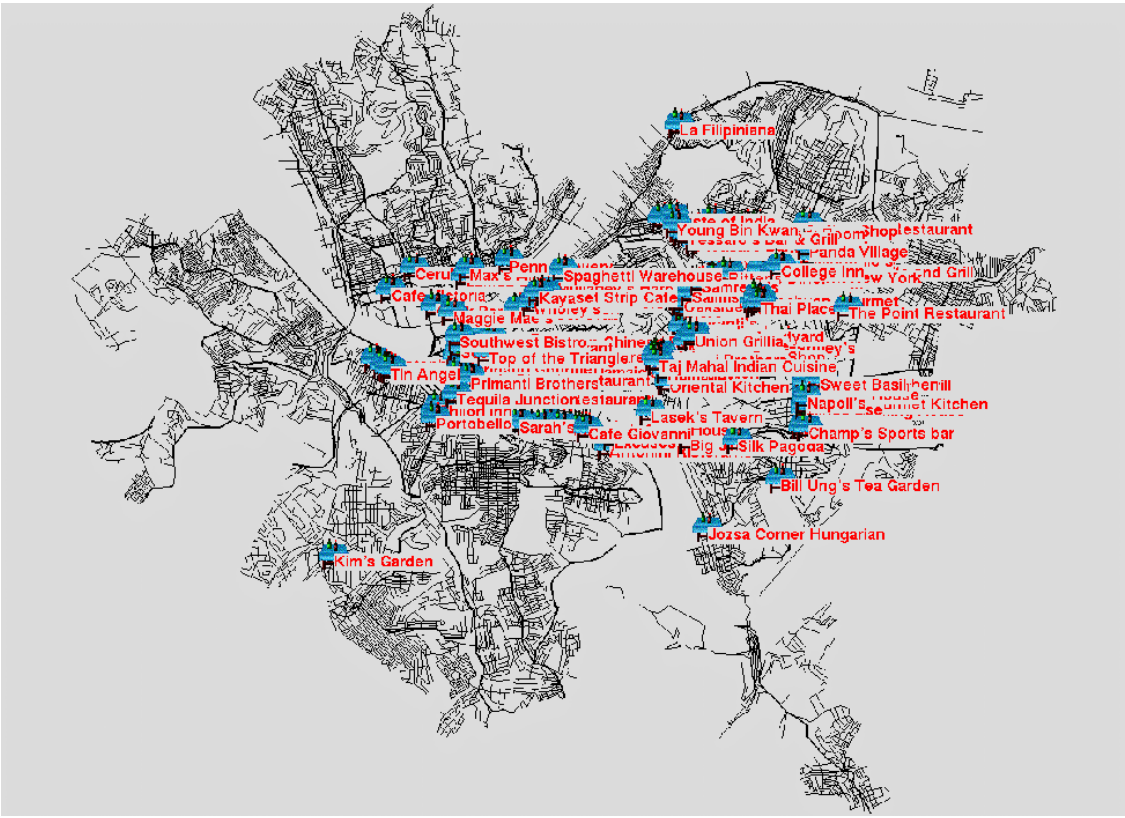
   Result: After you are done, the map display should look like this:

2. **Goal:** Get the map to display the way from Carnegie Mellon University to University of Pittsburgh and the distance you would have to traverse.

   Result: After you are done, the map display should look like this:

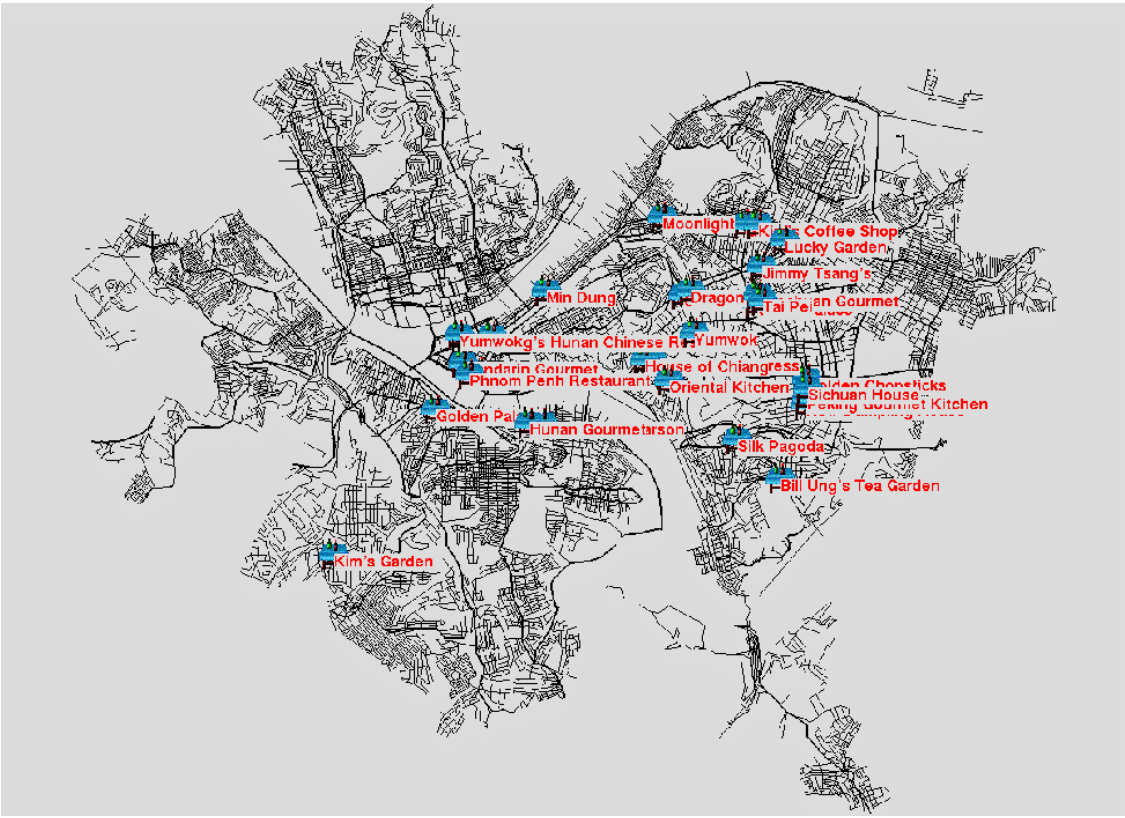
3. **Goal:** Get the map to display all the restaurants it knows about.

   Result: After you are done, the map display should look like this:

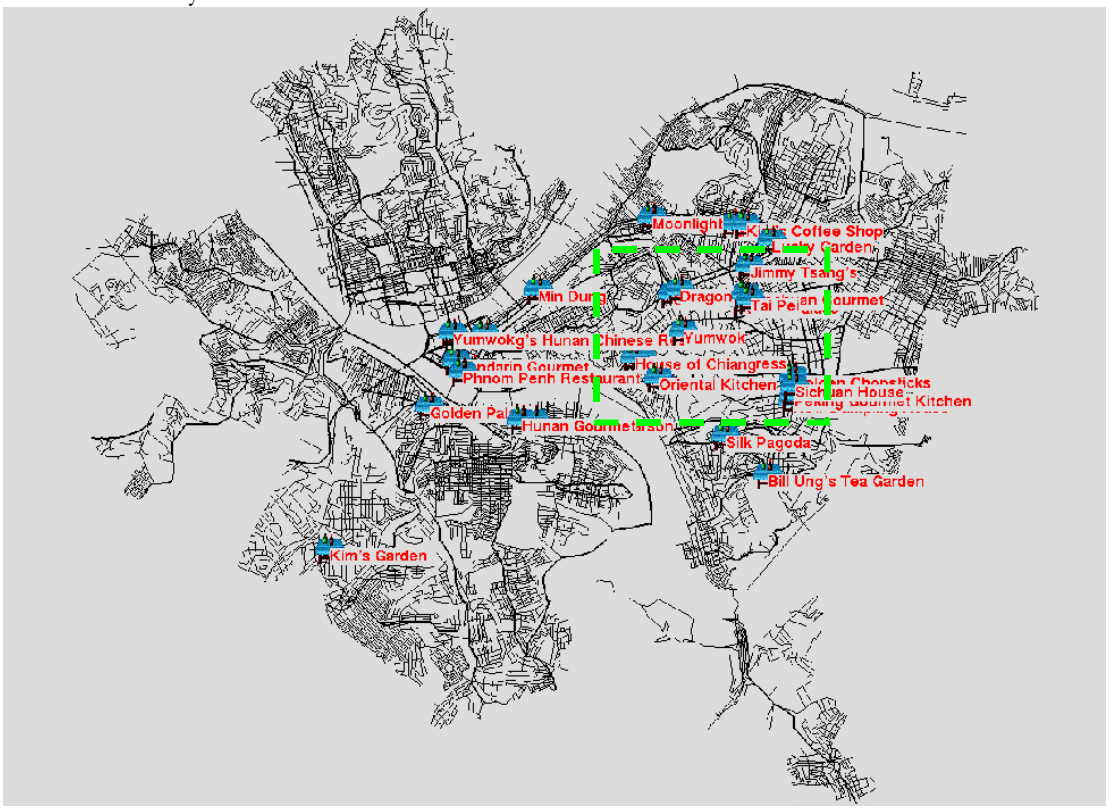


4. **Goal:** Get the map to display all the Chinese restaurants it knows about.

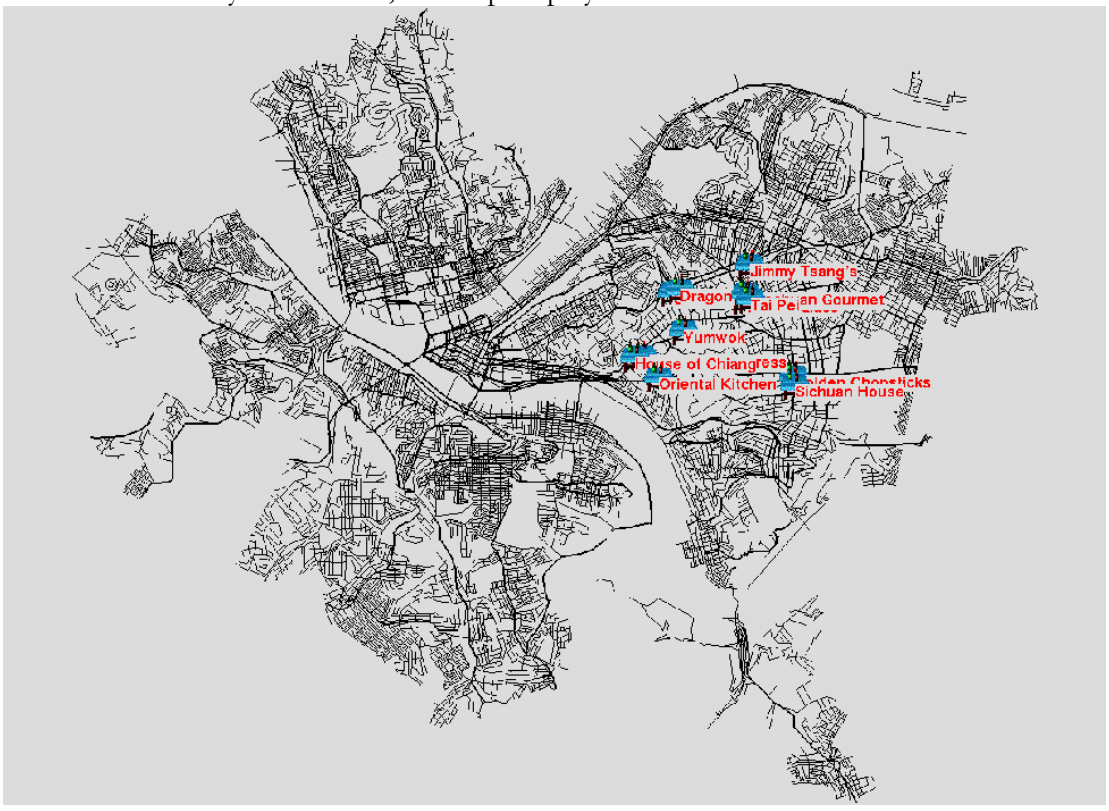
   Result: After you are done, the map display should look like this:



5. **Goal:** Get the map to display all the Chinese restaurants approximately within the area indicated by the dashed box:



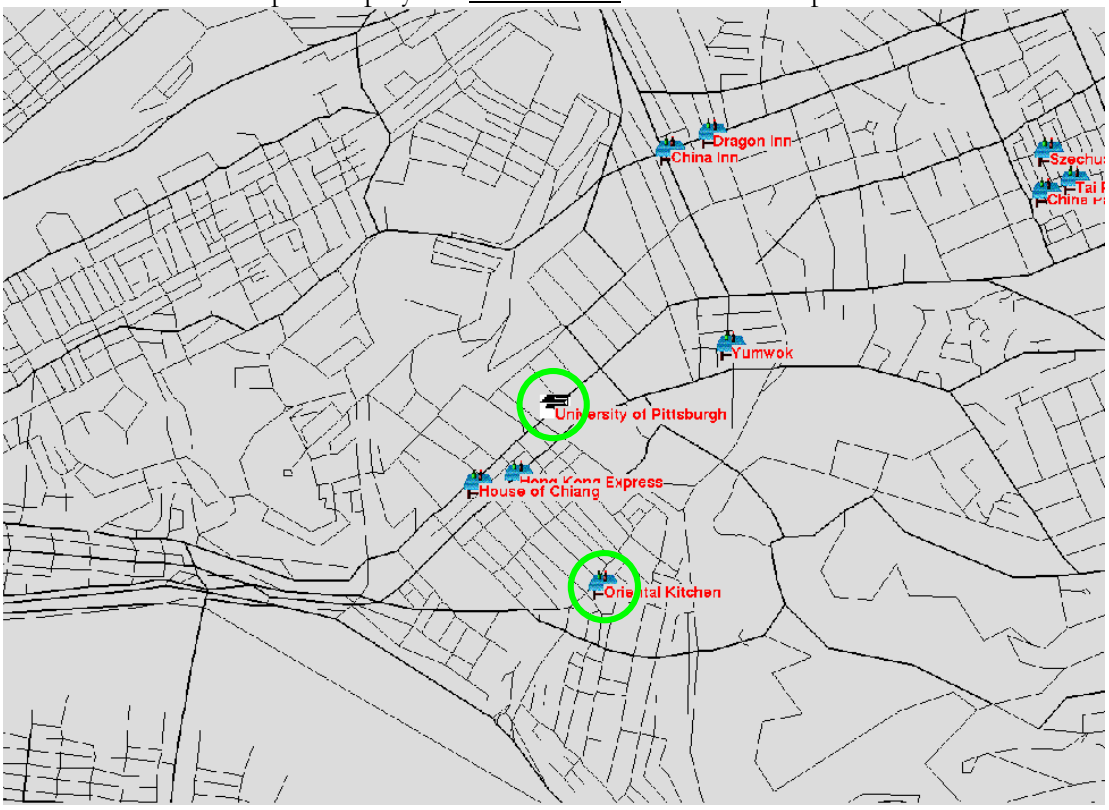
Result: After you are done, the map display should look like this:



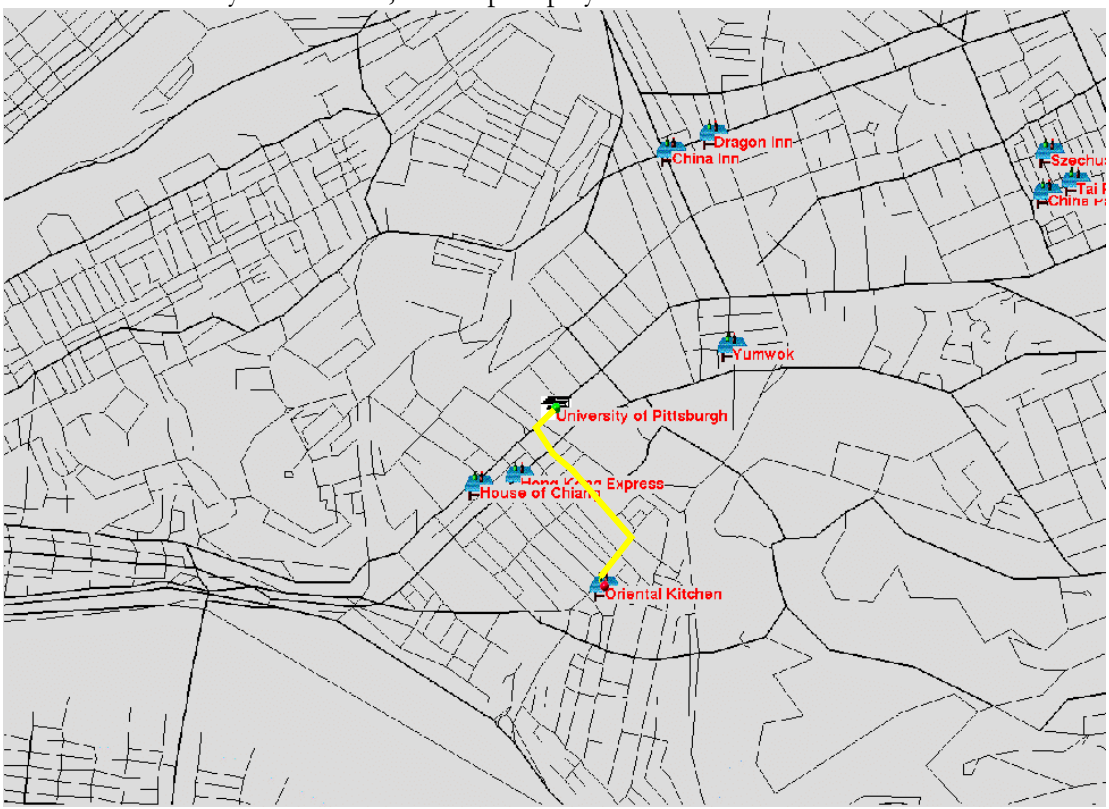
6. **Goal:** Get the map to display University of Pittsburgh in an area that measures 2 miles on each side.

   Result: After you are done, the map display should look like this:

7. **Goal:** Get the map to display the fastest route between the 2 places circled below:

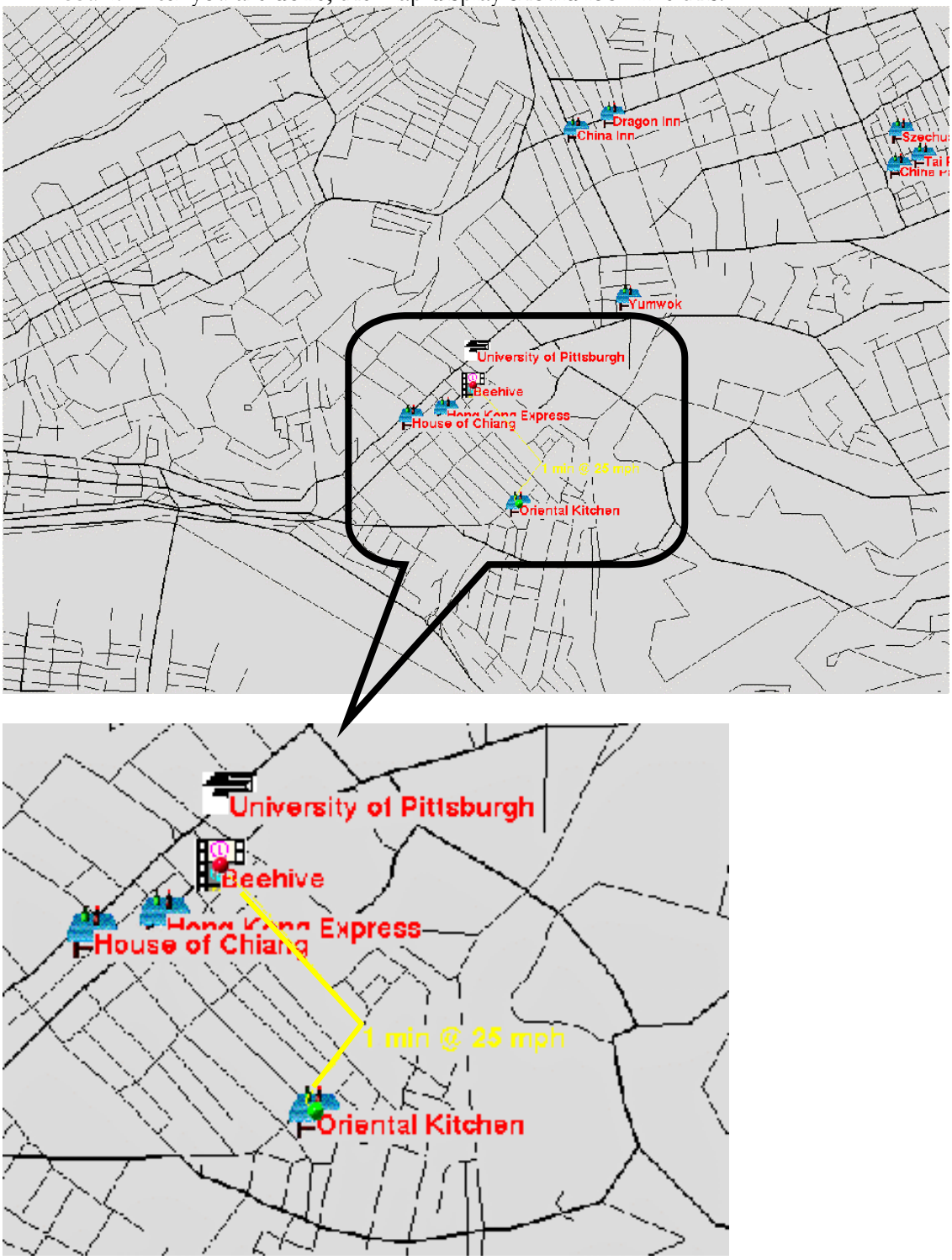


Result: After you are done, the map display should look like this:

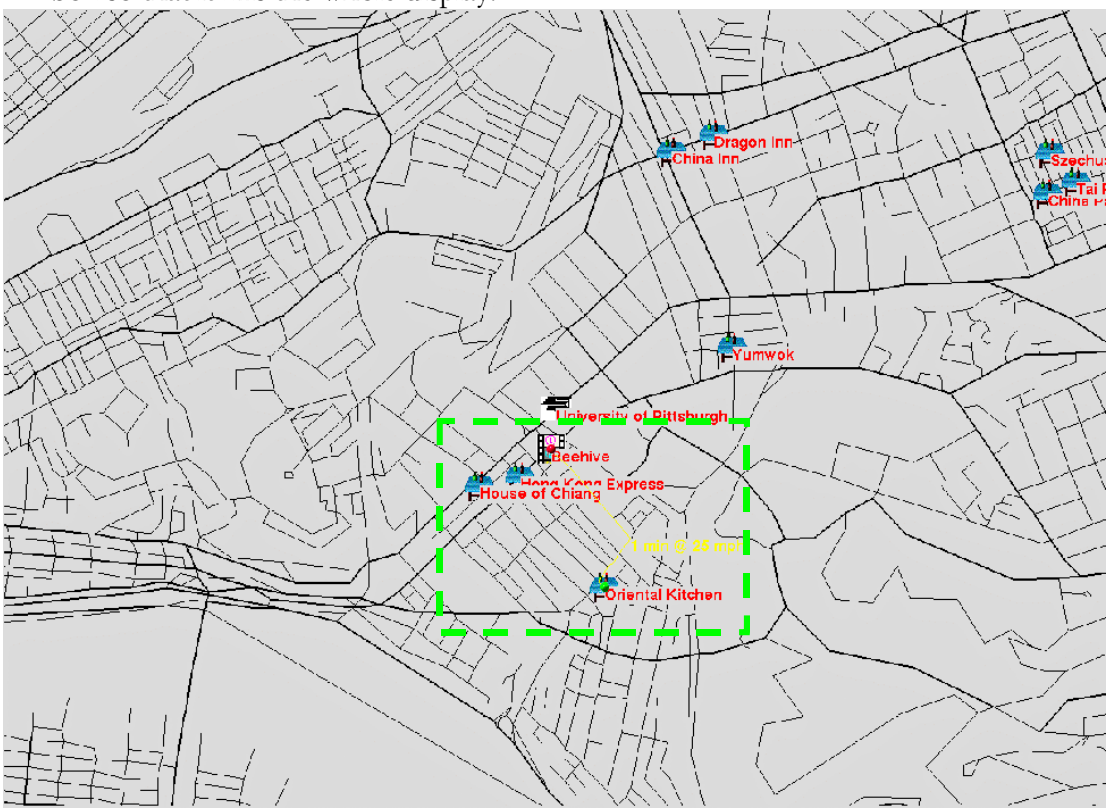


8. **Goal:** Get the map to display the way to the nearest movie theater from where you are (the Oriental Kitchen) and the time it would take you to get there.

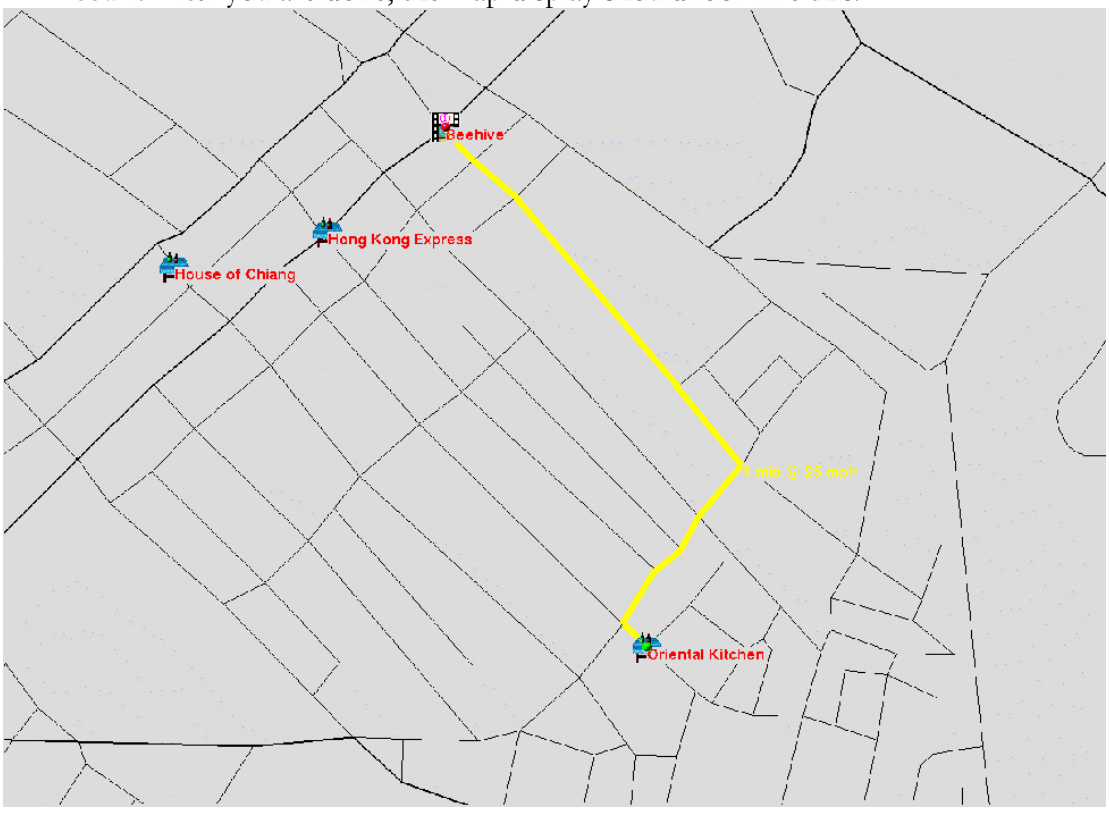
   Result: After you are done, the map display should look like this:



9. **Goal:** Get the map to <u>expand</u> (magnify) <u>the area</u> approximately indicated by the dashed box so that it fills the whole display:



Result: After you are done, the map display should look like this:



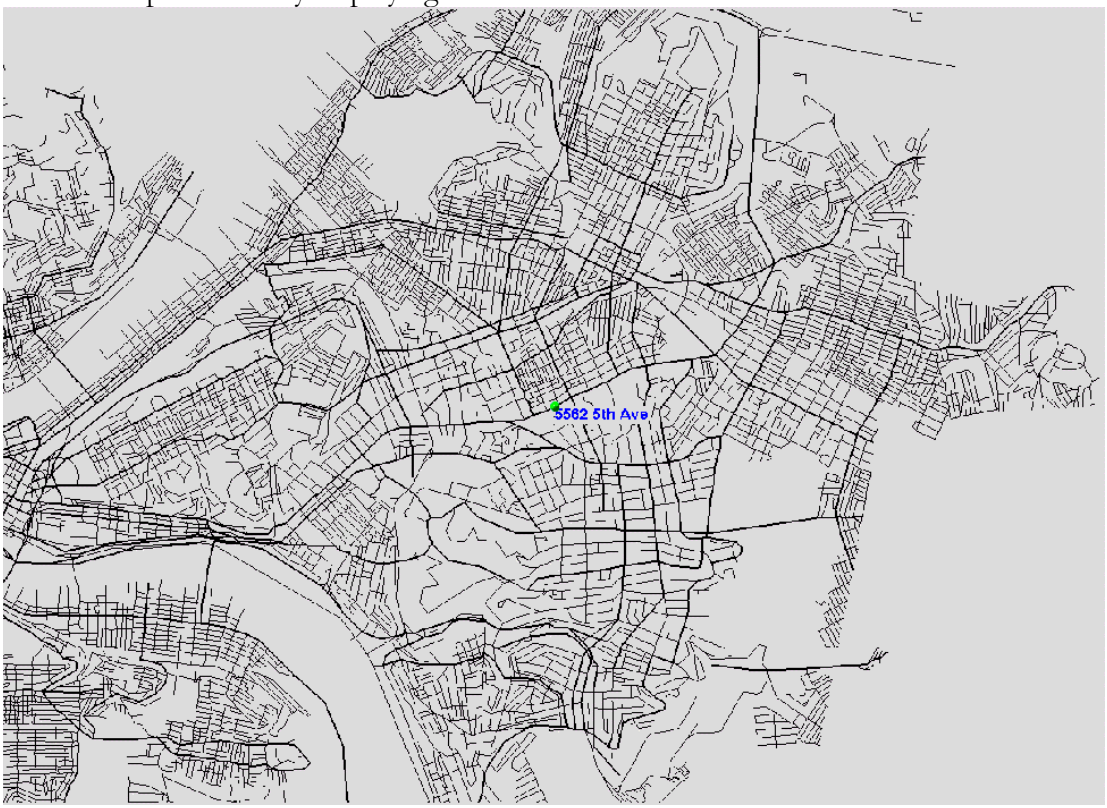
10. **Goal:** Get the map to locate and display 5562 Fifth Avenue.

    Result: After you are done, the map display should look like this:

11. **Goal:** Get the map to zoom out (shrink the view) to display an area 10 times the currently displayed area.

    Result: After you are done, the map display should look like this:

12. The map is currently displaying this:



Goal: Get the map to display this—3 miles to the left (west) of where you were:

THE END

QUESTIONNAIRE

User ID: _____

The first three questions refer to the way in which you could interact with the computer system.

1. You could speak to tell the computer what you wanted. How much did you like this style of interaction with the computer?

   _____ not at all
   _____ somewhat
   _____ quite well
   _____ very much
   _____ don't know (I didn't use it)

2. You could draw on the screen to tell the computer what you wanted. How much did you like this style of interaction with the computer?

   _____ not at all
   _____ somewhat
   _____ quite well
   _____ very much
   _____ don't know (I didn't use it)

3. You could BOTH speak AND draw on the screen to tell the computer what you wanted. How much did you like this style of interaction with the computer?

   _____ not at all
   _____ somewhat
   _____ quite well
   _____ very much
   _____ don't know (I didn't use these two modes in combination)

4. What did you find the most annoying about the computer program (even if you liked it in general)?

   _____ nothing (I was not annoyed at all)
   _____ it was too slow
   _____ it did not understand me well enough
   _____ others (please specify)

BIBLIOGRAPHY

[Alleva92] Alleva, F., Hon, H., Huang, X., Hwang, M., Rosenfeld, R., and Weide, R.
Applying SPHINX-II to the DARPA Wall Street Journal CSR Task.
In *DARPA Speech and Language Workshop*. Morgan Kaufmann (San Mateo, CA), 1992.

[Anderson77] Anderson, J.R.
Induction of Augmented Transition Networks.
Cognitive Science 1:125-157, 1977.

[Ando94] Ando, H., Kitahara, Y., and Hataoka, N.
Evaluation of Multimodal Interface Using Spoken Language and Pointing Gesture On Interior Design System.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, Vol. 2, 567-570. (Yokohama, Japan) September 1994.

[Arakawa78] Arakawa, H., Okada, K., and Masuda, J.
On-line Recognition of Handwritten Characters: Alphanumerics, Hiragana, Katakana, Kanji.
In Proceedings of *International Joint Conference on Pattern Recognition*, 810-812, 1978.

[Arnold96] Arnold, K. and Gosling, J.
The Java Programming Language.
Addison-Wesley (Reading, MA), 1996.

[Bahl89] Bahl, L.R., Brown, P.F., de Souze, P.V., and Mercer, R.L.
A Tree-Based Statistical Language Model for Natural Language Speech Recognition.
IEEE Transactions on Acoustics, Speech, and Signal Processing
ASSP-37(7):1001-1008, July 1989.

[Baluja94] Baluja, S. and Pomerleau, D.
Non-intrusive Gaze Tracking Using Artificial Neural Networks.
In *Advances in Neural Information Processing Systems (NIPS 6)*. Morgan Kaufmann (San Mateo, CA), 1994.

[Bellik97] Bellik, Y.
Media Integration in Multimodal Interfaces.
In Proceedings of *First Signal Processing Society Workshop on Multimedia Signal Processing*, 31-36. (Princeton, NJ) June 1997.

[Berthod79] Berthod, M. and Maroy, J.P.
Learning in Syntactic Recognition of Symbols Drawn on a Graphic Tablet.
Computer Graphics Image Processing 9:166-182, 1979.

[Beskow96] Beskow, J., Elenius, K., and McGlashan, S.
Olga — A Dialogue System with an Animated Talking Agent.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'96)*, Vol. 3, 1651-1654. (Philadelphia, PA) October 1996.

[Blachman68] Blachman, N.M.
The Amount of Information that Y Gives About X.
IEEE Transactions on Information Theory 14(1):27-31, 1968.

[Blattner96] Blattner, M.M. and Glinert, E.P.
Multimodal Integration.
IEEE Multimedia 3(4):14-24, 1996.

[Bohm92] Bohm, K., Hubner, W., and Vaananen, K.
GIVEN: Gesture Driven Interactions in Virtual Environments — A Toolkit Approach to 3D Interactions.
In Proceedings of *International Conference on Interface to Real and Virtual Worlds (Informatique'92)*. (Montpellier, France) March 1992.

[Bolt80] Bolt, R.A.
Put-That-There: Voice and Gesture at the Graphics Interface.
ACM Computer Graphics 14(3):262-270, 1980.

[Bonafonte96] Bonafonte, A., Marino, J.B., and Nogueiras, A.
Sethos: The UPC Speech Understanding System.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'96)*, Vol. 4, 2151-2154. (Philadelphia, PA) October 1996.

[Booch93] Booch, G.
Object-Oriented Analysis and Design with Applications, 2nd edition.
Benjamin/Cummings (Redwood City, CA), 1993.

[Boros96] Boros, M., Eckert, W., Gallwitz, F., Gorz, G., Hanrieder, G., Niemann, H.
Towards Understanding Spontaneous Speech: Word Accuracy vs. Concept Accuracy.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'96)*, Vol. 2, 1009-1012. (Philadelphia, PA) October 1996.

[Bourlard88] Bourlard, H. and Wellekens, C.J.
Links Between Markov Models and Multilayer Perceptrons.
Technical Report Manuscript M-263, Phillips Research Laboratory (Brussels, Belgium), 1988.

[Bregler93] Bregler, C., Hild, H., Manke, S., and Waibel, A.
Improving Connected Letter Recognition by Lipreading.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'93)*. (Minneapolis, MN) April 1993.

[Brown94] Brown, M.K. and Buntschuh, B.M.
A Context-Free Grammar Compiler for Speech Understanding Systems.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, Vol. 1, 21-24. (Yokohama, Japan) September 1994.

[Burr88] Burr, D.J.
Experiments on Neural Net Recognition of Spoken and Written Text.
IEEE Transactions on Acoustics, Speech, and Signal Processing 36:1162-1168, 1988.

[Burton76] Burton, R.R.
Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems.
BBN Report 3453, Bolt Beranek & Newman Inc., December 1976.

[Buxton85] Buxton, W.A.S., Sniderman, R., Reeves, W., Patel, S., and Baecker, R.
The Evolution of the SSSP Score-Editing Tools.
In *Foundations of Computer Music*, Roads, C. and Strawn, J. (Eds.), 387-392.
MIT Press (Cambridge, MA), 1985.

[Carbonell84] Carbonell, J.G. and Hayes, P.J.
Recovery Strategies for Parsing Extra-grammatical Language.
Technical Report CMU-CS-84-107, Carnegie Mellon University (Pittsburgh, PA), February 1984.

[Chen96] Chen, S., Kazi, Z., Beitler, M., Salganicoff, M., Chester, D., and Foulds, R.
Gesture-Speech Based HMI for a Rehabilitation Robot.
In Proceedings of *South East Conference*, 29-36. (Tampa, FL) 1996.

[Cheyer95] Cheyer, A. and Julia, L.
Multimodal Maps: An Agent-Based Approach.
In Proceedings of *International Conference on Cooperative Multimodal Communication (CMC'95)*. (Eindhoven, Netherlands) May 1995.

[Cheyer97] Cheyer, A., and Julia, L.
MVIEWS: Multimodal Tools for the Video Analyst.
In Proceedings of *International Conference on Intelligent User Interfaces*, 55-62. (San Francisco, CA) 1997.

[Chu97] Chu, C-C.P., Dani, T.H., and Gadh, R.
Multimodal Interface for a Virtual Reality Based Computer Aided Design System.
In Proceedings of *IEEE International Conference on Robotics and Automation*, Vol. 2, 1329-1334. (Albuquerque, NM) April 1997.

[Cohen89] Cohen, P.R., Dalrymple, M., Moran, D.B., Periera, F.C.N., Sullivan, J.W., Gargan, R.A., Jr., Schlossberg, J.L., and Tyler, S.W.
Synergistic Use of Direct Manipulation and Natural Language.
In Proceedings of *ACM Conference on Human Factors in Computing Systems (CHI'89)*, 227-234. (Austin, TX) April 1989.

[Cohen92] Cohen, P.R.
The Role of Natural Language in a Multimodal Interface.
In Proceedings of *Symposium on User Interface Software and Technology (UIST'92)*, 143-149. (Monterey, CA) November 1992.

[Coleman69] Coleman, M.L.
Text Editing on a Graphic Display Device Using Hand-Drawn Proofreader's Symbols.
In *Pertinent Concepts in Computer Graphics*, Proceedings of *University of Illinois Conference on Computer Graphics*, Faiman, M. and Nievergelt, J. (Eds.), 283-290. 1969.

[Coutaz95] Coutaz, J., Nigay, L., Salber, D., Blanford, A., May, J., and Young, R.
Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties.
In Proceedings of *International Conference on Human-Computer Interaction (INTERACT'95)*, 115-20. (Lillehammer, Norway) June 1995.

[Dillman78] Dillman, D.A.
Mail and Telephone Surveys: The Total Design Method.
John Wiley & Sons (New York), 1978.

[Dowding93] Dowding, J., Gawron, J.M., Appelt, D., Bear, J., Cherny, L., Moore, R., and Moran, D.
GEMINI: A Natural Language System for Spoken Language Understanding.
In Proceedings of *Annual Meeting of the Association for Computational Linguistics*, 54-61. (Columbus, OH) June 1993.

[Eastwood97] Eastwood, B., Jennings, A., and Harvey, A.
Neural Network Based Segmentation of Handwritten Words.
In Proceedings of *International Conference on Image Processing and its Applications*, Vol. 2, 750-755. (Dublin, Ireland) July 1997.

[Edelman90] Edelman, S., Flash, T., and Ullman, S.
Reading Cursive Handwriting by Alignment of Letter Prototypes.
International Journal of Computer Vision 5(3):303-331, 1990.

[Eglowstein90] Eglowstein, H.
Reach Out and Touch Your Data.
Byte 15(7):283-290, July 1990.

[Erbach92] Erbach, G.
Tools for Grammar Engineering.
In Proceedings of *3rd Conference on Applied Natural Language Processing*, 243-244. (Trento, Italy) March 1992.

[Erman80] Erman, L.D. and Lesser, V.R.
The Hearsay-II Speech Understanding System: A Tutorial.
In *Trends in Speech Recognition*, 361-381. Speech Science Publications (Apple Valley, MN), 1980.

[Faure93] Faure, C. and Julia, L.
Interaction Homme-Machine par la Parole et le Geste pour l'Edition de Documents: TAPAGE.
(Speech and Gesture for Man-Machine Interaction Applied to Document Editing: TAPAGE.)
In Proceedings of *International Conference on Interface to Real and Virtual Worlds (Informatique'93)*, 171-180. (Montpellier, France) March 1993.

[Favata92] Favata, J.T. and Srihari, S.N.
Off-line Recognition of Handwritten Cursive Words.
In Proceedings of *SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, 224-234. (San Jose, CA) February 1992.

[Fels90] Fels, S.S. and Hinton, G.E.
Building Adaptive Interfaces with Neural Networks: The Glove-Talk Pilot Study.
Technical Report CRG-TR-90-1, University of Toronto (Toronto, Canada), 1990.

[Finke97] Finke, M., Geutner, P., Hild, H., Kemp, T., Ries, K., and Westphal, M.
The Karlsruhe-Verbmobil Speech Recognition Engine.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*. (Munich, Germany) 1997.

[Firth91] Firth, C. and Thomas, R.C.
The Use of Command Language Grammar in Design Tool.
International Journal of Man-Machine Studies 34(4):479-496, April 1991.

[Fowler97] Fowler, M. and Scott, K.
UML Distilled: Applying the Standard Object Modeling Language.
Addison-Wesley (Reading, MA), 1997.

[Fu81] Fu, K.S.
Syntactic Pattern Recognition and Applications.
Prentice Hall, 1981.

[Gader94] Gader, P.D. and Keller, J.M.
Applications of Fuzzy Sets Theory to Handwriting Recognition.
In Proceedings of *IEEE International Fuzzy Systems Conference*, Vol. 2, 910-917. (Orlando, FL) June 1994.

[Gamma95] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1995.

[Gomoll90] Gomoll, K.
Some Techniques for Observing Users.
In *The Art of Human-Computer Interface Design*, Laurel, B. (Ed.), 85-90.
Addison-Wesley (Reading, MA), 1990.

[Gorin91] Gorin, A.L., Levinson, S., Gertner, A., and Goldman, E.
Adaptive Acquisition of Language.
Computer Speech and Language 5(2):101-132, April 1991.

[Gorin94] Gorin, A.L., Levinson, S.E., and Sankar, A.
An Experiment in Spoken Language Acquisition.
IEEE Transactions on Speech and Audio Processing 2(1)(Part II):224-240, January 1994.

[Gorin95] Gorin, A.L.
On Automated Language Acquisition.
Journal of the Acoustics Society of America 97(6):3441-3461, June 1995.

[Gorin96] Gorin, A.L.
Processing of Semantic Information in Fluently Spoken Language.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'96)*. (Philadelphia, PA) October 1996.

[Govindaraju97] Govindaraju, V., Kim, G., Srihari, S.N.
Paradigms in Handwriting Recognition.
In Proceedings of *IEEE International Conference on Systems, Man, and Cybernetics (ICSMC'97)*, 1498-1503. (Orlando, FL) October 1997.

[Gray84] Gray, R.M.
Vector Quantization.
IEEE ASSP Magazine 1(2):4-29, April 1984.

[Guyon91] Guyon, I., Albrecht, P., Le Cun, Y., Denker, W., and Hubbard, W.
Design of a Neural Network Character Recognizer for a Touch Terminal.
Pattern Recognition 24(2):105-119, 1991.

[Haffner92] Haffner, P. and Waibel, A.
Multi-State Time Delay Neural Networks for Continuous Speech Recognition.
In *Advances in Neural Network Information Processing Systems (NIPS 4)*, 135-142. Morgan Kaufmann (San Mateo, CA), 1992.

[Hand82] Hand, D.J.
Kernel Discriminant Analysis.
Research Studies Press (A Division of John Wiley and Sons, New York), 1982.

[Hauptmann89] Hauptmann, A.
Speech and Gestures for Graphic Image Manipulation.
In Proceedings of *ACM Conference on Human Factors in Computing Systems (CHI'89)*, 241-245. (Austin, TX) April 1989.

[Hayes81] Hayes, P.J. and Mouradian, G.V.
Flexible Parsing.
American Journal of Computational Linguistics 7.4:232-242, 1981.

[Hemphill89] Hemphill, C. and Picone, J.
Robust Speech Recognition in a Unification Grammar Framework.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'89)*. May 1989.

[Henry90] Henry, T.R., Hudson, S.E., and Newell, G.L.
Integrating Gesture and Snapping into a User Interface Toolkit.
In Proceedings of *Symposium on User Interface Software and Technology (UIST'90)*, 112-122. October 1990.

[Higgins85] Higgins, A.L. and Wohlford, R.E.
Keyword Recognition Using Template Concatenation.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'85)*, 1233-1236. 1985.

[Hollan88] Hollan, J., Rich, E., Hill, W., Wroblewski, D., Wilner, W., Wittenberg, K., Grudin, J., and Members of the Human Interface Laboratory.
An Introduction to HITS: Human Interface Tool Suite.
Technical Report ACA-HI-406-88, Microelectronics and Computer Technology Corporation (Austin, TX), 1988.

[Huang88] Huang, W.M. and Lippmann, R.P.
Neural Net and Traditional Classifier.
In *Neural Information Processing Systems*, Anderson, D. (Ed.), 387-396. Morgan Kaufmann (San Mateo, CA) 1988.

[Huang93] Huang, X., Alleva, F., Hon, H.W., Hwang, M.Y, Lee, K., and Rosenfeld, R.
The SPHINX-II Speech Recognition System: An Overview.
Computer Speech and Language 2:137-148, 1993.

[Jackson88] Jackson, P.L.
The Theoretical Minimal Unit for Visual Speech Perception: Visemes and Coarticulation.
The Volta Review 90(5):99-115, 1988.

[Jacob91] Jacob, R.J.K.
The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At Is What You Get.
ACM Transactions on Information Systems 9(3):152-169, April 1991.

[Jacobson92] Jacobson, I.
Object-Oriented Software Engineering: A Use Case Driven Approach.
Addison-Wesley (Reading, MA), 1992.

[Jelinek90] Jelinek, F.
Self-Organized Language Modeling for Speech Recognition.
1990. Reprinted in *Readings in Speech Recognition*, Waibel, A. and Lee, K.F. (Eds.), Morgan Kaufmann (San Mateo, CA), 1990.

[Jelinek98] Jelinek, F.
Statistical Methods for Speech Recognition.
MIT Press (Cambridge, MA), January 1998.

[Jing97] Jing, X., Yang, J., Vo, M.T., and Waibel, A.
Java Frontend for Web-Based Multimodal Human-Computer Interaction.
In Proceedings of *Workshop on Perceptual User Interface*, 78-81. (Banff, Canada) 1997.

[Johnston97] Johnston, M., Cohen, P.R., McGee, D., Oviatt, S.L., Pittman, J.A., and Smith, I.
Unification-based Multimodal Integration.
In Proceedings of *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, 281-288. 1997.

[Julia95] Julia, L. and Faure, C.
Pattern Recognition and Beautification for a Pen Based Interface.
In Proceedings of *International Conference on Document Analysis and Recognition (ICDAR'95)*, 58-63. (Montreal, Canada) August 1995.

[Kamio94] Kamio, H., Koorita, M., Matsu'ura, H., Tamura, M., and Nitta, T.
A UI Design Support Tool for Multimodal Spoken Dialogue System.
In Proceedings of *International Conference of Spoken Language Processing (ICSLP'94)*, Vol. 3, 1283-1286. (Yokohama, Japan) September 1994.

[Kawahara97] Kawahara, T., Chin-Hui Lee, and Biing-Hwang Juang.
Combining Key-phrase Detection and Subword-Based Verification for Flexible Speech Understanding.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, Vol. 2, 1159-1162. (Munich, Germany) April 1997.

[KeHan95] Ke Han and Sethi, I.K.
Off-line Cursive Handwriting Segmentation.
In Proceedings of *International Conference on Document Analysis and Recognition (ICDAR'95)*, Vol. 2, 894-897. (Montreal, Canada) August 1995.

[Kernighan88] Kernighan, B. and Ritchie, D.
The C Programming Language.
Prentice Hall (Englewood Cliffs, NJ), 1988.

[Kim97] Kim, G. and Govindaraju, V.
A Lexicon Driven Approach to Handwritten Word Recognition for Real-time Applications.
IEEE Transactions on Pattern Analysis and Machine Intelligence 19(4):366-379, April 1997.

[Knuth73] Knuth, D.
The Art of Computer Programming: Fundamental Algorithms, Vol. 1.
Addison Wesley (Reading, MA), 1973.

[Kohonen88] Kohonen, T., Barna, G., and Chrisley, R.
Statistical Pattern Recognition with Neural Networks: Benchmarking Studies.
In Proceedings of *International Conference on Neural Networks (ICNN'88)*, Vol. I, 61-68. (San Diego, CA) July 1988.

[Kolzay71] Kolzay, D.
Feature Extraction in an Optical Character Recognition Machine.
IEEE Transactions on Computers 20:1063-1067, 1971.

[Koons93] Koons, D.B., Sparrell, C.J., and Thorisson, K.R.
Integrating Simultaneous Input From Speech, Gaze, and Hand Gestures.
In *Intelligent Multimedia Interfaces*, Maybury, M.T. (Ed.), 257-276. MIT Press (Cambridge, MA), 1993.

[Kuch95] Kuch, J.J. and Huang, T.S.
Vision Based Hand Modeling and Tracking.
In Proceedings of *International Conference on Computer Vision*. (Cambridge, MA) June 1995.

[Kullberg95] Kullberg, R.L.
Mark Your Calendar! Learning Personalized Annotation from Integrated Sketch and Speech.
In Proceedings of *ACM Conference on Human Factors in Computing Systems (CHI'95)*, 302-303. (Denver, CO) May 1995.

[Kurtenbach91] Kurtenbach, G. and Buxton, W.A.S.
GEdit: A Test Bed for Editing by Continuous Gestures.
SIGCHI Bulletin, 22-26, 1991.

[Landay93] Landay, J.A. and Myers, B.A.
Extending an Existing User Interface Toolkit to Support Gesture Recognition
In Adjunct Proceedings of *INTERCHI'93*, 91-92. (Amsterdam, Netherlands) April 1993.

[Lavie93] Lavie, A. and Tomita, M.
GLR* — An Efficient Noise Skipping Parsing Algorithm for Context Free Grammars.
In Proceedings of *International Workshop on Parsing Technologies (IWPT'93)*. (Tilburg, Netherlands) August 1993.

[Lavie97] Lavie, A., Waibel, A., Levin, L., Finke, M., Gates, D., Gavalda, M., Zeppenfeld, T., and Zhan, P.
JANUS-III: Speech-to-Speech Translation in Multiple Languages.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*. (Munich, Germany) 1997.

[Lecolinet91] Lecolinet, E. and Crettez, J.
A Grapheme-Based Segmentation Technique for Cursive Script Recognition.
In Proceedings of *International Conference on Document Analysis and Recognition (ICDAR'91)*, 740-748. (Saint-Malo, France) September 1991.

[Lee90] Lee, J. and Zeevat, H.
Integrating Natural Language and Graphics in Dialogue.
In Proceedings of *International Conference on Human-Computer Interaction (INTERACT'90)*, short papers, 479-484. 1990.

[Leopold97] Leopold, J.L. and Ambler, A.L.
Keyboardless Visual Programming Using Voice, Handwriting, and Gesture.
In Proceedings of *IEEE Symposium on Visual Languages*, 28-35. (Isle of Capri, Italy) September 1997.

[Lippmann87] Lippmann, R.P. and Gold, B.
Neural Classifiers Useful for Speech Recognition.
In Proceedings of *International Conference on Neural Networks (ICNN'87)*, IV-417. 1987.

[Lipscomb91] Lipscomb, J.S.
A Trainable Gesture Recognizer.
Pattern Recognition, 1991.

[Loken-Kim94] Loken-Kim, K., Yato, F., Fais, L., Morimoto, T., and Kurematsu, A.
Linguistic and Paralinguistic Differences Between Multimodal and Telephone-Only Dialogues.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, 571-574. (Yokohama, Japan) September 1994.

[Lowerre80] Lowerre, B.T. and Reddy, D.R.
The HARPY Speech Understanding System.
In *Trends in Speech Recognition*. Speech Science Publications (Apple Valley, MN), 1980.

[Madhvanath96] Madhvanath, S.
The Holistic Paradigm in Handwritten Word Recognition and Its Application to Large and Dynamic Lexicon Scenarios.
PhD Thesis, Dept. of Computer Science, State University of New York at Buffalo (Buffalo, NY), 1996.

[Manke95] Manke, S., Finke, M., and Waibel, A.
NPen++: A Writer Independent, Large Vocabulary On-line Cursive Handwriting Recognition System.
In Proceedings of *International Conference on Document Analysis and Recognition (ICDAR'95)*. (Montreal, Canada) August 1995.

[Manke98] Manke, S.
On-line Erkennung kursiver Handschrift bei großen Vokabularen.
(On-line Large Vocabulary Cursive Handwriting Recognition.)
Ph.D. Thesis, Computer Science Department, University of Karlsruhe (Karlsruhe, Germany), February 1998.

[Markel76] Markel, J.D., Gray, A.H.
Linear Prediction of Speech.
Springer-Verlag (Berlin, Germany), 1976.

[Martin97] Martin, J-C.
Toward "Intelligent" Cooperation Between Modalities: The Example of a System Enabling Multimodal Interaction with a Map.
In Proceedings of *International Joint Conference on Artificial Intelligence (IJCAI'97) Workshop on "Intelligent Multimodal Systems."* (Nagoya, Japan) August 1997.

[Matsu'ura94] Matsu'ura, H., Masai, Y., Iwasaki, J., Tanaka, S., Kamio, H., and Nitta, T.
A Multimodal, Keyword-Based Spoken Dialogue System — MultiksDial.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'94)*, Vol. 2, 33-36. (Adelaide, Australia) April 1994.

[McClelland89] McClelland, J.L., St. John, M., and Taraban, R.
Sentence Comprehension: A Parallel Distributed Processing Approach.
Language and Cognitive Processes 314:287-335, 1989.

[McNair94] McNair, A. and Waibel, A.
Improving Recognizer Acceptance Through Robust, Natural Speech Repair.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, 1299-1302. (Yokohama, Japan) September 1994.

[Medress78] Medress, M.F. et al.
An Automatic Word Spotting System for Conversational Speech.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'78)*, 712-713. 1978.

[Menier94] Menier, G., Lorette, G., and Gentric, P.
A Genetic Algorithm for On-line Cursive Handwriting Recognition.
In Proceedings of *International Conference on Pattern Recognition*, Vol. 2, 522-525. (Jerusalem, Israel) October 1994.

[Miller93] Miller, L.G. and Gorin, A.L.
Structured Networks for Adaptive Language Acquisition.
International Journal of Pattern Recognition and Artificial Intelligence 7(4):873-898, 1993.

[Milota95] Milota, A.D. and Blattner, M.M.
Multimodal Interfaces with Voice and Gesture Input.
In Proceedings of *IEEE International Conference on Systems, Man, and Cybernetics (ICSMC'95)*, Vol. 3, 2760-2765. (Vancouver, Canada) October 1995.

[Minsky84] Minsky, M.R.
Manipulating Simulated Objects with Real-World Gestures Using a Force and Position Sensitive Screen.
Computer Graphics 18(3):195-203, July 1984.

[Mitchell97] Mitchell, T.M.
Machine Learning.
WCB/McGraw-Hill, 1997.

[Moran97] Moran, D.B., Cheyer, A.J., Julia, L.E., Martin, D.L, and Park, S.
Multimodal User Interfaces in the Open Agent Architecture.
In Proceedings of *International Conference on Intelligent User Interfaces (IUI'97)*, 61-68. (Orlando, FL) January 1997.

[Murase88] Murase, H. and Wakahara, T.
Online Hand-Sketched Figure Recognition.
Pattern Recognition 19(2):147-160, 1988.

[Myers80] Myers, C.S., Rabiner, L.R., and Rosenberg, A.E.
An Investigation of the Use of Dynamic Time Warping for Word Spotting and Connected Word Recognition.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'80)*, 173-177. (Denver, CO) April 1980.

[Myers90] Myers, B.A., Giuse, D.A., Dannenberg, R.B., Zanden, B.V., Kosbie, D.S., Pervin, E., Mickish, A., and Marchal, P.
Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces.
Computer 23(11):71-85, 1990.

[Myers97] Myers, B.A., McDaniel, R.G., Miller, R.C., Ferrency, A.S., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A., and Doane, P.
The Amulet Environment: New Models for Effective User Interface Software Development.
IEEE Transactions on Software Engineering 23(6):347-365, June 1997.

[Nagai94] Nagai, A., Ishikawa, Y., and Nakajima, K.
A Semantic Interpretation Based on Detecting Concepts for Spontaneous Speech Understanding.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, Vol. 1, 95-98. (Yokohama, Japan) September 1994.

[Nakagawa94] Nakagawa, S. and Zhang, J.X.
An Input Interface with Speech and Touch Screen.
Transactions of the Institute of Electrical Engineers of Japan
114-C(10):1009-1017, October 1994.

[Neal91] Neal, J.G. and Shapiro, S.C.
Intelligent Multimedia Interface Technology.
In *Intelligent User Interfaces*, Sullivan, J.R. and Tyler, S.W. (Eds.), 12-67. ACM Press/Addison-Wesley (Reading, MA), 1991.

[Newell71] Newell, A. et al.
Speech-Understanding Systems: Final Report of a Study Group.
Computer Science Department, Carnegie Mellon University (Pittsburgh, PA), May 1971.

[Newman79] Newman, W.M. and Sproull, R.F.
Principles of Interactive Computer Graphics.
McGraw-Hill, 1979.

[Ney84] Ney, H.
The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition.
IEEE Transactions on Acoustics, Speech, and Signal Processing 32(2):263-271, 1984.

[Ney87] Ney, H.
Dynamic Programming Speech Recognition Using a Context-Free Grammar.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'87)*, 3.2.1-3.2.4. April 1987.

[Nigay93] Nigay, L. and Coutaz, J.
A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion.
In Proceedings of *INTERCHI'93*, 172-178. (Amsterdam, Netherlands) April 1993.

[Nigay95] Nigay, L. and Coutaz, J.
A Generic Platform for Addressing the Multimodal Challenge.
In Proceedings of *ACM Conference on Human Factors in Computing Systems (CHI'95)*, 98-105. (Denver, CO) May 1995.

[Nishimoto94] Nishimoto, T., Shida, N., Kobayashi, T., and Shirai, K.
Multimodal Drawing Tool Using Speech, Mouse and Keyboard.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, Vol. 3, 1287-1290. (Yokohama, Japan) September 1994.

[Olivieri95] Olivieri, P., Gips, J., and McHugh, J.
EagleEyes: Eye Controlled Multimedia.
In Proceedings of *ACM Multimedia'95*, 537-538. (San Francisco, CA) 1995.

[Ousterhout94] Ousterhout, J.
Tcl and the Tk Toolkit.
Addison-Wesley (Reading, MA), 1994.

[Oviatt91] Oviatt, S.L., and Cohen, P.R.
The Contributing Influence of Speech and Interaction on Human Discourse Patterns.
In *Intelligent User Interfaces*, Sullivan, J.R. and Tyler, S.W. (Eds.), 69-83. ACM Press/Addison-Wesley (Reading, MA), 1991.

[Oviatt92] Oviatt, S.L., Cohen, P.R., Fong, M., and Frank, M.
A Rapid Semi-Automatic Simulation Technique for Investigating Interactive Speech and Handwriting.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'92)*, Vol. 2, 1351-1354. 1992.

[Oviatt94a] Oviatt, S.L. and Olsen, E.
Integration Themes in Multimodal Human-Computer Interaction.
In Proceedings of *International Conference on Spoken Language Processing (ICSLP'94)*, Vol. 2, 551-554. (Yokohama, Japan) September 1994.

[Oviatt94b] Oviatt, S.L., Cohen, P.R., and Wang, M.
Toward Interface Design for Human Language Technology: Modality and Structure as Determinants of Linguistic Complexity.
Speech Communication (Netherlands) 15(3-4):283-300, December 1994.

[Oviatt96] Oviatt, S.L.
Multimodal Interfaces for Dynamic Interactive Maps.
In Proceedings of *ACM Conference on Human Factors in Computing Systems (CHI'96)*, 95-102, 1996.

[Oviatt97a] Oviatt, S.L., DeAngeli, A., and Kuhn, K.
Integration and Synchronization of Input Modes During Multimodal Human-Computer Interaction.
In Proceedings of *ACM Conference on Human Factors in Computing Systems (CHI'97)*, 415-422. (Atlanta, GA) March 1997.

[Oviatt97b] Oviatt, S.L.
Multimodal Interactive Maps: Designing for Human Performance.
Human-Computer Interaction 12(1-2):93-129, 1997.

[Rabiner78] Rabiner, L.R., Rosenberg, A.E., and Levinson, S.E.
Considerations in Dynamic Time-Warping Algorithms for Discrete Word Recognition.
IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-26, October 1978.

[Rabiner89] Rabiner, L.R.
A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.
In *Proceedings of the IEEE*, 1989.

[Rainer97] Rainer, S., Yang, J., and Waibel, A.
A Model-Based Gaze-Tracking System.
International Journal of Artificial Intelligence 6(2):193-209, 1997.

[Reddy73] Reddy, D.R., Erman, L.D., Fennell, R.D., and Neely, R.B.
The Hearsay Speech Understanding System: An Example of the Recognition Process.
In Proceedings of *International Joint Conference on Artificial Intelligence (IJCAI'73)*, 185-193. (Stanford, CA) 1973.

[Rehg93] Regh, J.M. and Kanade, T.
Digiteyes: Vision-Based Human Hand Tracking.
Technical Report CMU-CS-93-220, Carnegie Mellon University (Pittsburgh, PA) 1993.

[Rhyne86] Rhyne, J.R., and Wolf, C.G.
Gestural Interfaces for Information Processing Applications.
Technical Report RC12179, IBM T.J. Watson Research Center, IBM Corporation (Yorktown Heights, NY), September 1986.

[Robbe96] Robbe, S., Carbonell, N., and Valot, C.
Toward Usable Multimodal Command Languages: Definition and Ergonomic Assessment of Constraints on Users' Spontaneous Speech and Gestures.
In Proceedings of *International Conference of Spoken Language Processing (ICSLP'96)*, Vol. 3, 1655-1658. (Philadelphia, PA) October 1996.

[Rohlicek92] Rohlicek, J.R. et al.
Gisting Conversational Speech.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'92)*, Vol. II, 113-116. March 1992.

[Rose91] Rose, R.C.
Techniques for Information Retrieval from Voice Messages.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'91)*, 317-320. May 1991.

[Rosenfeld94] Rosenfeld, R., Thayer, E., Mosur, R., Chase, L., Weide, R., Hwang, M., Huang, X., and Alleva, F.
Improved Acoustic and Adaptive Language Models for Continuous Speech Recognition.
In Proceedings of *ARPA Spoken Language Technology Workshop (SLT'94)*, 106-109. March 1994.

[Rozak97] Rozak, M.
An Overview of the Microsoft Speech API.
Microsoft Corporation (Redmond, WA), 1997.
http://www.microsoft.com/directx/pavilion/dsound/overviewapi.htm

[Rubine91] Rubine, D.H.
The Automatic Recognition of Gestures.
PhD Thesis CMU-CS-91-202, Carnegie Mellon University (Pittsburgh, PA), December 1991.

[Rudnicky90] Rudnicky, A.I., Sakamoto, M., and Polifroni, J.H.
Spoken Language Interaction in a Goal-Directed Task.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'90)*, 45-48. 1990.

[Rumbaugh91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenson, W.
Object-Oriented Modeling and Design.
Prentice Hall (Englewood Cliffs, NJ), 1991.

[Rumelhart86] Rumelhart, D.E. and McClelland, J.L.
Parallel Distributed Processing: Exploration in the Microstructure of Cognition (Vols. 1&2).
MIT Press (Cambridge, MA), 1986.

[Sakoe71] Sakoe, H. and Chiba, S.
A Dynamic Programming Approach to Continuous Speech Recognition.
In Proceedings of *International Congress on Acoustics*, Paper 20 C-13. (Budapest, Hungary) 1971.

[Sakoe87] Sakoe, H. and Iso, K.
Dynamic Neural Networks — A New Speech Recognition Model Based on Dynamic Programming and Neural Network.
IEICE Technical Report 87, NEC Corporation.

[Salber93] Salber, D. and Coutaz, J.
Applying the Wizard of Oz Technique to the Study of Multimodal Systems.
In Proceedings of *EWHCI'93*, 219-230. (Moscow, Russia) August 1993.

[Salisbury90] Salisbury, M.W., Hendrickson, J.H. Lammers, T.L., and Fu, C.
Talk and Draw: Bundling Speech and Graphics.
Computer 23(8):59-65, 1990.

[Sankar93] Sankar, A. and Gorin, A.L.
Adaptive Language Acquisition in a Multisensory Device.
In *Artificial Neural Networks for Speech and Vision*, Mammone, R. (Ed.), 324-356. Chapman and Hall (London, UK), 1993.

[Schafer75] Schafer, R.W. and Rabiner, L.R.
Digital Representations of Speech Signals.
In *Proceedings of the IEEE* 63(4):662-667, April 1975.

[Schalkwyk98] Schalkwyk, J., Colton, D., and Fanty, M.
The CSLUsh Toolkit for Automatic Speech Recognition.
Center for Spoken Language Understanding, Oregon Graduate Institute of Science and Technology, March 1998.
http://www.cse.ogi.edu/CSLU/toolkit/

[Schenkel94] Schenkel, M., Guyon, I., and Henderson, D.
On-line Cursive Script Recognition Using Time Delay Neural Networks and Hidden Markov Models.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'94)*, (Adelaide, Australia) April 1994.

[Shankar93] Shankar, R.V. and Krishnaswamy, D.
Classification of Pen Gestures Using Learning Vector Quantization.
In Proceedings of *Neural and Stochastic Methods in Image and Signal Processing II*, Vol. 2032, 138-143. (San Diego, CA) July 1993.

[Sharma96] Sharma, R., Huang, T.S., and Pavlovi'c, V.I.
A Multimodal Framework for Interacting with Virtual Environment.
In Proceedings of *Human Interaction with Complex Systems: Conceptual Principles and Design Practice*, 53-71. 1996.

[Shaw70] Shaw, A.C.
Parsing of Graph-Representable Pictures.
JACM 17(3):453, 1970.

[Shimazu95] Shimazu, H. and Takashima, Y.
Multimodal Definite Clause Grammar.
Systems and Computers in Japan 26(3):93-102, March 1995.

[Simon89] Simon, J. and Baret, O.
Formes Régulières et Singulières: Application à la Reconnaisance de l'Ecriture Manuscrite.
(Regular and Singular Forms: Application to Handwriting Recognition.)
C.R. Acad. Scr. Paris 309(II):1901-1906, 1989.

[Smailagic96] Smailagic, A. and Siewiorek, D.P.
Modalities of Interaction with CMU Wearable Computers.
IEEE Personal Communications 3(1):14-25, 1996.

[SRAPI97] SRAPI Committee.
SRAPI Programmers Guide.
SRAPI Committee (Utah), 1997.
http://www.srapi.com/

[Stahl96] Stahl, H., Muller, J., and Lang, M.
An Efficient Top-down Parsing Algorithm for Understanding Speech by Using Stochastic, Syntactic, and Semantic Models.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'96)*, Vol. 1, 397-400. (Atlanta, GA) May 1996.

[Stock93] Stock, O.
The AlFresco Interactive System.
In Proceedings of *INTERCHI'93*, 523. (Amsterdam, Netherlands) April 1993.

[Stoehr95] Stoehr, E. and Lieberman, H.
HearingAid: Adding Verbal Hints to a Learning Interface.
In Proceedings of *ACM Multimedia'95*, 223-230. (San Francisco, CA) 1995.

[Stroustrup91] Stroustrup, B.
The C++ Programming Language.
Addison-Wesley (Reading, MA), 1991.

[Sturman94] Sturman, D.J. and Zeltzer, D.
A Survey of Glove-Based Input.
IEEE Computer Graphics and Applications 14(1):30-39, January 1994.

[Suhm97] Suhm, B.
Empirical Evaluation of Interactive Multimodal Error Recovery.
In Proceedings of *IEEE Workshop on Speech Recognition and Understanding (ASRU'97)*. (Santa Barbara, CA) December 1997.

[Sun97] Sun Microsystems, Inc.
Java Speech API White Paper.
Sun Microsystems, Inc. (Palo Alto, CA), 1997.
http://java.sun.com/products/java-media/speech/

[Tappert90] Tappert, C., Suen, C., and Wakahara, T.
The State of the Art in On-line Handwriting Recognition.
IEEE Transactions on Pattern Analysis and Machine Intelligence 12(8), 1990.

[Tebelskis90] Tebelskis, J. and Waibel, A.
Large Vocabulary Recognition Using Linked Predictive Neural Networks.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'90)*. April 1990.

[Thompson63] Thompson, F.B.
The Semantic Interface in Man-Machine Communication.
Technical Report RM 63TMP-35, General Electric Co. (Santa Barbara), September 1963.

[Tishby94] Tishby, N.Z. and Gorin, A.L.
Algebraic Learning of Statistical Associations.
Computer Speech and Language 8(1):51-78, 1994.

[Tsuboi92] Tsuboi, H. and Takebayashi, Y.
A Real-time Task-Oriented Speech Understanding System Using Keyword-Spotting.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'92)*, Vol. I, 197-200. (San Francisco, CA) March 1992.

[Tsuboi97] Tsuboi, H., Takebayashi, Y., and Hashimoto, H.
Spontaneous Speech Understanding Method Based on LR Parsing of Keyword Lattice.
Transactions of the Information Processing Society of Japan 38(2):260-269, February 1997.

[Vidal89] Vidal, E., Garcia, P., and Segarra, E.
Inductive Learning of Finite-State Transducers.
In *Structural Pattern Analysis*, Mohr, R., Pavlidis, T, and Sanfelin, A. (Eds.), 17-35. World Scientific (River Edge, NJ), 1989.

[Viterbi67] Viterbi, A.J.
Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm.
IEEE Transactions on Information Theory IT-13:260-269, April 1967.

[Vo93a] Vo, M.T. and Waibel, A.
A Multimodal Human-Computer Interface: Combination of Speech and Gesture Recognition.
In Adjunct Proceedings of *INTERCHI'93*. (Amsterdam, Netherlands) April 1993.

[Vo93b] Vo, M.T. and Waibel, A.
Multimodal Human-Computer Interaction.
In Proceedings of *ISSD'93*. (Waseda, Japan) 1993.

[Vo95] Vo, M.T., Houghton, R., Yang, J., Bub, U., Meier, U., Waibel, A., and Duchnowski, P.
Multimodal Learning Interfaces.
In Proceedings of *ARPA Speech Language Technology Workshop (SLT'95)*. (Austin, TX) 1995.

[Vo96] Vo, M.T. and Wood, C.
Building an Application Framework for Speech and Pen Input Integration in Multimodal Learning Interfaces.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'96)*, Vol. 6, 3545-3548. (Atlanta, GA), May 1996.

[Vo97] Vo, M.T. and Waibel, A.
Modeling and Interpreting Multimodal Inputs: A Semantic Integration Approach.
Technical Report CMU-CS-97-192, Carnegie Mellon University (Pittsburgh, PA), December 1997.

[Wagner83] Wagner, R.W.
Formal-Language Error Correction.
In *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Sankoff, D. and Kruskal, J.B. (Eds.), Chapter 13. Addison-Wesley (Reading, MA), 1983.

[Wahlster91] Wahlster, W.
User Discourse Models for Multimodal Communication.
In *Intelligent User Interfaces*, Sullivan, J.R. and Tyler, S.W. (Eds.), 45-68. ACM Press/Addison-Wesley (Reading, MA), 1991.

[Waibel89a] Waibel, A., Hanazawa, T., Hinton, G., Shiano, K., and Lang, K.
Phoneme Recognition Using Time-Delay Neural Networks.
IEEE Transactions on Acoustics, Speech, and Signal Processing 37(3):328-339, March 1989.

[Waibel89b] Waibel, A., Sawai, H., and Shikano, K.
Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'89)*, Vol. 1, 112-115. (Glasgow, UK) May 1989.

[Waibel89c] Waibel, A., Sawai, H., and Shikano, K.
Modularity and Scaling in Large Phonemic Neural Networks.
IEEE Transactions on Acoustics, Speech, and Signal Processing, 37(12):1888-1898, December 1989.

[Waibel90] Waibel, A. and Lee, K.F. (Eds.)
Readings in Speech Recognition
Morgan Kaufmann (San Mateo, CA), 1990.

[Waibel91] Waibel, A., Jain, A., McNair, A., Saito, H., Hauptmann, A., and Tebelskis, J.
JANUS: A Speech-to-Speech Translation System Using Connectionist and Symbolic Processing Strategies.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'91)*, Vol. 1, 365-367. (Toronto, Canada) 1991.

[Waibel96a] Waibel, A., Vo, M.T., Duchnowski, P., and Manke, S.
Multimodal Interfaces.
Artificial Intelligence Review, *Special Volume on Integration of Natural Language and Vision Processing*, McKevitt, P. (Ed.), 10(3-4):299-319, August 1996.

[Waibel96b] Waibel, A.
Translation of Spoken Dialogs.
IEEE Computer, 1996.

[Waibel97] Waibel, A., Suhm, B., Vo, M.T., and Yang, J.
Multimodal Interfaces for Multimedia Information Agents.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*. 1997.

[Wang95] Wang, J.
Integration of Eye-gaze, Voice and Manual Response in Multimodal User Interface.
In Proceedings of *IEEE International Conference on Systems, Man, and Cybernetics (ICSMC'95)*, 3938-3942. (Vancouver, Canada) October 1995.

[Ward91] Ward, W.
Understanding Spontaneous Speech: the Phoenix System.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'91)*, Vol. 1, 365-367. (Toronto, Canada) May 1991.

[Ward95] Ward, W. and Issar, S.
The CMU ATIS System.
In Proceedings of *ARPA Spoken Language Technology Workshop (SLT'95)*, 249-251. (Austin, TX) January 1995.

[Wilpon90] Wilpon, J.G., Rabiner, L.R., Lee, C.H., and Goldman, E.R.
Automatic Recognition of Keywords in Unconstrained Speech Using HMMs.
IEEE Transactions on Acoustics, Speech, and Signal Processing
ASSP-38:1870-1878, November 1990.

[Woods83] Woods, W.A.
Language Processing for Speech Understanding.
1983. Reprinted in *Readings in Speech Recognition*, Waibel, A. and Lee, K.F. (Eds.), Morgan Kaufmann (San Mateo, CA), 1990.

[Woszczyna98] Woszczyna, M.
Fast Speaker Independent Large Vocabulary Continuous Speech Recognition.
Ph.D. Thesis, Computer Science Department, University of Karlsruhe (Karlsruhe, Germany), February 1998.

[Yamada96] Yamada, H. and Nakano, Y.
Cursive Handwritten Word Recognition Using Multiple Segmentation Determined by Contour Analysis.
IEICE Transactions on Information and Systems E79-D(5):464-470, May 1996.

[Yang95] Yang, J. and Waibel, A.
Tracking Human Faces in Real-time.
Technical Report CMU-CS-95-210, Carnegie Mellon University (Pittsburgh, PA), 1995.

[Young92] Young, S.J.
Hidden Markov Model Toolkit v1.3 Reference Manual.
Cambridge University Engineering Speech Group, February 1992.

[Zeppenfeld93] Zeppenfeld, T., Houghton, R., and Waibel, A.
Improving the MS-TDNN for Word Spotting.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'93)*, Vol. 2, 475-478. (Minneapolis, MN) April 1993.

[Zeppenfeld97] Zeppenfeld, T., Finke, M., Ries, K., Westphal, M., and Waibel, A.
Recognition of Conversational Telephone Speech Using the JANUS Speech Engine.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*. (Munich, Germany) 1997.

[Zue90] Zue, V., Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., Seneff, S.
The VOYAGER Speech Understanding System: Preliminary Development and Evaluation.
In Proceedings of *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'90)*, Vol. 1, 73-76. (Albuquerque, NM) April 1990.

INDEX

application framework 5, 8, 14, 34, 68, 111

design process 1, 2, 3, 4, 5, 6, 37, 38, 53, 81, 82, 98, 104, 111, 117, 124, 142

face tracking 143

gaze tracking 3, 19, 20, 143, 146

gesture
- 3D 19, 32, 143, 146
- pen-based 3, 16, 18, 19, 146

JANUS 14, 56, 57, 58, 60, 62, 106, 109, 128, 151, 188, 189

JEANIE 27, 64, 111

lip-reading 19, 48, 143

multimodal
- input event 33, 35, 36, 38, 43, 48, 77, 78, 79, 80, 81, 82, 83, 85, 120, 126, 129, 140, 141, 142, 143, 147, 148
- input model 6, 40, 53, 85
- integration 5, 6, 25, 54, 82, 85, 109, 116, 117, 120, 121, 130, 139, 140, 141, 143, 154
- interaction 5, 20, 22, 23, 24, 26, 27, 29, 68, 102, 111, 125, 140
- interpretation 1, 4, 5, 6, 31, 33, 35, 36, 38, 53, 79, 81, 87, 98, 116, 120, 130, 140, 153
- semantic model 1, 5, 21, 40, 81, 111, 117, 140, 141, 145, 148

NetscapeSRec 59, 60, 62, 151

NPen++ 56, 58, 59, 63, 70, 180

QUARTERBACK 113, 116, 117

QUICKTOUR 6, 41, 100, 102, 103, 104, 109, 111, 115, 116, 117, 118, 119, 120, 122, 123, 124, 125

rapid prototyping 5, 27, 117, 140, 142

recognition
- gesture 2, 16, 18, 56, 63, 109, 125, 129, 130, 132, 136
- handwriting 15, 16, 35, 48, 56, 58, 77, 81, 143, 144
- speech 3, 8, 9, 11, 12, 13, 14, 15, 16, 19, 24, 48, 55, 56, 57, 58, 59, 62, 63, 64, 81, 82, 85, 109, 116, 124, 128, 132, 142, 146

SPHINX 56, 58, 60, 62, 106, 151

SRecServer 59, 60, 62, 63, 114, 151

TmplGRec 63, 64, 68, 70, 109, 114, 130, 151

XPRecServer 63, 70, 151