# Toward Widely-Available and Usable Multimodal Conversational Interfaces

by

Alexander Gruenstein

B.S. Stanford University (2003)
M.S. Stanford University (2003)

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009



Author .....
Department of Electrical Engineering and Computer Science
May 18, 2009

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

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

# Toward Widely-Available and Usable Multimodal Conversational Interfaces

by

Alexander Gruenstein

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

## Abstract

Multimodal conversational interfaces, which allow humans to interact with a computer using a combination of spoken natural language and a graphical interface, offer the potential to transform the manner by which humans communicate with computers. While researchers have developed myriad such interfaces, none have made the transition out of the laboratory and into the hands of a significant number of users. This thesis makes progress toward overcoming two intertwined barriers preventing more widespread adoption: *availability* and *usability*.

Toward addressing the problem of availability, this thesis introduces a new platform for building multimodal interfaces that makes it easy to deploy them to users via the World Wide Web. One consequence of this work is *City Browser*, the first multimodal conversational interface made publicly available to anyone with a web browser and a microphone. *City Browser* serves as a proof-of-concept that significant amounts of usage data can be collected in this way, allowing a glimpse of how users interact with such interfaces outside of a laboratory environment.

*City Browser*, in turn, has served as the primary platform for deploying and evaluating three new strategies aimed at improving usability. The most pressing usability challenge for conversational interfaces is their limited ability to accurately transcribe and understand spoken natural language. The three strategies developed in this thesis – context-sensitive language modeling, response confidence scoring, and user behavior shaping – each attack the problem from a different angle, but they are linked in that each critically integrates information from the conversational context.

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Scientist

*To "Norbert"*

*I can't wait to meet you*

# Acknowledgments

From the first day I met my advisor, Stephanie Seneff, I thought we would get along just fine. Luckily, I was right; more so than I could have imagined. Stephanie has been the best advisor, mentor, advocate, colleague, and friend I could have asked for. Thank you.

I am grateful to my committee members, Victor Zue and Randall Davis, for their comments. This thesis has improved a great deal based on their feedback.

It's been a distinct pleasure to work with the staff of the Spoken Language Systems group over the last five years. Jim Glass, in particular, has provided mentorship, encouragement, advice, and support on a daily basis. Chao Wang taught me how to do so many things I've lost count. T.J. Hazen helped with the recognition confidence module, and a major part of this thesis would not have been possible without his help. Scott Cyphers and Lee Hetherington helped with innumerable technical challenges. Finally, I am grateful to Marcia Davidson, who was always ready with a laugh to buy whatever random item I needed on very short notice.

Collaborating with Ian McGraw on all things WAMI has been amazingly fun and rewarding. Sean Liu has been a constant collaborator on *City Browser*, and I shudder to think of what the interface might have looked like without him.

I'd like to thank my officemates, Ali Mohammad, Harr Chen, Tara Sainath, and Yuan Shen, who have provided friendship, diversion, advice, and many fascinating discussions. I've also benefited from my interactions other SLS students, including Ibrahim Badr, Brad Cater, Ghinwa Choueiter, Ed Filisko, Paul Hsu, John Lee, JingJing Liu, Gary Matthias, Liz Murnane, Alex Park, Mitchell Peabody, Ken Schutte, Han Shu, Yushi Xu, Brandon Yoshimoto, and Helen You.

It was only with the help of a number of collaborators that the automotive *City Browser* system could be created, and data collected. I'd like to thank in particular Jeff Zabel, Shannon Roberts, Jarrod Orszulak, and Bryan Reimer.

My interest in the field began at Stanford, under the mentorship of Stanley Peters and Oliver Lemon. I am particularly indebted to Oliver, both for his friendship, and for teaching me what it is to do research. If not for him, I have no idea what I would be doing today.

Another important experience over the last five years was meeting other young researchers in the field at the *Young Researchers Roundtable on Spoken Dialogue Systems*. My friendships with Verena Rieser and Mihai Rotaru, in particular, stand out – and I thank them for making so many trips much more fun and interesting.

I could never have written this thesis without the constant love and support of my family and friends. My parents, John and Carolyn, and my sisters, Cassie and Elizabeth, have always supported me and loved me unconditionally. Justin, who feels more like family than friend, is always challenging me, and making me laugh. I would be lost without my wife Anna, who keeps me sane, happy, focused, and relaxed; and somehow put up with five years of this.

This research is funded in part by the T-Party project, a joint research program between MIT and Quanta Computer Inc., Taiwan.

# Contents


**1 Introduction** **19**
- 1.1 Multimodal Interfaces on the Web . . . 21
- 1.2 Data Collection and Annotation . . . 21
- 1.3 Context-Sensitive Language Modeling . . . 22
- 1.4 Context-Sensitive Confidence Scoring . . . 22
- 1.5 Contextual User Utterance Shaping . . . 23
- 1.6 Thesis Outline . . . 24

**2 *City Browser*: A Widely Available Multimodal Conversational Interface** **27**
- 2.1 User Interface . . . 28
- 2.2 Architectural Overview . . . 30
- 2.3 Web Availability . . . 30
- 2.4 Graphical User Interface . . . 33
  - 2.4.1 Multimodal Error Correction . . . 33
- 2.5 Natural Language Processing Pipeline . . . 35
  - 2.5.1 Speech Recognizer . . . 35
  - 2.5.2 Natural Language Parser . . . 35
  - 2.5.3 Discourse and Gesture Resolution . . . 35
  - 2.5.4 Dialogue Management . . . 36
  - 2.5.5 Natural Language Generation . . . 37
  - 2.5.6 Suggestions Module . . . 37
  - 2.5.7 Confidence Annotator . . . 37
  - 2.5.8 Application . . . 37
  - 2.5.9 Speech Synthesis . . . 37
- 2.6 Database Creation via Web Crawling . . . 38
- 2.7 Related Work . . . 38
  - 2.7.1 Web-Based Speech Interfaces . . . 38
  - 2.7.2 Multimodal Conversational Interfaces . . . 40
  - 2.7.3 Widely Available Multimodal Speech Interfaces . . . 40
- 2.8 Summary . . . 41

**3 The WAMI Toolkit and Example Applications** **43**
- 3.1 Toolkit Configurations . . . 43
  - 3.1.1 Toolkit-Only . . . 43

3.1.2 Toolkit+Portal . . . 44
3.2 Lightweight Semantic Understanding . . . 45
3.2.1 Incremental Understanding . . . 45
3.3 WAMI Applications . . . 49
3.3.1 SLS Applications . . . 49
3.3.2 Student Applications . . . 51
3.4 Conclusion . . . 56

**4 Corpora** **57**
4.1 Overview of Corpora . . . 57
4.1.1 *Tablet* Corpus . . . 58
4.1.2 *Web* Corpus . . . 60
4.1.3 *Car* and *Car-Pilot* corpora . . . 63
4.2 Comparison to Similar Corpora . . . 66
4.3 Annotation Tools . . . 68
4.4 Summary . . . 69

**5 Context-Sensitive Language Modeling** **73**
5.1 Background . . . 74
5.1.1 $n$-gram Language Models . . . 76
5.1.2 Probabilistic context-free grammars . . . 77
5.1.3 Training Corpora . . . 78
5.1.4 Word Error Rate . . . 79
5.2 Related Work . . . 79
5.2.1 Limitations . . . 82
5.3 Contextualized Semantic Classes . . . 83
5.3.1 Cues . . . 86
5.3.2 Scalability and Flexibility . . . 86
5.4 Conclusion . . . 87

**6 An Empirical Evaluation of Contextualized Semantic Classes** **89**
6.1 Experiments in the Flight Reservation Domain . . . 89
6.1.1 Verbal Cues . . . 90
6.1.2 Prompt cues . . . 92
6.1.3 Experimental Conditions . . . 92
6.1.4 Experimental Results . . . 94
6.2 *City Browser* Experiments . . . 97
6.2.1 Graphical and Implicit Cues . . . 98
6.2.2 Experiments on the *Tablet* corpus . . . 98
6.2.3 Experiments on the *Car-Pilot* and *Car* corpora . . . 100
6.3 Conclusion . . . 101

**7 Context-Sensitive Confidence Scoring** **103**
7.1 Related Work . . . 106
7.1.1 Recognition Confidence Scores . . . 107
7.1.2 Language Understanding . . . 107
7.1.3 Stochastic Dialogue Management . . . 108
7.1.4 Functional Accuracy and Multimodal Disambiguation . . . 109
7.2 Data Exploration . . . 110
7.2.1 Reduced Response Set . . . 110
7.2.2 Hypotheses with Errors . . . 111
7.3 Response Confidence Scoring Algorithm . . . 111
7.4 Training: Data and Annotation . . . 114
7.5 Implementation Considerations . . . 115
7.6 Conclusion . . . 116

**8 Response Confidence Scoring Algorithm Evaluation** **117**
8.1 Baseline . . . 117
8.2 Feature Sets . . . 118
8.3 Evaluation Metrics . . . 119
8.3.1 Statistical Significance . . . 120
8.4 Offline Experiments with the *Web* Corpus . . . 120
8.4.1 Results . . . 122
8.5 Online Experiments . . . 123
8.5.1 Results . . . 124
8.5.2 Latency Considerations . . . 127
8.6 Conclusion . . . 127
8.6.1 Future Work . . . 129

**9 Contextual Utterance Shaping** **131**
9.1 Related Work . . . 132
9.1.1 Multimodal Help Modules . . . 133
9.1.2 Targeted Help . . . 133
9.1.3 Artificial Language . . . 136
9.2 Context-Sensitive Suggestions Interface . . . 136
9.2.1 Look and Feel . . . 137
9.2.2 Goals . . . 137
9.2.3 Suggestion Types . . . 142
9.3 Templates . . . 143
9.3.1 Examples . . . 144
9.4 Evaluation . . . 148
9.4.1 Survey Results . . . 148
9.4.2 Usage Data . . . 148
9.5 Conclusion . . . 152

**10 Conclusion and Future Work** **155**
10.1 Future Work . . . 156

# List of Figures


2-1 An example interaction with *City Browser* . . . . . 28
2-2 Screenshot of *City Browser* . . . . . 29
2-3 *City Browser*'s information flow and web-accessible architecture . . . 31
2-4 Correctable N-best list. . . . . 34
2-5 The TINA parser's hierarchical semantic representation for *Show me an Italian restaurant in Boston.* . . . . . 36

3-1 WAMI "Toolkit-only" configuration . . . . . 44
3-2 WAMI Toolkit+Portal configurations . . . . . 46
3-3 Screenshot and complete source code for a simple WAMI "mash-up" . 47
3-4 Sample JSGF grammar snippet for a WAMI application that makes use of a lightweight semantic understanding module, and an example utterance with associated slot/value output. . . . . 48
3-5 The *Word War* game. . . . . 49
3-6 The *Rainbow Rummy* game. . . . . 50
3-7 A translation game where students translate sentences related to travel. 51
3-8 *Flight Browser*, a multimodal conversational interface for booking flights. 52
3-9 A conversational interface to a home entertainment system. . . . . 52
3-10 *Voice Race* and *Voice Scatter* games. They are publicly available at `http://quizlet.com`. . . . . 54
3-11 A WAMI mash-up with several Google AJAX APIs. . . . . 55
3-12 Speech interface to *Media Enclave.* . . . . . 55
3-13 The *ESPN 1-click* application. . . . . 56

4-1 Screenshot of the *City Browser* interface, as it appeared during collection of the *Tablet* corpus. . . . . 58
4-2 Screenshot of the interface used to collect the *Web* dataset . . . . . 61
4-3 Screenshots of the *City Browser* interface deployed in the car. . . . . 64
4-4 Screenshots of the web-based transcription tool used to transcribe and annotate the *City Browser* corpora. . . . . 70

5-1 Context-sensitive language modeling as part of the information flow in a typical conversational interface . . . . . 75
5-2 An example VoiceXML program, and a dialogue that it allows . . . . 80
5-3 A (very small) example corpus of transcribed utterances from an actual system interaction, tagged with contextualized semantic classes. . . . 84

5-4 Potential ranges for the number of lexical items in a contextualized semantic class, as a function of the type of cue used to populate that class. . . . . 87
5-5 Several prompts that include a request for an airline name. . . . . 88

6-1 An example interaction with MERCURY, where user utterances have been tagged with contextualized semantic classes created with *verbal* cues. . . . . 91
6-2 An example interaction with MERCURY, where user utterances have been tagged with contextualized semantic classes created with *prompt* cues. . . . . 93
6-3 An example interaction with *City Browser*, where user utterances have been tagged with the appropriate contextualized semantic classes, based on *graphical* and *implicit* cues . . . . . 99

7-1 Two example conversations in which utterance *U3* might sensibly appear; in each case, a potential recognition hypothesis with 50% word error rate is shown in bold below it. . . . . 105
7-2 The 5-best recognition hypotheses for the utterance *Directions to thirty two Vassar street* and the system response generated by each one depending on whether each is interpreted in the context of conversation (a) or (b) from Figure 7-1 above. . . . . 106
7-3 The mean N-best recognition hypothesis list length, mean number of unique parses derived from the N-best list of recognition hypotheses, and mean number of unique system responses derived from those parses, given a maximum recognition N-best list length. . . . . 111
7-4 Histogram of utterances from the *City Browser Car* corpus for which the top-scoring system response was annotated as correct or incorrect, as a function of the number of word errors made by the speech recognizer in the hypothesis associated with that top scoring response. . . 112
7-5 The feature extraction and classification process . . . . . 113

8-1 Algorithm for calculating the F-measure confusion matrix . . . . . 121
8-2 Receiver Operator Characteristic (ROC) curves (averaged across each cross-validation fold) comparing the baseline to response confidence models using combinations of each feature set. . . . . 124
8-3 Receiver Operator Characteristic (ROC) curves on the tuning set. . . 125
8-4 Receiver Operator Characteristic (ROC) curves for the response confidence module and the utterance confidence baseline module. . . . . 126
8-5 Histograms of scores for the top-ranked response for responses annotated as correct or incorrect. . . . . 126
8-6 Response latency in the *Car* corpus. . . . . 128

9-1 Example dialogue including targeted help . . . . . 134
9-2 Snippet of an example interaction between a user and a Speech Graffiti interface for obtaining movie information . . . . . 136

9-3 Screenshots of *City Browser* showing context-sensitive suggestions in the interface as it appeared during the collection of the *Web* corpus . 138
9-4 Screenshot of the suggestions screen shown as part of the automotive version of *City Browser* used to collect data for the *Car-Pilot* and *Car* corpora . . . . . 139
9-5 Suggestions list evolving over the course of a dialogue . . . . . 140
9-5 Suggestions list evolving over the course of a dialogue (continued from previous page). . . . . 141
9-6 Template specification constraints . . . . . 145
9-7 Example global suggestion template groups that fill in values from a randomly selected database entry . . . . . 146
9-8 Examples of subsetting and anaphoric suggestions templates. . . . . 147
9-9 Example of the contrastive suggestion *What about in Boston?* . . . . 149
9-10 Survey results from the *Web* corpus pertaining to the context-sensitive suggestions module. . . . . 150
9-11 Usage and effect of the context-sensitive suggestions in the *Web* corpus. 151

# List of Tables


2.1 Examples of utterances where the semantic representation produced by the parser will be augmented with gestural or contextual information. 36

4.1 Overview of *City Browser* corpora . . . . . 59
4.2 Tasks assigned to subjects in the *Web* data collection effort in the order in which they were performed. . . . . 62
4.3 Tasks assigned in the *Car-Pilot* and *Car* datasets . . . . . 65
4.4 Characteristics of corpora of similar multimodal conversational interfaces 67

6.1 Contextual semantic classes cued by system prompts. . . . . 92
6.2 Experimental conditions for MERCURY experiments in which the number of cities supported by the system, and the within-class weights for those city names are varied. . . . . 94
6.3 Word error rates for the baseline class $n$-gram language model, and two context sensitive language models incorporating contextualized semantic classes derived from *verbal* and *prompt* cues respectively, in four conditions . . . . . 95
6.4 Word error rates of the baseline class $n$-gram language model and one incorporating contextualized semantic classes derived from *verbal* cues broken down by active classes . . . . . 95
6.5 Word error rates of the baseline class $n$-gram language model and one incorporating contextualized semantic classes derived from *prompt* cues broken down by active classes . . . . . 97
6.6 Word error rate and proper noun error rates for the per-metro language model and the language model with graphically- and implicitly-cued contextualized semantic classes. . . . . 100
6.7 Word error rate and proper noun error rates – when synthetic training data is included – for the per-metro language model and the context-sensitive language model with graphically- and implicitly-cued contextualized semantic classes. . . . . 100
6.8 Word error rates and proper noun error rates for the per-metro language model and the context-sensitive language model with graphically- and implicitly-cued contextualized semantic classes. Trigram language models were trained on the *Car-Pilot* corpus, and tested using the *Car* corpus . . . . . 101

7.1 Word error rates (WER) for several research prototype spoken or multimodal conversational interfaces. . . . . 103

8.1 Features used to train the acceptability classifier . . . . . 118
8.2 The set of possible values for non-numerical features, which are converted to sets of binary features. . . . . 119
8.3 Average F-measures obtained via per-user cross-validation . . . . . 123

# Chapter 1

# Introduction

Multimodal conversational interfaces allow humans to interact with a computer using a combination of spoken natural language and other communication channels, including the mouse and keyboard, pen or stylus, or gestures made using the hands, face, or body. They offer the potential to transform the manner by which humans interact with computers. Spoken natural language is an essentially effortless means of communication for most people, and when paired with the ability to interact with some sort of visual display, it becomes an extremely effective way to sift through information or communicate complex ideas.

While researchers have developed myriad multimodal conversational interfaces [49, 50, 99, 40, 91, 33, 57], none have made the transition out of the laboratory and into the hands of a significant number of users. In sharp contrast, conversational interfaces available via the telephone have been widely adopted, largely because they are readily accessible to anyone with a telephone. They are widely commercially available, and even systems developed in the laboratory have been used to collect large amounts of data from real user interactions (*e.g.*, [101, 70]).

This thesis is concerned with two major barriers that prevent the widespread adoption of more sophisticated multimodal conversational interfaces:

**Availability:** One important constraint on the development of multimodal conversational interfaces has been the difficulty of making them available to a large number of users. User interaction data is critical, both so that developers can learn through trial and error – and controlled experiments – how to build better interfaces, and to collect data to train the statistical models critical to speech and natural language understanding. Unfortunately, it's often difficult and/or expensive to distribute prototype applications to real users and retrieve usage data indicating how they were used, especially when such systems are still rough around the edges and built using research-grade components. As a result, experimental subjects typically must interact with systems deployed on hardware in the laboratory. They are given training on how to use the interface, and then their actions are monitored by an experimenter. While such user testing can be quite valuable, especially at an early stage of system development, it is time-intensive, expensive, and artificial. Most importantly, it may be a poor indicator of how a user will behave outside of the lab, while completing a

real task [2].

**Usability:** Telephone-based conversational interfaces typically provide transactional interactions, in which a caller is prompted to provide enough information to complete a transaction, such as buying a train ticket. In such systems, the goal is typically to complete the transaction as quickly as possible and end the call. On the other hand, multimodal conversational interfaces typically provide more open-ended, exploratory interactions. They typically offer a relatively complex set of features, and provide users a wide latitude in conversational flow. Because users also have access to a rich graphical user interface (GUI), interaction is typically less directed – as users interact both with the GUI and via a natural language interface. As the complexity of conversational interfaces increases, speech recognition and natural language accuracy tend to suffer [68]. Relatively poor speech recognition performance, in turn, makes multimodal conversational interfaces less usable, and potentially counter-productive. As such, it is key both that users understand how to interact with the interface, and that, when they do speak, they are understood.

Availability and usability are, of course, intertwined. With greater availability, researchers can collect usage data which will help them both tune existing systems and develop novel approaches to increasing usability. As easily accessible systems become more compelling and usable, there is greater impetus to further increase their availability, perhaps at a higher cost. This thesis is unified around bridging the two barriers of availability and usability.

Toward addressing the problem of availability, this thesis introduces a new platform for building multimodal interfaces that makes it easy to deploy them to users via the World Wide Web. Deploying interfaces via the web allows for simple iterative refinement – updates need only be made to the software on the server – and, moreover, provides for a simple, standard, and familiar way for users to access them: via a web browser. One consequence of this work is *City Browser*, the first multimodal conversational interface made publicly available to anyone with a web browser and a microphone. *City Browser* is a map-centric interface that gives users access to a variety of urban information via a rich multimodal interface.

*City Browser*, in turn, has served as the primary platform for deploying and evaluating three strategies presented in this thesis aimed at improving usability. The most pressing usability problem for feature-rich conversational interfaces is their limited ability to accurately recognize and understand spoken natural language. The three strategies developed in this thesis – context-sensitive language modeling, response confidence scoring, and user behavior shaping – each attack the problem from a different angle, but they are linked in that each critically integrates information from the conversational context. Each demonstrates that contextual knowledge – typically used by conversational interfaces to interpret natural language – can be put to use to improve accuracy in other ways. Increased accuracy, in turn, provides for the development of interfaces in which users have greater latitude in taking conversational initiative, leading to more natural and effective interactions.

Taken together, the contributions outlined below represent a significant change

in the manner by which multimodal interfaces may be conceived, implemented, and deployed.

## 1.1 Multimodal Interfaces on the Web

A major technical contribution presented in this thesis is the development of *City Browser*, a conversational interface that is accessed by users with a web browser and a microphone. As a proof-of-concept, *City Browser* has shown that it is feasible to deploy a rich multimodal interface via the web using modern web-programming techniques. Rich web applications are now common, and are rapidly displacing native programs for performing tasks like word processing, writing e-mail, and managing calendars – while, at the same time, giving rise to entirely new paradigms, such as social networking. However, as rich as these web applications have become, they do not allow for the use of spoken natural language input and output.

*City Browser* has demonstrated that speech technology can be successfully integrated with rich web applications, making it possible for researchers to easily deploy interfaces available to users around the world. Moreover the architecture underlying *City Browser* has been generalized, and made available as the WAMI toolkit [34]. WAMI has been used to develop a variety of web-based speech interfaces at MIT – both conversational and not. Now, after being released as open-source software, it is already being integrated into a number of projects around the world.

## 1.2 Data Collection and Annotation

Collecting and analyzing usage data is critical to improving any computer interface, as system designers can never anticipate exactly what will be difficult, obvious, or easy for users. This is especially true for interfaces which rely on speech recognition – transcribed speech data has proved critical, for example, to improving the accuracy of telephone-based conversational interfaces (see, *e.g.*, Figure 10 of [101]). Such data is critical, both so that system developers can iteratively improve the design of their systems, and because many of the models underlying speech recognition and natural language processing are stochastic – and rely on training data to estimate probabilities. There is every reason to believe, then, that collecting usage data from multimodal conversational interfaces could dramatically improve their performance.

As such, an important contribution of this thesis is several transcribed and annotated corpora of users interacting with *City Browser*. Data was collected in several conditions: in the lab, via the web, and in an automobile. It was transcribed, and much of it was annotated with regards to whether the system's response was appropriate. The collected corpora serve as the basis for many of the experiments in this thesis, and will hopefully prove useful to other researchers as well.

## 1.3 Context-Sensitive Language Modeling

A critical factor contributing to the usability of a multimodal dialogue system is the accuracy with which the speech recognizer transcribes the words spoken by a user. Indeed, many telephone-based dialogue systems are designed such that users are carefully shepherded through a series of prompts, each designed to elicit a very specific type of response (*e.g.*, a city name). So long as users are cooperative, speech recognition accuracy can be quite high, since the system can form strong expectations about what will be said.

While this type of interaction may work well in some transactional domains – *e.g.*, purchasing a train ticket – it is often awkward in more exploratory domains, especially when a visual display is available. For example, it is difficult to anticipate what a user browsing through several restaurant reviews will want to do next: get directions to a particular restaurant, change his search constraints, look for a movie theater nearby, and so on. To support such open-ended conversations, conversational interfaces must be prepared to understand a wider variety of natural language at any given time.

While users may have more freedom, contextual expectations about what they are likely to say next still do exist. Perhaps a user is more likely to ask about making a reservation at one of the restaurants on the screen than at one which hasn't been mentioned yet. It is an open problem, however, how best to actually make use of weaker contextual expectations like these to improve speech recognition accuracy.

In this thesis, a new technique is described that integrates such expectations into the language model used by the speech recognizer to estimate the *a priori* probability that a user will utter a particular sequence of words. In particular, *contextualized semantic classes* are described, which group words and phrases together based on conversational context, and can be updated over the course of a conversation to reflect such expectations in a class $n$-gram language model. This technique is shown to significantly increase accuracy in *City Browser*, as well as in MERCURY, a conversational interface for making flight reservations.

## 1.4 Context-Sensitive Confidence Scoring

Despite a system designer's best intentions, it is inevitable that speech recognition errors will occur – that is, that a user's utterance will not be transcribed accurately. Generally, conversational interfaces rely on speech recognition confidence scores, which are meant to indicate the likelihood that a transcription error has occurred, in order to decide how to make use of the recognition hypothesis. If the confidence is high, then the transcribed input will be processed, and a response will be provided. If confidence is low, the system may ignore the input, and indicate that it has not understood.

Confidence scoring modules are typically developed in a data-driven way, using a corpus of transcribed utterances. They are designed to determine if each word spoken by the user was understood correctly. However, from the user's perspective,

what is truly important when interacting with a conversational interface is whether the system responded appropriately, not whether the utterance was transcribed correctly. At times, an errorful recognition hypothesis may result in a correct response; conversely, a near-perfect hypothesis oftentimes evokes an incorrect system response. As such, it makes sense to integrate the entire natural language processing pipeline into the decision, and to recast the problem from one of assigning confidence scores to recognition hypotheses to one of assigning them to candidate system responses. If the system can't formulate a response in which it has high confidence, then it should clarify, indicate non-understanding, and/or attempt to provide appropriate help.

This thesis describes a novel *response confidence scoring* approach that integrates information obtained not only from the speech recognition process, but also culled from the process of generating and compiling a list of candidate system responses. Uncertainty in the speech recognizer's transcription – expressed via a ranked list of hypotheses – is transformed into uncertainty over a set of system responses. In this way, the contextual information used in the natural language generation and response generation process can be brought to bear in assessing a confidence score. The approach is shown to improve the functional accuracy of *City Browser*, both in offline experiments and in a live system deployment.

## 1.5 Contextual User Utterance Shaping

While interacting with a conversational system where the system directs the user through a series of prompts can be frustrating, it can be equally frustrating for users to have no idea what to say or do. In preliminary studies with *City Browser*, it was observed that subjects often had a difficult time deciding what to do next, and then formulating an utterance to express this desire in terms that the system could understand. Even after successfully accessing some system capabilities, users weren't sure what else it might be capable of; often, they greatly over- or under-estimated the language understanding capabilities of, and knowledge available to, the system. Moreover, as they explored the bounds of what the system could understand, they sometimes were misunderstood; yet, they could not always determine if this was due to speech recognition errors, or if the utterance was simply one the system was not programmed to understand. This distinction is critical: if a user speaks within the bounds of what the system should be able to understand, but is met with failure, then she has no way of knowing whether or not she should repeat the same utterance or try a different tactic.

While no conversational interface will be able to understand every utterance, the visual display of a multimodal interface offers a simple, yet compelling way to help users understand what they can say at any given point in the conversation: via the use of context-sensitive utterance suggestions. This thesis describes the development and evaluation of just such a module, which takes into account the current conversational context to produce a set of relevant suggested utterances. The suggestions are displayed unobtrusively to the user, and are updated as the conversation progresses. In this way, users can come to understand the range of capabilities provided by *City*

*Browser*, and how to access those capabilities via natural language. Humans are quite flexible, and by "shaping" their behavior, the quality of their experience can be greatly improved, without requiring any effort to augment the natural language understanding capabilities of the interface.

## 1.6 Thesis Outline

The overarching goal of this thesis is largely pragmatic: to make progress toward improving the usability and availability of multimodal interfaces, especially those capable of interacting via spoken natural language conversations. Each chapter, listed below, describes work toward such progress.

**Chapter 2: *City Browser*: A Widely Available Multimodal Conversational Interface**

This chapter introduces *City Browser*. It covers system capabilities, and underlying architecture. It highlights how the modules described in subsequent chapters fit into the overall architecture. Earlier versions of *City Browser*'s architecture have previously been described in [38, 37].

**Chapter 3: The WAMI Toolkit and Example Applications**

This chapter gives an overview of WAMI, a toolkit for building web-accessible multimodal interfaces that has grown out of the architecture developed for *City Browser*. A number of applications developed with WAMI are discussed, demonstrating the broad impact of the web-based multimodal architecture originally developed for *City Browser*. The WAMI Toolkit has previously been discussed in [34].

**Chapter 4: Corpora**

This chapter provides an overview of data collection efforts involving *City Browser*. In particular, four transcribed and annotated corpora are discussed. They were collected in three experimental conditions: in the lab, via the web, and in an automobile. These efforts have previously been discussed, in much less detail, in [36, 37, 35].

**Chapter 5: Context-Sensitive Language Modeling**
**Chapter 6: An Empirical Evaluation of Contextualized Semantic Classes**

In Chapter 5, a context-sensitive language modeling technique is developed in which *contextualized semantic classes* are used to dynamically incorporate conversational expectations into an $n$-gram language model. In Chapter 6, the technique is evaluated using *City Browser* interaction data, and using a corpus of interactions with MERCURY, a conversational flight reservation system [77]. The evaluation shows significant accuracy improvements in a variety of conditions. Some of the experimental results have previously appeared in [36, 39].

**Chapter 7: Context-Sensitive Confidence Scoring**
**Chapter 8: Response Confidence Scoring Algorithm Evaluation**

In Chapter 7, a context-sensitive confidence scoring technique is developed that assigns confidence scores to system responses, rather than to recognition hypotheses, as is typical. In Chapter 8, the approach is evaluated, showing that it leads to improved functional accuracy in *City Browser*. A description of the technique, and initial experimental results, previously appeared in [32].

**Chapter 9: Contextual Utterance Shaping**

This chapter focuses on a context-sensitive technique for "shaping" user behavior, so that users can speak in a way that is more likely to be understood by a conversational interface. It describes a module for generating and presenting graphically a list of contextually-relevant utterance suggestions, which are updated as the conversation progresses. Survey results show that subjects appreciated these suggestions when interacting with *City Browser*. A brief discussion of the module, and a preliminary analysis, appeared in [37].

# Chapter 2

# *City Browser*: A Widely Available Multimodal Conversational Interface

This chapter gives an overview of the interface and architecture of *City Browser*, a multimodal conversational system used in this thesis as an experimental platform. Two points regarding the architecture highlight contributions of this thesis. First, *City Browser* was the first multimodal conversational interface deployed to the public that could be accessed by anyone with a web browser and a microphone. To enable this novel interaction paradigm, several key architectural components were developed, which have subsequently been refined and released as the WAMI toolkit. WAMI's usefulness and impact is discussed in Chapter 3.

Second, while *City Browser* relies on a natural language processing pipeline previously used in quite a number of conversational interfaces, three new general purpose components implementing contributions of this thesis have also been integrated: the response confidence annotator, the context-sensitive suggestions module, and the context-sensitive language model. Of particular note from an architectural standpoint is the confidence annotator. It sends multiple speech recognition hypotheses through the natural language processing pipeline to produce a set of candidate system responses, which are then scored. Traditionally, natural language processing results in only a single response.

This chapter first introduces the *City Browser* user interface, focusing on its natural language understanding capabilities and graphical user interface. It then provides an overview of the system architecture. In particular, the client-server model that makes *City Browser* widely available as a web interface is highlighted, as are the context-sensitive natural language processing components detailed in subsequent chapters. Finally, related work on building multimodal conversational interfaces and on making them widely available is discussed.

It is important to note that *City Browser* was not created from scratch for this thesis. It has borrowed from, and built upon, two existing interfaces in particular: DINEX [78] and VOYAGER [99]. Both are database-driven conversational interfaces with information about a few hundred points of interest in the Boston area. The

| | |
|---|---|
| $U_1$: | Show me 32 Vassar Street in Cambridge. |
| $S_2$: | Here is 32 Vassar Street in Cambridge. [*shown on map*] |
| $U_3$: | Are there any Indian restaurants near here? |
| $S_4$: | There are 10 Indian restaurants near 32 Vassar Street in Cambridge. The particular neighborhoods are Central Square, Back Bay, Fenway, and Inman Square.[*results displayed*] |
| $U_5$: | What is the nearest subway stop to India Pavilion? |
| $S_6$: | Central Square station is located in Central Square, Cambridge at the intersection of Massachusetts Avenue and Western Avenue, Prospect Street, and Magazine Street. It is on the Red Line. |
| $U_7$: | Are there any museums in this area? [*circles an area of the map*] |
| $S_8$: | There are two museums in the region you specified. The Isabella Stewart Gardner Museum and the Museum of Fine Arts. |
| $U_9$: | Please give me driving directions to this museum [*circles one of the museums*] from 77 Massachusetts Avenue in Cambridge. |
| $S_{10}$: | Here are driving directions from 77 Massachusetts Avenue in Cambridge to the Museum of Fine Arts. [*shows driving directions on the map*] |
| $U_{11}$: | Are there any Italian restaurants along this street. [*draws line*] |
| $S_{12}$: | There are 12 Italian restaurants along this street. [*shown on map*]. The particular neighborhoods are Back Bay and Theater District. |
| $U_{13}$: | Show the web page for this one. [*circles a restaurant*] |
| $S_{14}$: | OK. [*displays web page from which the data was harvested*] |
| $U_{15}$: | I'd like to add a landmark here. [*clicks mouse on location*] |
| $S_{16}$: | OK. [*dialogue box displayed for user to type name*] |
| $U_{17}$: | [*user types: "Fenway Park"*] |
| $S_{18}$: | OK. I have added a landmark named Fenway Park. [*shows it*] |
| $U_{19}$: | Are there any cheap restaurants near Fenway Park? |
| $S_{20}$: | There are 12 inexpensive restaurants near Fenway Park.[*displayed*] Some of the options are pizza, Thai, and Italian. They are found mostly on Peterborough Street and Beacon Street. |

Figure 2-1: An example interaction with *City Browser*. $U$ indicates User; $S$ indicates System. Gestures and system actions are bracketed.

grammars for natural language parsing and generation, dialogue management rules, and discourse resolution rules from those systems served as a starting point for the development of *City Browser*.

## 2.1 User Interface

Users interact with *City Browser* by navigating to its web page in a web browser. Once there, they are greeted by *City Browser* and can have an interaction like the one shown in Figure 2-1. A screenshot taken in the course of this interaction is depicted in Figure 2-2.

As the example interaction and screenshot demonstrate, users can interact both via spoken natural language and by clicking or drawing on the graphical user interface (GUI). The system responds by speaking, and by updating the GUI appropriately. *City Browser* has knowledge about restaurants, museums, hotels, and subway stations in 11 major metropolitan areas in North America. Metropolitan areas typically consist of a major city and 50-250 nearby cities and towns, altogether typically containing

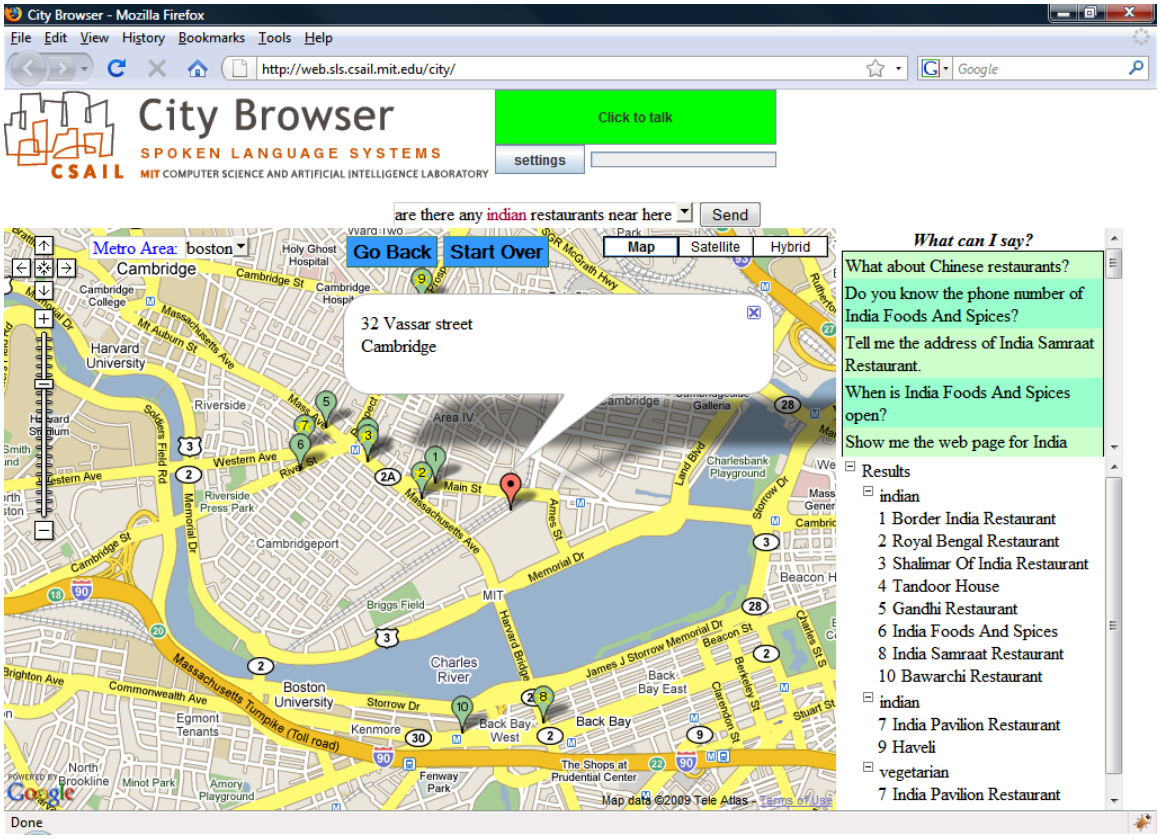



Figure 2-2: Screenshot of the *City Browser* interface in a web browser following utterance $U_3$ of the example interaction in the previous figure. At the top, there is a large button that the user presses to start speaking, with a bar underneath that moves left and right as users speak. Immediately below the bar is the top recognition hypothesis for the user's previous utterance, shown as a correctable N-best list. In the upper-right corner, suggestions of what to say next are shown; below that is a list of restaurants returned in response to the user's search request. These restaurants are shown as the numbered markers on the map at the center. In the top left corner of the map is a control that allows the user to change the current metropolitan area. To the right of it are buttons that allow the user to *go back* (undo the previous utterance) and *start over*. Standard map controls also overlay the map for zooming, panning, and switching to satellite or hybrid map views.

thousands of restaurants and hotels, and tens or hundreds of museums. In addition, *City Browser* can show addresses on the map, and provide driving directions. Users can click on search results shown on the map to get more information about them. They can constrain searches by drawing to outline or indicate geographical areas. Finally, they can customize *City Browser* by adding personal landmarks to the map – as in utterances 15–18 of Figure 2-1.

## 2.2 Architectural Overview

Figure 2-3a shows the conceptual flow of information through the *City Browser* architecture. The user's speech is captured as audio and streamed to the speech recognizer. It, in turn, produces a list of N-best hypotheses of the words the user said. Each hypothesis is then passed to a pipeline of natural language processing components. The hypothesis is parsed, yielding a semantic representation, which is then augmented with any gestures or clicks captured by the GUI, and context-resolved with discourse information. The context-resolved semantic representation is passed to the dialogue manager, which produces a response, often by using provided constraints to search a database. An appropriate natural language response is then created by the natural language generator, as are a list of context-sensitive suggestions of what to say next to be displayed along with the response.

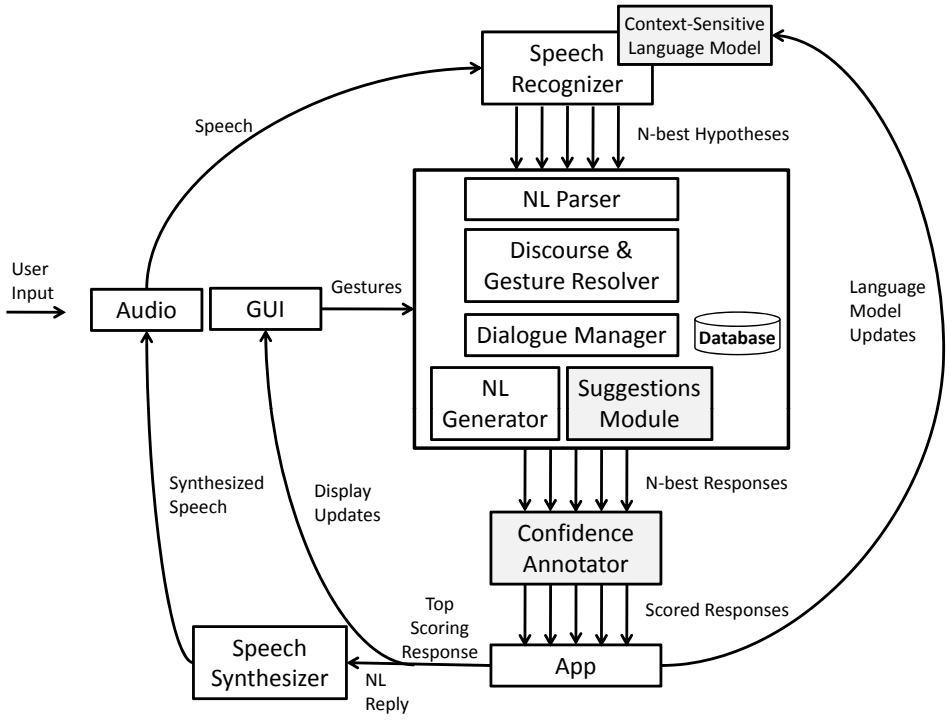
The set of candidate responses is passed to the confidence annotator, which assigns a score to each. The application then either chooses the best scoring response, or decides to choose none of them and respond by indicating non-understanding. The chosen response is then used to update the GUI and provide the speech synthesizer with a natural language utterance, which is synthesized and streamed to the user interface. Based on the response, the context-sensitive language model used by the speech recognizer is also updated.

The flow of information is enabled by the multi-tiered client-server architecture shown in Figure 2-3b. Message passing and audio streaming between the client web browser and the web server are handled by a set of components that have been generalized over time and released publicly as part of the WAMI toolkit [34], discussed in the following chapter. Built on top of this WAMI framework is the map-centric GUI, which communicates with an application controller (or "app") that is specific to *City Browser* on the server. Here, communication among the speech recognizer, natural language processing components, confidence annotator, and speech synthesizer is coordinated, producing the information flow in Figure 2-3a.

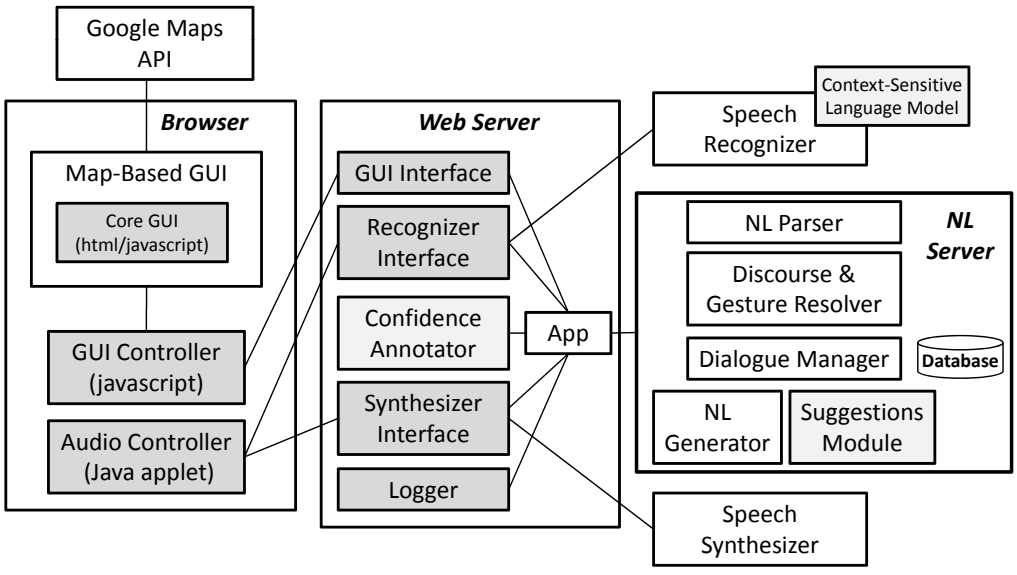
The remainder of this chapter describes each of the components in Figure 2-3.

## 2.3 Web Availability

A key technical contribution of this thesis is the development of an architecture for deploying conversational multimodal interfaces to a wide audience, via the web. The darkly-shaded components shown in Figure 2-3 are the key to realizing this vision;




(a) Information flow




(b) Web-accessible architecture

Figure 2-3: *City Browser*'s information flow and web-accessible architecture. Lightly-shaded boxes indicate thesis contributions discussed in subsequent chapters. Darkly-shaded boxes are part of the WAMI toolkit [34], which is discussed in Chapter 3

they allow *City Browser* to function as a web application, accessible via any standard web browser. These components have evolved a great deal since the first prototypes of *City Browser*, which relied on the GALAXY architecture [74] to facilitate communication among components (see Figure 4 of [38]). With the help of collaborators, the dependency on GALAXY has been eliminated, allowing support for a wider range of web-based multimodal interfaces. In addition, the architecture now functions robustly across all common web browsers and operating systems.

Taken together, the darkly-shaded components provide the "plumbing" to allow a web browser client to communicate with a speech recognizer, synthesizer, and other natural language processing components on the server. The web browser acts as a "thin" client, responsible only for providing the GUI and capturing or playing audio – all significant speech and language processing occurs on the server.

The GUI communicates with the web server via AJAX (Asynchronous Javascript And XML) methods [23], allowing applications like *City Browser* to be highly interactive because communication between client and server occurs bidirectionally in the background. WAMI hides the complexities of these message passing techniques from developers, providing an abstraction for communicating between client and server. The audio controller communicates with recognizer and synthesizer interfaces, which again hide the complexities involved with capturing, playing back, and streaming audio. These interfaces can be configured to work with various speech recognizers and synthesizers.

Though "just" plumbing, this architecture represents a departure from previous methods of deploying multimodal speech interfaces. Web browsers are ubiquitous, and the ability to provide services to, and collect usage data from, anyone with a web browser and a microphone represents a leap forward in terms of the ease of deploying multimodal speech interfaces.

Developing multimodal applications using web-standards has a number of key advantages. First and foremost, millions of users have access to, and are familiar with, web browsers. Moreover, web browsers run on many sorts of devices, including desktop and laptop computers, smartphones and PDAs, and even in automobiles. A network-delivery approach to building spoken dialogue systems means that users of all of these different types of devices can access them from anywhere with an Internet connection. Moreover, they need not run computationally demanding processes such as speech recognition when adequate computational resources are not available locally – instead, centralized servers can provide fast speech recognition and natural language understanding services. From a research perspective, collecting usage data is straightforward: users interact with the system via the comfort of their own device, while those interactions are centrally logged. Finally, developing web-based applications has the key advantage that it becomes easy to make "mash-ups" – in which content and services from multiple sources can be "mashed" together.

Given the advantages just listed, and *City Browser*'s success as a proof-of-concept, the "plumbing" used for *City Browser* was packaged into a toolkit called WAMI [34] and made available to other developers. The next chapter gives an overview WAMI. It also highlights the range of applications which have been developed with it. Through WAMI, a technical contribution first developed for *City Browser* has been able to

have a much broader impact.

## 2.4 Graphical User Interface

The *City Browser* graphical user interface communicates with the natural language processing components on the server via the WAMI architecture discussed immediately above. As shown in Figure 2-2 above, it is a map-centric interface rendered in a web browser using HTML and Javascript. It makes use of the Google Maps API [29] to provide a map on which search results are displayed. This highlights a major advantage of building web-based multimodal interfaces: they can integrate feature-rich third-party web services.

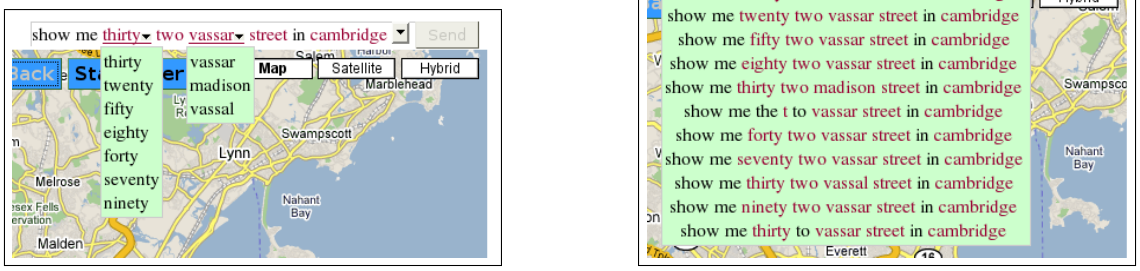
In addition to the map, the interface prominently shows the suggestions produced by the utterance-shaping module (Chapter 9) on the right-hand side. These are updated as the conversation progresses. Also shown on the right-hand side is a listing of any search results currently shown on the map. Users can click on search results to see detailed information. They may also draw on the map while speaking, using a mouse (or pen, if using a tablet PC or mobile device), after pressing the prominently displayed "click-to-talk" button at the top of the screen. Drawing can be used to circle entities shown on the map, outline regions of interest, mark points to search near, or draw lines along which to search.

### 2.4.1 Multimodal Error Correction

One of the most potentially frustrating aspects of interacting with conversational interfaces is speech recognition errors. Indeed, much of the focus of this thesis is on mechanisms for integrating contextual information to improve speech recognition and functional accuracy in such systems. Nonetheless, speech recognition errors are still quite frequent in *City Browser*.

One technique for dealing with errors in multimodal interfaces is to show speech recognition hypotheses to the user, and provide some mechanism for correcting them. While extensive research has been performed on multimodal error correction techniques for dictation systems (*e.g.*, [82]) – especially with regard to techniques that display alternative hypotheses – such alternatives-based error correction techniques have not previously been integrated into multimodal conversational interfaces.

In *City Browser*, many speech recognition errors involve confusion about semantically important entities like proper nouns (*e.g.*, restaurant names) and numbers (*e.g.*, "thirty" *vs.* "fifty"). Other conversational interface designers working in domains with large sets of proper nouns have also noted this difficulty [91]. In an effort to make it easy to correct such errors, *City Browser* includes a straightforward mechanism for alternatives-based multimodal error correction, which utilizes the fact that semantic classes are used in the speech recognizer's language model (see Chapter 6). The user interface displays a user-correctable list of recognition hypotheses in the upper-right corner of the screen, where corrections can easily be made to such semantically important words and phrases.




```
show me thirty<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me twenty<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me fifty<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me thirty<TENS> two<DIGITS> madison<STREET> street<STREET_T> in cambridge<CITY>
show me the t<SUBWAYNAME> to vassar<STREET> street<STREET_T> in cambridge<CITY>
show me forty<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me seventy<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me thirty<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me twenty<TENS> two<DIGITS> vassal<STREET> street<STREET_T> in cambridge<CITY>
show me ninety<TENS> two<DIGITS> vassar<STREET> street<STREET_T> in cambridge<CITY>
show me thirty<TENS> to vassar<STREET> street<STREET_T> in cambridge<CITY>
...
```

Figure 2-4: The correctable N-best list associated with the speech recognition hypotheses shown for the utterance *Show me thirty two Vassar Street in Cambridge*. Semantic class members are tagged in each hypothesis. The top-left image is what is flashed to the user briefly to advertise the corrections capability.

*City Browser* displays the top recognition hypothesis associated with the selected system response, and allows users to correct it in two ways. First, a drop-down menu is available that allows the user to replace the top hypothesis with any of up to 15 of the top hypotheses appearing on the N-best list. In addition, a list of alternative values for each semantic class is accumulated by examining the top 50 recognition hypotheses. If a class member appears in the top hypothesis, a drop-down menu allows the user to change the value of this class member to that of any other, and then resubmit the altered hypothesis to *City Browser* for processing. Figure 2-4 shows an N-best list generated by the speech recognizer, with semantic class members tagged, and the resulting drop-down menus that are then provided to correct the recognition result.

To advertise that the capability exists, *City Browser* briefly displays each of the available *token replacements* for 1.2 seconds as soon as they are displayed. This also allows the user a chance to see if the correct alternative appears on the list, without having to activate the drop-down list with the mouse. 58% of subjects in one user study (the *Web* corpus in Chapter 4) made use of the corrections capability at least once.

## 2.5 Natural Language Processing Pipeline

*City Browser* builds on a natural language processing pipeline that has been used previously to build a number of conversational interfaces, for example: [77, 99, 101]. These systems involve a flow of information quite similar to that in Figure 2-3a: speech recognition hypotheses are parsed, the parses are augmented with discourse context and gestures, and are then processed by a dialogue manager component that decides on the system's response. The response is then realized as natural language and graphical user interface updates. Unlike the aforementioned systems, *City Browser* incorporates a response confidence annotator, which scores *multiple* candidate responses produced from the speech recognizer's N-best list. In addition, *City Browser* incorporates a novel context-sensitive suggestions module, and a context-sensitive language model, as shown in Figure 2-3a.

### 2.5.1 Speech Recognizer

A speech recognizer takes as input a recording of a person speaking and hypothesizes what words were said. Given a particular $N$, it can typically produce a ranked list of N hypotheses of what words were spoken. *City Browser* uses the SUMMIT speech recognizer [26]. SUMMIT makes use of four probabilistic sources of information: an acoustic model, a set of phonological rules, lexical constraints, and a language model. This thesis is primarily concerned with the language model, which assigns a probability $P(W)$ to any sequence of words $W = w_1, w_2, ..., w_3$. Several context-sensitive $n$-gram language models have been developed for *City Browser*, and are discussed in Chapter 6. These language models rely on the ability of the SUMMIT speech recognizer to be able to efficiently update the list of class members in a class $n$-gram language model, a capability described in [16].

### 2.5.2 Natural Language Parser

*City Browser* uses the natural language parser TINA [76], which, given a recognition hypothesis, produces a hierarchical semantic representation. Figure 2-5 gives the semantic representation derived for the utterance *Show me Italian restaurants in Boston.* TINA uses a manually developed grammar in which a core syntactic grammar is augmented with domain-specific information. The *City Browser* grammar leverages years of work on the core grammar, as well as development for similar domains [99, 16]. TINA augments the manually-created grammar with probabilistic information inferred from language model training data.

### 2.5.3 Discourse and Gesture Resolution

The discourse and gesture resolution component used by *City Browser* is described in [20]. It uses manually created rules to integrate gestures and discourse information drawn from conversational context into the hierarchical semantic representations

```
{c request
   :pred {p show
      :topic {q a_restaurant
         :quantifier "indef"
         :pred {p pred_cuisine
            :topic "italian" }
         :pred {p in_city
            :prep "in"
            :topic {q city_name
            :name "cambridge" } } } } }
```

Figure 2-5: The TINA parser's hierarchical semantic representation for *Show me an Italian restaurant in Boston.*

| **Utterance** | **Discourse or Gesture Resolution** |
|---|---|
| *Show me restaurants in this area. [circle an area on the map]* | Match the referent *here* to the gesture outlining the area on the map. |
| *Tell me the address of this restaurant.* | Resolve the discourse referent for *this restaurant.* |
| *Give me directions from 32 Vassar Street in Cambridge* | Find a discourse referent to which to give directions. |
| *What about cheap ones?* | Given a previous utterance, such as *Show me expensive French restaurants in Boston*, augment the new semantic representation with the additional constraints represented by: *French restaurants in Boston.* |

Table 2.1: Examples of utterances where the semantic representation produced by the parser will be augmented with gestural or contextual information.

produced by the natural language parser. Table 2.1 shows several utterances which have augmented semantic representations after this stage.

### 2.5.4 Dialogue Management

The dialogue manager takes as input the context-resolved semantic representation and determines what to do in response. The representation is further simplified to a set of keys and values (*e.g.*, `cuisine=italian`), which are used by a manually created dialogue control table [75] to determine how to respond. A large portion of the functionality is generic, and has also been used in a variety of applications centered around searching databases based on constraints specified with natural language (*e.g.*, [33, 99]). Other capabilities, such as providing driving directions, are encoded through domain-specific function calls in the dialogue control table. The dialogue manager outputs both instructions to update the GUI (*e.g.*, a list of search results to display),

and a semantic representation of what should be said.

### 2.5.5 Natural Language Generation

The natural language generation component, GENESIS [5], takes the semantic representation formed by the dialogue manager and produces appropriate natural language. It is quite flexible, and can be used to produce both natural language, and constructs in artificial languages such such SQL or HTML. For *City Browser*, domain-specific template-based rules were manually constructed, which convert the dialogue manager's response both to natural language, and to XML, which is used to update the GUI.

### 2.5.6 Suggestions Module

The suggestions module produces context-sensitive utterance suggestions, which are displayed to the user on the GUI. These use contextual information to help the user understand what is appropriate to say at any given time. The module is discussed in detail in Chapter 9.

### 2.5.7 Confidence Annotator

Each of the speech recognizer's N-best hypotheses are passed to the natural language processing components discussed above, and a candidate system response is generated corresponding to each of them. The response confidence annotator assigns a score to each response. The confidence annotator is described in detail in Chapters 7 and 8.

### 2.5.8 Application

The *City Browser* application (denoted as "app" in Figure 2-3) takes the confidence-annotated responses and chooses to either respond to the user with the best scoring one, or reject all of them. If one is selected, then it is used to update the GUI, and the appropriate natural language reply is sent to the speech synthesizer. Each response also contains instructions on how to update the context-sensitive language model – see Chapter 5 – which are dispatched to the speech recognizer. If none is selected, then the system indicates to the user that the utterance was not understood by saying "pardon me".

### 2.5.9 Speech Synthesis

For the experiments described in this thesis, the DECtalk speech synthesizer [41] was used. However, *City Browser* can easily be configured to use any synthesizer.

## 2.6 Database Creation via Web Crawling

*City Browser* relies on a database of points of interest like restaurants, hotels, museums, and subway stations. For the most part, the database is populated via an automatic process of crawling websites to extract information. This is especially important for restaurants, since there are a lot of them, and each may have a good deal of useful metadata associated with it. For instance, each serves a particular type of cuisine in a particular price range, and each may have received ratings or reviews. *City Browser* harvests this data from the World Wide Web via an automated process.

To start the process, a custom-built web spider crawls information sources available on the web to get information about restaurants in a particular metropolitan area. Depending on the metropolitan area, this data may include anywhere from around 50 to 250 nearby cities, containing from 3,000 to 17,000 restaurants. The retrieved web pages are then "scraped" to create a structured database with name, address, telephone number, and so forth. To convert textual representations of a restaurant's opening hours into a structured format, the text is parsed using the methodology developed previously in [78]. This structured information makes it possible for *City Browser* to generate a natural language description of a restaurant's opening hours, which can then be spoken as synthesized speech in response to a question such as: *What are the hours of this restaurant?*.

Next, restaurant names must be "cleaned" so that they can be included in the speech recognizer's language model, and pronounced reasonably by the speech synthesizer. For example, an abbreviation like "pzzr" should be converted to "pizzeria". In addition, variations on each restaurant name are produced for the recognizer's language model, so that users can refer to restaurant in a natural manner. For example, people naturally assume that a restaurant named *Caprice Restaurant and Lounge* could be referred to as *Caprice Restaurant*, or simply *Caprice*. These variations are produced through a set of manually-defined rules. The rules appear to produce reasonable, though not perfect, results.

## 2.7 Related Work

*City Browser* has drawn inspiration, both in form and function, from a number of areas of related work. In this section, three in particular are covered. First, previous efforts to bring speech interfaces to the web are discussed. Second, several similar multimodal interfaces are compared to *City Browser*. Third, more recent work focusing on making multimodal speech applications widely available via mobile devices is covered.

### 2.7.1 Web-Based Speech Interfaces

There have been two other notable lines of work toward making speech interfaces widely available via the web. One effort involves coupling the telephone with a graphical user interface provided via the web. This was a successful proof-of-concept, but

was never made available to the public, and offered more of a static information display than a rich multimodal interface. The other line of work involves developing a speech markup language, similar to HTML, to speech-enable web pages. Both approaches are described below.

### Telephony Dialogue Systems on the Web

WebGALAXY [56] allowed users to visit a web site and then make a telephone call to a spoken dialogue system which provided weather information. The system's responses to user queries were spoken over the phone, with the corresponding information also displayed on the web page. Essentially, WebGALAXY augmented traditional telephony dialogue systems by providing a means for displaying information to the user at the same time on their computer screen, with some very limited GUI interaction.

The WAMI platform discussed in the next chapter subsumes the capabilities of WebGALAXY – indeed, the new platform has been used to make a multimodal version of the same weather information system [33] accessed via WebGALAXY. Simply due to advances in web technology over the last decade, WAMI provides for the capability to build much richer multimodal interfaces than WebGALAXY. Moreover, *City Browser*, and other WAMI applications, have been deployed publicly – unlike WebGALAXY – and have been used to collect significant amounts of usage data. Moreover, while WAMI can be used with the GALAXY architecture [74], is not tied to it, making it more widely useful.

### Speech Markup Languages

More recently, efforts have been undertaken to develop standards for deploying multimodal interfaces to web browsers, typically entailing the use of Voice over IP (VoIP) or speech recognition on the client machine. The two major standards under development, Speech Application Language Tags (SALT) [89] and XHTML+Voice (X+V) [4] are both aimed at augmenting the existing HTML standard. The intent of these specifications is that, just as current web browsers render a GUI described via HTML, browsers of the future would also provide a speech interface based on this augmented HTML. As such, interfaces implemented using either of these specifications require the use of a special web browser which supports the standard.

SALT and X+V are designed mainly to enable the use of speech to fill in the field values of traditional HTML forms. X+V, for example, integrates the capabilities of VoiceXML into the browser, so that interactions typically consist of telephony-like mixed initiative dialogues to fill in field values. Related multimodal markup languages, such as those in [53] and [46], generally attempt to make designing these sorts of interactions more straightforward.

Unlike with SALT and X+V, applications like *City Browser* produced using WAMI do not require the use of special web browsers; instead, they can be accessed from any modern browser. Furthermore, while these markup languages are geared toward form-filling, the architecture presented in this chapter allows for a much wider range of speech interfaces – one subset of which may be form filling applications.

### 2.7.2 Multimodal Conversational Interfaces

There is a large body of work on multimodal interfaces which incorporate speech, and other modalities such as gestures made by drawing, the hand, or the face. The discussion here is restricted to conversational multimodal interfaces that provide similar functionality to *City Browser*. Of particular note is VOYAGER [99], which provided a map-based multimodal interface much like *City Browser*. Indeed, *City Browser* makes use of much the same natural language processing pipeline, and makes use of a gesture recognizer developed for VOYAGER.

Another similar multimodal interface is MATCH [49], which provides extensive multimodal capabilities for accessing urban information. There is significant overlap between *City Browser* and MATCH. For instance, both provide multimodal access to restaurant and public transit information. A major feature of the MATCH system that *City Browser* lacks is handwriting recognition, as it is not assumed that *City Browser* users will have access to a pen-based interface.

Another map-centric multimodal conversational interface is AdApt [40]. AdApt provides information about apartments for rent in downtown Stockholm. Unlike the other interfaces discussed, AdApt includes an animated talking face, whose movements are coordinated with synthesized speech.

*City Browser* stands out in that it provides support for large databases containing thousands of entries, extending throughout a metropolitan area; in particular, the restaurant databases are comparable in size to those of commercially available, web-based city guides. Moreover, *City Browser* supports a multitude of metropolitan areas, rather than just one or two cities. Also, *City Browser* provides driving directions and supports the recognition of arbitrary addresses with any street name in the metropolitan area.

Finally, as has been noted, *City Browser* is unique in that it is built using web technologies, and has been deployed publicly. This has made it possible to collect interaction data from users all over the world, from the comfort of their own homes. None of the conversational multimodal interfaces described here have been deployed as widely.

### 2.7.3 Widely Available Multimodal Speech Interfaces

A few speech-enabled multimodal interfaces have become available to a large audience as mobile-device applications quite recently [97, 30, 80]. These interfaces allow users to fill in a single input field – the contents of which are typically sent to a web search engine – by speaking. The search results are then presented visually. These applications demonstrate the great potential of multimodal interfaces: while it may often be easier for users to speak a search query than type it, often it is more convenient to browse through the results visually. Moreover, by using language models with a very large vocabulary, two of these applications ([97, 30]) are able to accurately transcribe a huge variety of search queries. Conversational interfaces like *City Browser*, on the other hand, can understand a much smaller number of words, and provide information about much more limited domains.

However, these interfaces do not understand natural language, nor are they conversational. Users may search again by speaking, but they can't, for example, carry on a conversation to find out more about one of the search results. Moreover, these interfaces are not available on a standard web browser – they are native applications for mobile devices.

## 2.8 Summary

This chapter has outlined the architectural structure of the *City Browser* interface, both conceptually in terms of information flow, as well as topologically. The web-based architecture of *City Browser* is a major technical contribution of this thesis, as it serves as a proof of concept that rich multimodal conversational interfaces can be made available widely via the web.

From an information processing point of view, *City Browser* relies on a pipeline of speech and natural language processing components that have previously been deployed in a number of conversational interfaces. Each has been customized for the domain. Moreover, the standard processing pipeline has been augmented with a context-sensitive language module and suggestions generator, as well as modified to support the generation and scoring of multiple candidate system responses. Each represents a contribution of this thesis.

# Chapter 3

# The WAMI Toolkit and Example Applications

As was noted in the previous chapter, one of the outgrowths of the effort to make *City Browser* available via the World Wide Web has been the development of the WAMI Toolkit [34]. WAMI makes general the "plumbing" developed for *City Browser*, so that other developers can build similar applications. The goal of the toolkit is to provide a framework in which developers can create a wide variety of web-accessible multimodal interfaces. The great majority of the work reported in this chapter has been collaborative, and most of the applications discussed here were developed by others. As such, this chapter serves as a report on the broad impact of the web-based architecture developed for *City Browser*, rather than as a description of a direct contribution of this thesis.

## 3.1 Toolkit Configurations

The WAMI Toolkit is intended to be used primarily in one of two distinct configurations. First, it can be configured to work with any speech recognizer and synthesizer, allowing speech researchers to build web-accessible multimodal interfaces using existing tools. Second, for the millions of web developers who are not speech experts, and do not have easy access to a speech recognizer, it can be configured to use speech services provided by the WAMI Portal. The WAMI Portal is a service hosted at MIT that allows any web developer to stream audio to the SUMMIT speech recognizer [26] and obtain speech recognition hypotheses. The versatility of the WAMI toolkit makes it attractive both to speech experts, and to web developers who have no experience with speech technology.

### 3.1.1 Toolkit-Only

In the "toolkit-only" configuration, the WAMI toolkit is used to provide the "plumbing" to connect a speech recognizer, synthesizer, application logic, and a web-based GUI. This configuration is shown in Figure 3-1. Specifically, the WAMI toolkit pro-

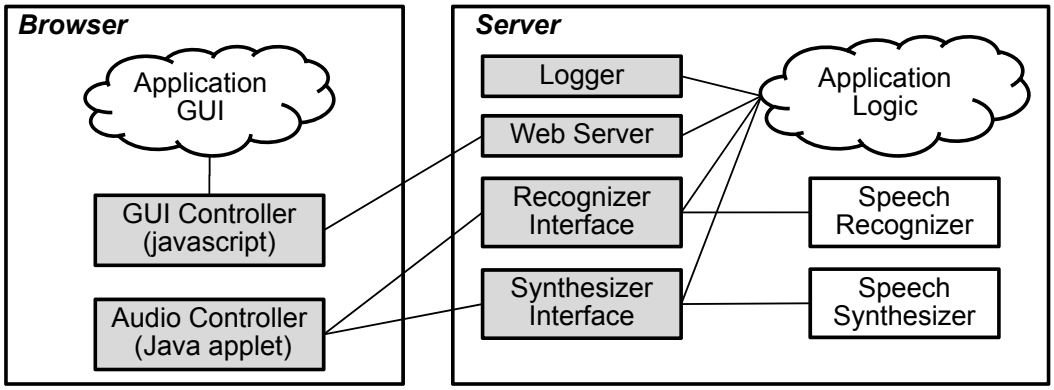



Figure 3-1: "Toolkit-only" configuration in which a developer provides a speech recognizer and/or synthesizer, application logic, and a web-based GUI. The WAMI Toolkit components, which are shaded, provide the "plumbing" connecting them.

vides infrastructure to capture speech and send the audio to the server, where it is passed to the speech recognizer. Recognition hypotheses are passed to the developer-provided application logic, which decides how to respond. It can send messages to the application's web-based GUI via methods provided by WAMI, and WAMI will play any audio provided by the speech synthesizer. This configuration evolved directly from the *City Browser* system, and is how *City Browser* is currently constructed – as is evident in Figure 2-3b in the previous chapter. This configuration works well for conversational interface designers who have access to existing speech recognition technology, and who wish to integrate complex natural language processing components.

### 3.1.2 Toolkit+Portal

In the "toolkit+portal" configuration, developers do not need to provide a speech recognizer or synthesizer. Instead, they can make use of speech services hosted by servers at MIT via the WAMI Portal. Using this method, they do not have to worry about installing and configuring speech recognition or synthesis technology. There are two major advantages to providing speech services in this way. First, developers can build applications with the WAMI toolkit without any prior experience with speech technology, and need not worry about installing and configuring a speech recognizer or synthesizer. This means that developers can begin writing applications in minutes. Second, because speech recognition is provided by servers hosted at MIT, all incoming audio can be logged for research purposes. Speech data is extremely valuable to researchers, as accurate speech recognition requires large amounts of labeled training data. Moreover, data gathered from web-based interfaces in one application domain should be useful for improving ones in other domains, as noise conditions and microphones should be similar.

The WAMI Portal can be accessed in two ways. First, developers can build applications that interact with the Portal via server-side application logic, as depicted in

Figure 3-2a. Second, developers can integrate the speech services directly into a web page as a "mash-up", as diagrammed in Figure 3-2b. The "mash-up" configuration is appealing, as it makes it extremely simple for any web developer to quickly add speech capabilities to an existing web page, or to develop a new speech application. Developers simply include a single Javascript file hosted on the WAMI Portal, and they can then use Javascript to specify the speech recognizer's grammar, obtain recognition hypotheses, and request speech synthesis. In this way, any developer capable of making a web page can also add speech capabilities to it with a grammar, and just a few additional lines of HTML and Javascript. Figure 3-3 shows a screenshot of, and source code for, an extremely simple application built in this manner which parrots back to the user what it hears.

In either case, developers must specify a JSGF (Java Speech Grammar Format [48]) grammar to be used as the speech recognizer's language model. The grammar may be customized for each interaction, and can be changed as the interaction proceeds. Moreover, if written in a standard way, it can be integrated with the lightweight semantic understanding module described in the next section.

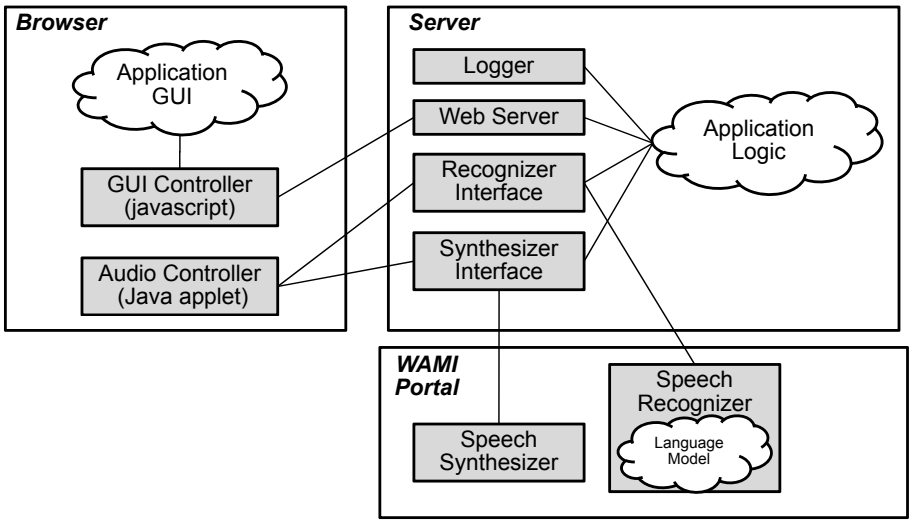
## 3.2 Lightweight Semantic Understanding

Many multimodal interfaces make use of natural language parsers to obtain a syntactic and/or semantic understanding of what a user said. *City Browser*, for instance uses the TINA parser [76] to obtain a mixed syntactic and semantic representation of the recognition hypothesis, as well as a context resolution component [20], which integrates contextual information into the meaning representation. Using natural language understanding components like these often requires a good deal of expertise. However, without such tools, it can be difficult to extract a meaning representation from a speech recognition hypothesis.

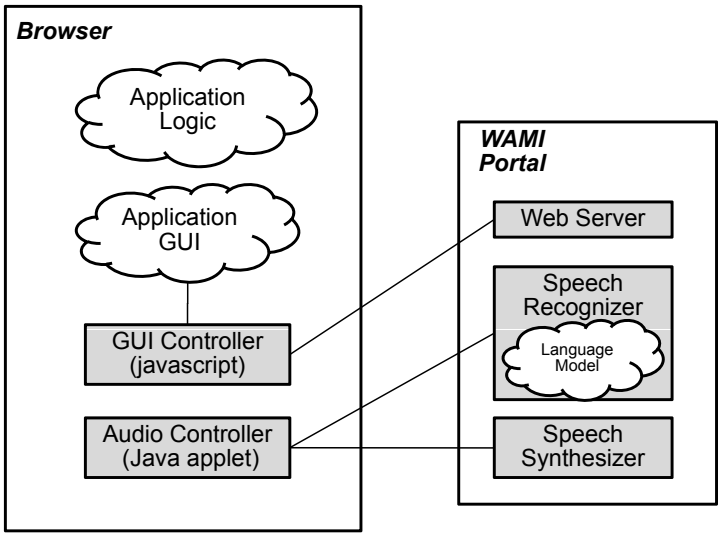
To simplify application development, WAMI provides a lightweight semantic understanding module. Developers need not use the module; however, it provides a simple framework suitable for developing a wide variety of applications. To make use of the module, a developer simply embeds semantic "tags" in the application's JSGF grammar in a standard way. The tags specify a set of semantic slots and their values, which will be associated with each recognition hypothesis. Figure 3-4 shows an example grammar for a hypothetical WAMI application. The grammar includes semantic slots such as `command`, `color`, and `size`. Each slot can take several different values.

### 3.2.1 Incremental Understanding

Since the lightweight semantic understanding module relies on small grammars that output sequences of slot/value pairs, speech recognition and natural language understanding processing often occur rapidly, in near real time. Because of this low overhead, the WAMI Portal is able to provide incremental speech recognition and natural language understanding capabilities. This means that, as the user speaks,

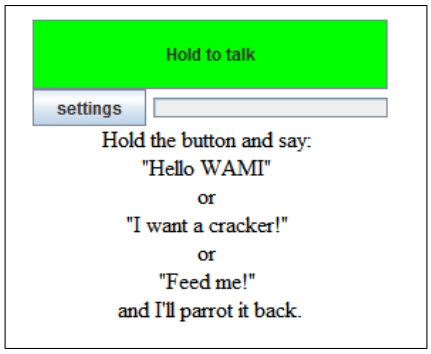



(a)




(b)

Figure 3-2: WAMI Toolkit+Portal configurations in which speech recognition and/or synthesis is provided to developers via the WAMI Portal, which is hosted at MIT. (a) Applications can access the WAMI Portal via server-side application logic, or (b) "mash-ups" can be created in which speech services are integrated directly into web pages. In both diagrams, WAMI-provided components are shaded, while developer-provided components are indicated as white clouds.




(a)

```
<html><head><title>WAMI Parrot</title>
<script src="http://wami.csail.mit.edu:8080/portal/wami.js?devKey=123">
</script><script>
var myWamiApp;
function onLoad() {
  //audio button goes here
  var audioDiv = document.getElementById('AudioContainer');
  var grammar = { "language" : "en-us",
                  "grammar" : "#JSGF V1.0;\ngrammar example;\n" +
                              "public <top> = " +
                                 hello wami | i want a cracker | feed me;" } ;
  var audioOptions = { "pollForAudio" : true } ;
  var configOptions = { "sendIncrementalResults" : false,
                        "sendAggregates" : false } ;
  var handlers = { "onRecognitionResult" : onRecognitionResult};
  myWamiApp = new WamiApp(audioDiv, handlers, "json", audioOptions,
                          configOptions, grammar);
}
//handle recognition results
function onRecognitionResult(result) {
  var hyp = result.hyps[0].text;
  myWamiApp.replayLastRecording(); //play back the audio
  myWamiApp.speak(hyp); //Text-to-speech what you heard
  alert("You said: '" + hyp + "'");
}
</script></head>

<body onload="onLoad()">
<center><div id="AudioContainer"></div>
Hold the button and say:
<br>"Hello WAMI" <br> or <br> "I want a cracker!" <br> or <br>
"Feed me!" <br>and I'll parrot it back.</center>
```

(b)

Figure 3-3: (a) Screenshot of, and (b) complete source code for, a simple WAMI "mash-up". The code shown can be deployed to a web server to create a working application. In this "parrot" application, the user holds the button to speak and can say one of three simple phrases. The recorded audio is played back, followed by synthesized speech of the top recognition hypothesis.

```
<command>    = <select> | <drop> | <make> ;
<select>     = (select | choose) {[command=select]} <shape_desc>+ ;
<shape_desc> = [the] <shape_attr>+ ;
<shape_attr> = <size> | <color> | <shape_type> ;
<size>       = <size_adj> [sized] ;
<size_adj>   = (small | tiny) {[size=small]} | medium {[size=medium]} |
               (large | big)  {[size=large]} ;
<color>      = red {[color=red]} | green {[color=green]} | blue {[color=blue]} ;
<shape_type> = (rectangle | bar) {[shape_type=rectangle]} |
               square            {[shape_type=square]} |
               circle {[shape_type=circle]} |
               triangle {[shape_type=triangle]} ;
...
```

| *select* | *the* | *small* | *red* | *rectangle* |
|---|---|---|---|---|
| [command=select] | | [size=small] | [color=red] | [shape=rectangle] |

Figure 3-4: Sample JSGF grammar snippet for a WAMI application that makes use of a lightweight semantic understanding module, and an example utterance with associated slot/value output.

the system's best guess as to what has been said so far can be periodically provided to the application. Because the small grammars typically used provide strong constraints, the incremental hypotheses tend to be accurate, especially if the "bleeding edge" is ignored, and so long as the user's utterance remains within the confines of the grammar. This means that, while a user is speaking, slots and their values can be emitted as soon as the partial utterance is unambiguous. The grammar in Figure 3-4 is written in just such a way: slots and values are assigned as soon as possible. This means that the slot/value pairs shown with the example utterance in that figure are emitted as soon as the user finishes speaking the word shown above each pair. WAMI's lightweight semantic module provides a framework for accumulating these slots and values as users speak, making it easy to provide feedback even as the user might string together multiple commands in a single utterance.

Incremental speech recognition hypotheses can be used to provide incremental language understanding and immediate graphical feedback, which has been shown to improve the user experience and system accuracy [3]. For instance, an application making use of the grammar in Figure 3-4 could highlight the set of appropriate shapes as each attribute is spoken. Indeed, careful readers will note that the grammar is constructed so as to support false starts and repetitions, since visual feedback may prompt users to correct utterances as they speak. For instance, an in-grammar utterance with a false start would be: *select the small . . . the large red rectangle.* Such an utterance might arise because, as soon as the user sees graphical confirmation of "small", he might realize he actually meant to say "large" and correct himself mid-utterance.

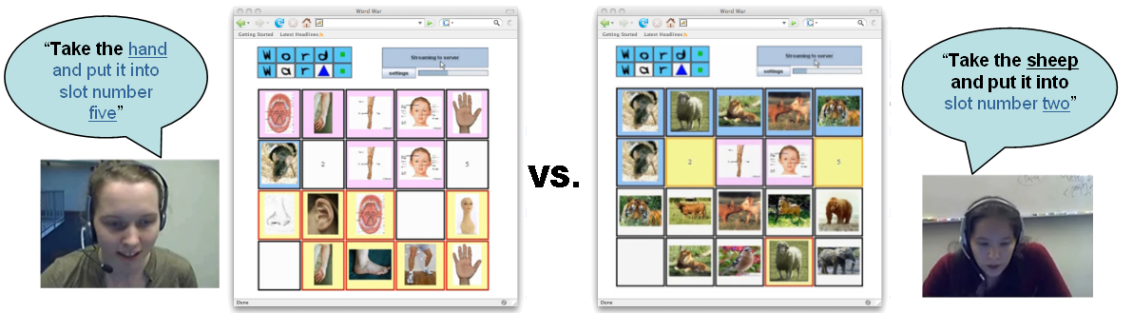



Figure 3-5: The *Word War* game.

## 3.3 WAMI Applications

A toolkit like WAMI is only useful inasmuch as it is actually used to build applications. Over the last several years, the Spoken Language Systems (SLS) group at MIT has developed a number of applications using the various versions of the toolkit that iteratively evolved over time. Recently, the toolkit and portal were made publicly available. Since then, developers outside the group have created a number of multimodal interfaces as well. This section describes some of the applications.

### 3.3.1 SLS Applications

This section outlines some of the applications which have been built over the last several years by graduate students and researchers in the Spoken Language Systems group at MIT.

#### Word War

*Word War* is a single or multiplayer game that students learning English or Mandarin Chinese can use to memorize and practice speaking vocabulary words [62]. Players create flashcards that associate images with words or phrases they wish to study. Images from their flashcards are then used to populate the game grid, shown in Figure 3-5. Players speak commands to move images from the bottom of the grid to align them with images in the top row, by placing each image in a numbered slot. While simple for a native speaker, this can be a challenging and fun task in a non-native language. A JSGF grammar suitable for WAMI's lightweight semantic understanding module is constructed dynamically for each game.

*Word War* makes extensive use of the incremental speech recognition capabilities discussed in the previous section. As players speak, the game gives graphical feedback about what has been understood so far. Figure 3-5 demonstrates this incremental feedback by showing what the game screen looks like for each player mid-utterance. For example, the player on the right has so far uttered:

*Take the sheep and put it into...*

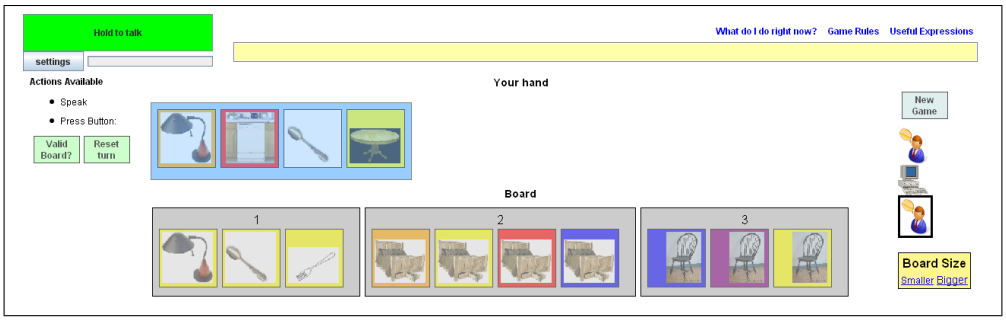



Figure 3-6: The *Rainbow Rummy* game.

As a result of this partial utterance, the image of the sheep has been outlined in red and the two numbered squares available into which it can be dropped have been highlighted. Such incremental feedback allows for rapid game play, and mid-utterance repairs.

*Word War* has been evaluated via a web-based user study involving 20 users [62]. It is currently publicly available and attracts users from around the world.

### Rainbow Rummy

*Rainbow Rummy* [98], like *Word War*, is a game that uses an association between images and vocabulary words to help students learning English or Mandarin Chinese practice vocabulary by speaking. A screenshot is shown in Figure 3-6. The game rules are similar to those of the popular games gin rummy or Maj Jong, so *Rainbow Rummy* offers an addictive and complex interaction. Players not only must be able to speak in order to play during their turn, but they must be able to follow spoken instructions when it is the computer's turn. In this way, they practice both speaking and comprehension.

### Translation Games

WAMI has been used to develop another group of games for students of English or Mandarin Chinese. In these games, the goal is to engage in a conversation by translating sentences. Games have been developed which center around discussing hobbies [11] and travel [96]. The games both provide a fun environment for learners to practice, and data for researchers working on the challenging problem of speech-to-speech translation. Figure 3-7 shows a screenshot of the travel-domain translation game.

### Flight Browser

*Flight Browser* [61] is a multimodal version of the telephone-based MERCURY conversational interface for making flight reservations [77]. Beyond using the visual display for showing search results, *Flight Browser* also explores using it both for providing incremental feedback as the user speaks, and to provide a simple mechanism by which

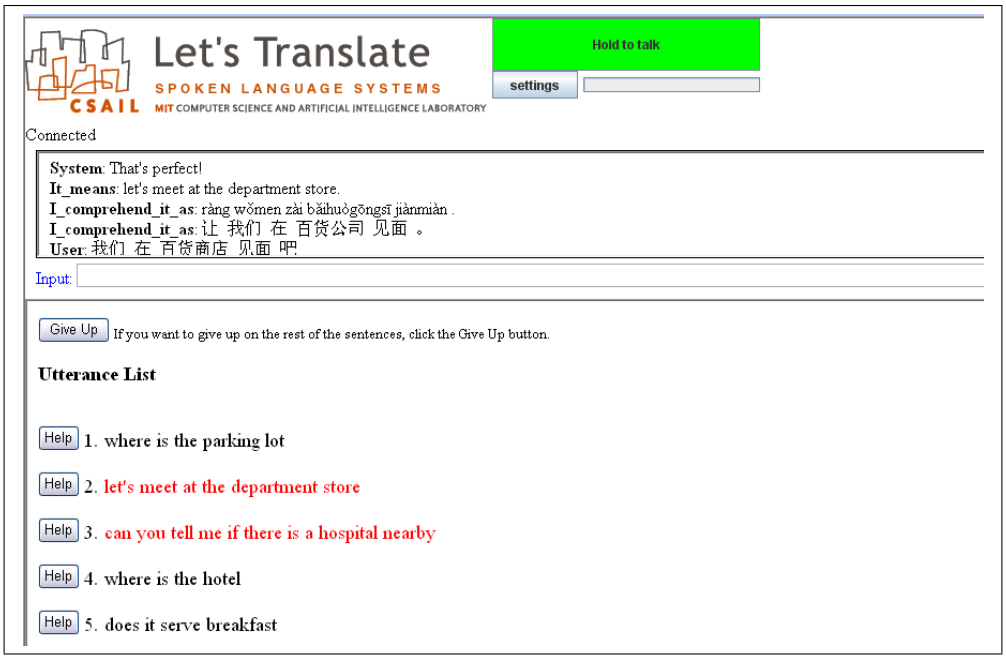



Figure 3-7: A translation game where students translate sentences related to travel.

users can correct semantic misunderstandings. Figure 3-8 shows a screenshot of the interface. As users speak, values for semantic slots such as *destination* and *airline* are displayed on the left-hand side. Users can type to correct these concepts, or they can click on one and then speak to change its value.

#### Multimodal Home Entertainment Interface

The WAMI Toolkit has also been used to build a multimodal interface to a home entertainment system [33], pictured in Figure 3-9. A GUI, built using WAMI, is displayed on a television screen. Users speak to the system with, and can view a smaller GUI on, a smart phone. This interface demonstrates the potential versatility of WAMI, as it can be used to build interfaces accessible via devices like mobile phones and televisions. Current work on the toolkit is focused on making multimodal web-based applications developed with WAMI accessible from a range of popular mobile devices.

### 3.3.2 Student Applications

The applications discussed above were developed by experts in speech technology. However, one important design goal of WAMI has been to make it easier for non-experts to develop multimodal applications using speech. Several WAMI applications have been designed as class projects by MIT undergraduate students since the toolkit was made publicly available. This section describes several of them.

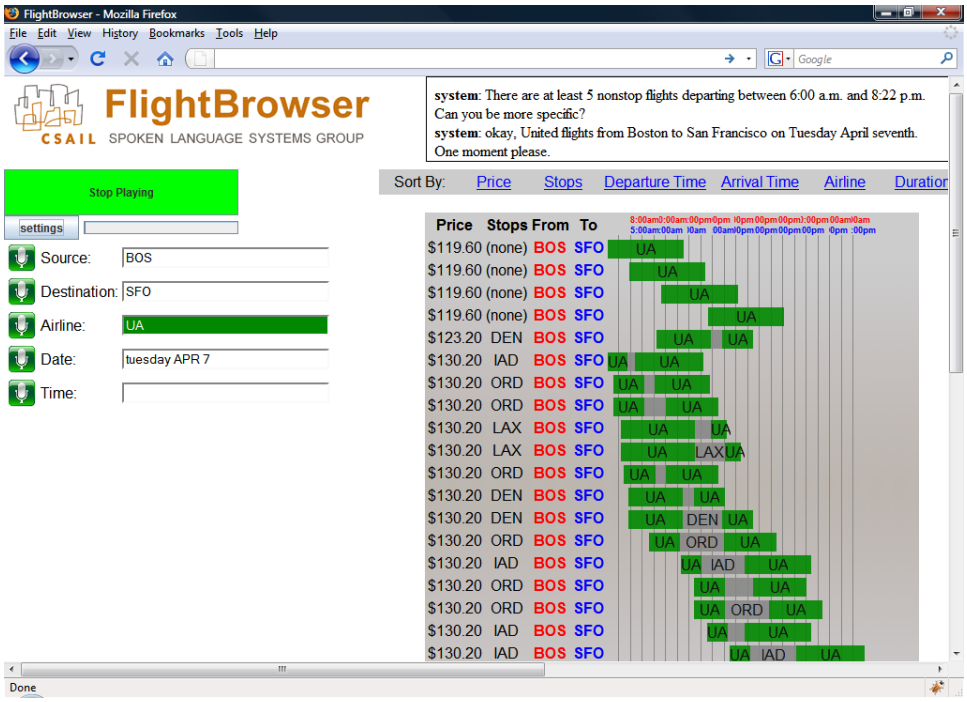



Figure 3-8: *Flight Browser*, a multimodal conversational interface for booking flights.


Figure 3-9: A conversational interface to a home entertainment system.

### Voice Race and Voice Scatter

The *Voice Race* and *Voice Scatter* games were designed by Andrew Sutherland. Both are games that elicit speech to help players memorize flashcards with arbitrary content. Each flashcard contains a *term* and *definition*. In *Voice Scatter*, shown in Figure 3-10a, terms and definitions from a set of flashcards are "scattered" randomly across the screen and players must speak to match the correct ones together. Correctly matched pairs collide and then explode. In *Voice Race*, shown in Figure 3-10b, definitions fly from left to right across the screen, and players must say the correct term in order to "hit" a definition.

Both games have been made available on Quizlet [69], a popular flashcard website where users can make and share sets of flashcards. Quizlet has 420,000 registered users who have created over 875,000 sets of flashcards covering diverse topics such as vocabulary words, scientific terms, mathematical concepts, and historical facts. Quizlet users can play *Voice Scatter* and *Voice Race* using any set of English flashcards available on the site.

During the first month of availability on Quizlet.com, the games were played by thousands of users and generated over 100,000 utterances. Given these usage levels, they appear to be successful in providing a fun way to study flashcards. Moreover, the experience illustrates the power of making speech technology available to web developers to "mash-up" with their existing sites.

### Google AJAX API Mash-up

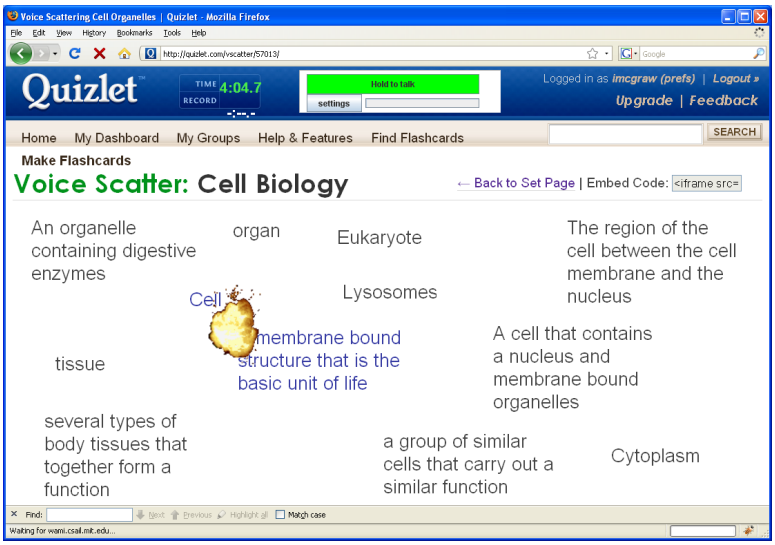
Edgar M. Salazar created a mash-up by combining WAMI with several Google AJAX APIs [28]. His application, shown in Figure 3-11, provides an interface where users speak to search for points of interest like restaurants and hotels on a map. Users can also request driving directions between several major cities, and can access Google "street view" photos.

### Media Enclave

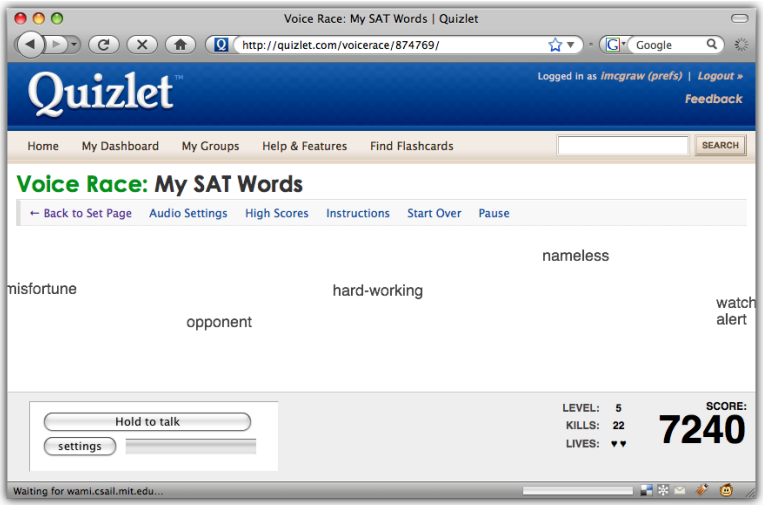
Erica Cooper used WAMI to integrate speech recognition capabilities into an existing web-based collaborative media manager called *Media Enclave* [64]. Users can speak to search a database of songs, and can enqueue ones that they want to hear. The application is intended to be used as a kiosk, or, in an audio-only configuration, via the telephone. A screenshot is shown in Figure 3-12.

### ESPN 1-Click

Rick Mancuso and Ryan MacDowell designed an application called *ESPN 1-Click*, which adds spoken shortcuts to make it easier to navigate the popular sports website. A screenshot is shown in Figure 3-13. Users can use speech to find out the latest scores or obtain the schedule of upcoming games for their favorite teams.




(a) Voice Scatter




(b) Voice Race

Figure 3-10: *Voice Race* and *Voice Scatter* games. They are publicly available at `http://quizlet.com`.

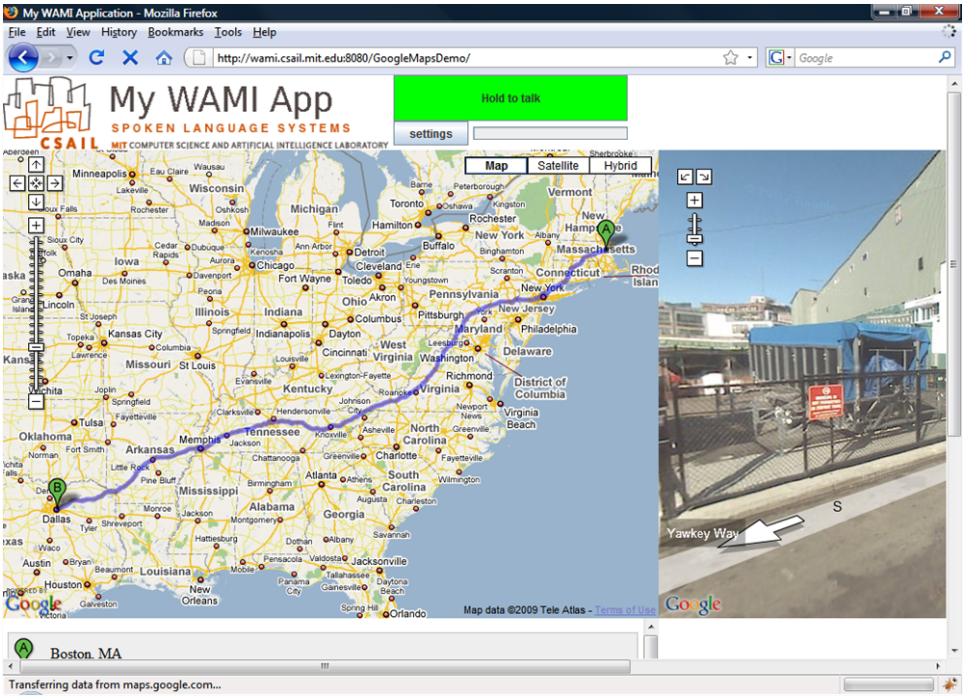



Figure 3-11: A WAMI mash-up with several Google AJAX APIs.

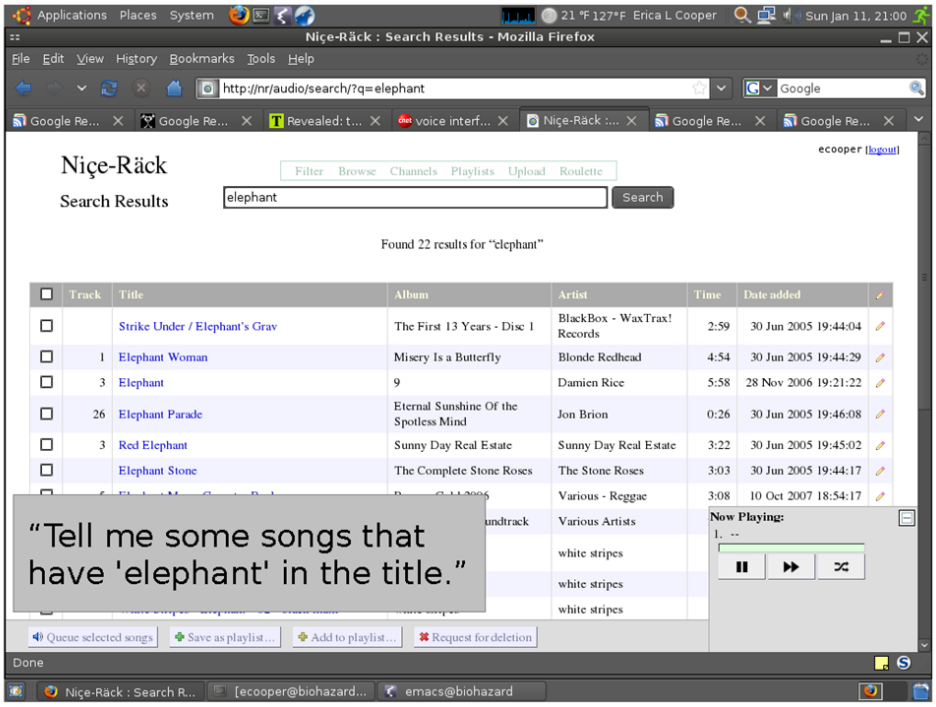



Figure 3-12: Speech interface to *Media Enclave*.

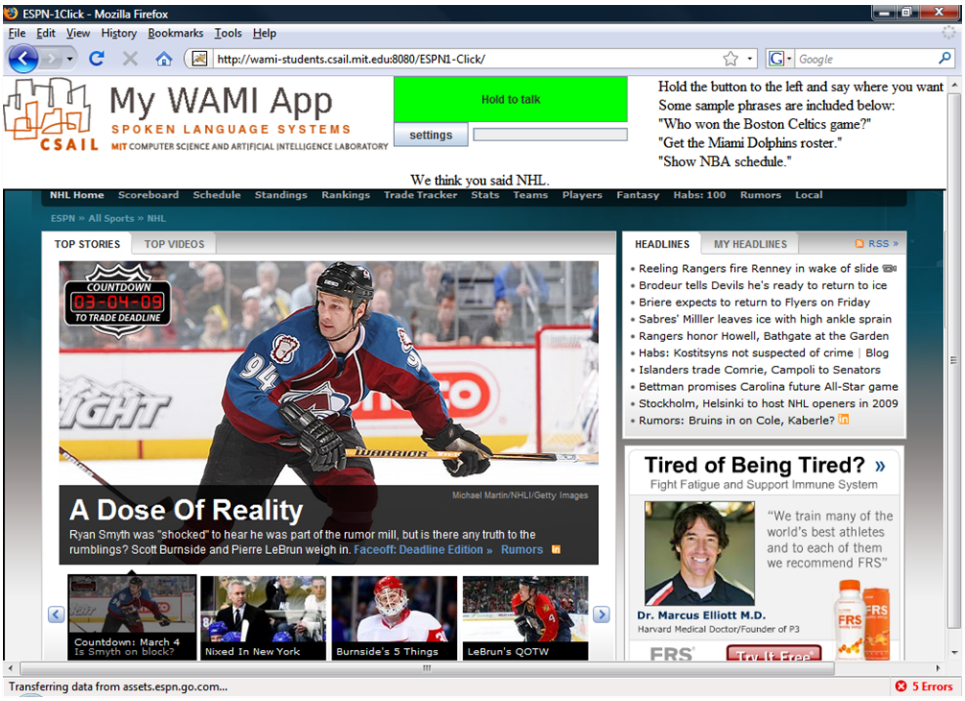



Figure 3-13: The *ESPN 1-click* application.

## 3.4 Conclusion

This chapter provided an overview of the WAMI toolkit and the associated WAMI Portal, which have grown out of the "plumbing" originally developed to make *City Browser* available as a multimodal Web application. The goal in developing the toolkit and portal has been to make it easy to build multimodal interfaces available via the World Wide Web. Millions of developers are familiar with developing web applications, and it is straightforward to make them available to large numbers of users. These two properties make the WAMI toolkit an appealing way to build multimodal applications.

WAMI has been used to create a number of applications over the last several years. They cover a range of genres and domains. While many were built by members of the Spoken Language Systems group at MIT, a number were developed independently by MIT undergraduate students. Moreover, applications developed using the toolkit have so far been used to collect over 100,000 utterances. The toolkit has been publicly available for about 7 months, and as it continues to mature it will hopefully be used to create new applications that are useful, educational, and/or fun.

# Chapter 4

# Corpora

> Computer, I know of this Japanese restaurant - don't know how to say the name of it – but I know it's uh in Brookline - can you give me um restaurants, Japanese restaurants in Brookline?
>
> Don't you speak English? Let's try again. Chelmsford. Chelmsford Garlic Bistro restaurant.
>
> –Experimental subjects speaking to *City Browser*

Several data collection efforts were undertaken as part of this thesis in which subjects interacted with *City Browser*. The aims of each collection effort were three-fold. First – and most narrowly – data and observations were used to improve core capabilities specific to *City Browser*: the speech recognition language model and parser coverage were expanded, context resolution rules were tweaked, dialogue processing rules were modified, natural language generation rules were improved, the user interface was refined, and so forth. Second, the context-sensitive modules designed to improved usability discussed in this thesis – which aim to be general purpose across a range of multimodal interfaces – were proposed and developed based on observations and offline data analysis, and then validated as part of live system deployments in subsequent data collection efforts. Third – and most broadly – collected data and annotations will be packaged to be shared with the community, in the hope that the data will be useful to other researchers in the field.

This chapter proceeds in three parts. Section 4.1 describes the corpora collected as part of this thesis. In Section 4.2, the collected corpora are compared to those from similar conversational interfaces. Finally, Section 4.3 concludes with a discussion of the tools created to transcribe and annotate the collected corpora.

## 4.1 Overview of Corpora

This section describes in detail three data collection efforts, resulting in the collection of four distinct corpora:

1. The *Tablet* corpus was collected from 10 subjects making use of *City Browser*

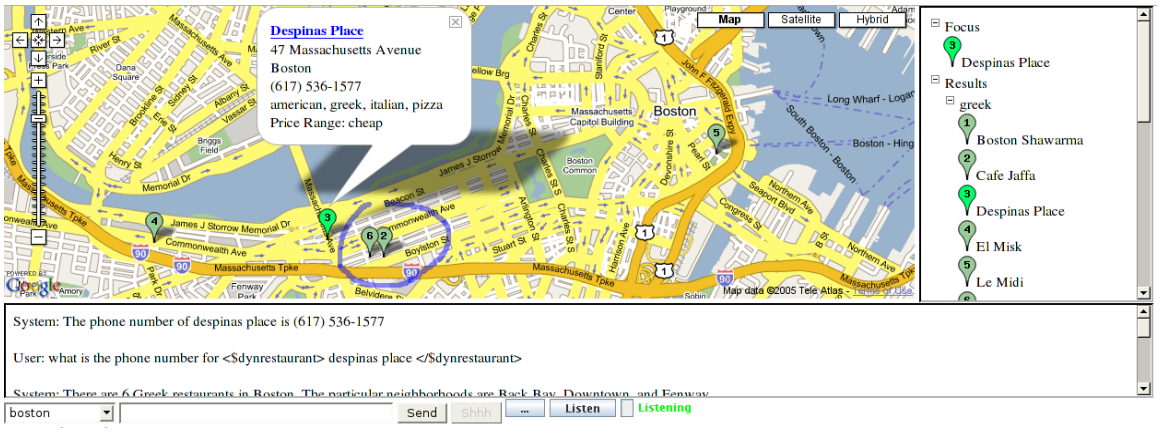



Figure 4-1: Screenshot of the *City Browser* interface, as it appeared during collection of the *Tablet* corpus.

on a tablet computer; each subject was tasked with finding four appealing restaurants.

2. The *Web* corpus includes data from 38 subjects, each performing 11 tasks using his or her own web browser.

3. The *Car* and *Car-Pilot* corpora together include 125 subjects, each of whom performed a series of tasks from the driver's seat of a BMW sedan using an integrated version of *City Browser*. Each subject's heart rate, skin conductance level, and respiration rate were recorded during the experiment.

Table 4.1 provides a high-level overview of the four corpora, which altogether comprise 7,827 utterances collected from 173 subjects.

### 4.1.1 *Tablet* Corpus

The *Tablet* corpus was collected in 2006, using an early prototype of the *City Browser* system which provided access to a database of restaurants only. 10 subjects were recruited from the MIT community and brought into an informal setting in the lab. They used an IBM Thinkpad X41 tablet computer and a headset microphone to complete an open-ended task in which they were asked to find four restaurants – two in the Boston metropolitan area, and two in any of the other metropolitan areas in the *City Browser* database. Subjects were given a brief demonstration of the interface, and then interacted with *City Browser* while their interactions were observed. Each was compensated with a $10 gift certificate. 9 of the 10 subjects were able to complete the study successfully, although some required additional hints on how to operate the interface and phrase utterances. Figure 4-1 shows a screenshot of the interface used by the subjects in this experiment.

A total of 546 utterances were recorded and transcribed. The transcripts show that the system operated with a word error rate of 27.4%. Excluding all proper nouns, the core vocabulary size of the class trigram language model used was 1,050 words.

| | **Tablet** | **Web** | **Car-Pilot** | **Car** |
|---|---|---|---|---|
| **Basic Facts** | | | | |
| Date | 2006 | 2007 | 2008/2009 | 2009 |
| Number of Subjects | 10 | 38† | 33 | 92 |
| Tasks per subject | 4 | 11 | 12 | 10 |
| User utterances | 546 | 2,138† | 1,651 | 3,492 |
| Compensation | $10 | $20 | None | $40 |
| Location | In Lab | Via the Web | In Car | In Car |
| "Frozen" development | Yes | Yes | No | Yes |
| **Logged Data** | | | | |
| Utterances | ✓ | ✓ | ✓ | ✓ |
| Dialogue State | Partial | ✓ | ✓ | ✓ |
| Audio/Video Recordings | | | Some subjects | ✓ |
| Physiological Measures | | | Some subjects | ✓ |
| Survey | | ✓ | Some subjects | ✓ |
| **Annotations** | | | | |
| Transcripts | ✓ | ✓ | ✓ | ✓ |
| Response Correctness | | ✓ | ✓ | ✓ |
| N-best Response Correctness | | | ✓ | ✓ |
| **Accuracy Measures** | | | | |
| WER (top 1) | 27.4% | 29.2%† | 31.9% | 36.1% |
| Response Error Rate | * | 52.2% | 44.0% | 46.9% |
| ***City Browser* Capabilities** | | | | |
| Restaurants Database | ✓ | ✓ | ✓ | ✓ |
| Museums Database | | ✓ | ✓ | ✓ |
| Subway Database | | ✓ | ✓ | ✓ |
| Hotels Database | | | ✓ | ✓ |
| Driving Directions | | ✓ | ✓ | ✓ |
| Drawing/Gesture | ✓ | ✓ | | |
| Current GPS Location | | | ✓ | ✓ |
| Two-pass speech recognition | ✓ | ✓ | | |
| Context-sensitive LM | | ✓ | ✓ | ✓ |
| Utterance Suggestions | | ✓ | ✓ | ✓ |
| Correctable N-best | | ✓ | | |
| Response Confidence | | | ✓ | ✓ |

†Subset of subjects who successfully completed experiment

Table 4.1: Overview of *City Browser* corpora. The four corpora were collected in three different environments: in the lab on a tablet computer, via the web, and while seated in a parked car. Different versions of *City Browser* were deployed in each environment, and thus the subjects in the different studies encountered different sets of system capabilities.

However, depending on the chosen metropolitan area, the number of proper nouns in the language model ranged from a few thousand up to about 30,000.

Utterances are associated with system debug logs, allowing for a partial reconstruction of *City Browser's* internal state at the time of each utterance.

The data collection effort had two primary results. First, it was used to gather language modeling training data. Transcribed utterances were incorporated into the language model training set, as were "synthetic" utterances, generated from templates that generalized the patterns observed in the transcripts. Moreover, the transcribed utterances, and associated system logs, were used to develop and analyze several context-sensitive language modeling techniques – see [36, 38] and Chapter 5.

Second, based on observations of subjects' interactions, several changes were made to the graphical user interface to make it easier to understand and use. Subjects clearly needed more help simply understanding how to operate the interface – understanding for example, that they needed to push a button before speaking and that they could draw on the screen while speaking. In response, the click-to-talk button was moved to the top of the screen and made much more prominent, and yellow help bubbles were overlayed over the interface to acquaint new users with the system's capabilities. These differences can be seen by comparing Figures 4-1 and 4-2.

In addition, after observing subject interactions, two novel interface techniques were developed specifically for multimodal conversational interfaces: a correctable N-best list (see Section 2.4.1) and a frame containing contextually relevant suggested utterances (see Chapter 9). The correctable N-best list allows users to both see and correct the top speech recognition hypothesis, providing an easy means of correcting common recognition errors, such as confusing one city name for a similar sounding one. The context-sensitive suggestions frame allows users to get an idea of what they can say to move the conversation along, helping to "shape" their utterances so that they are appropriate to the domain, and to the natural language understanding capabilities of the system.

### 4.1.2 *Web* Corpus

The *Web* corpus was collected in 2007, using a substantially improved *City Browser* prototype. In addition to the graphical user interface improvements described in the previous section, the system was also made substantially more useful. Databases of museums and subway stations were added, which augmented the existing restaurants database. The natural language capabilities of the system were also expanded, with the system now able to understand spoken addresses, and to give driving directions.

Given the improved robustness and usefulness of the interface, the next round of data collection was intended to better simulate usage of the system in a real world environment, by having users perform more realistic tasks. Toward this goal, in this data collection effort, subjects accessed *City Browser* from their own computers, remotely via a web page, and performed a series of scenario-based tasks meant to invoke common situations in which a user might want to access an urban information system like *City Browser*.

Subjects were recruited via e-mail announcements; each subject enrolled in the

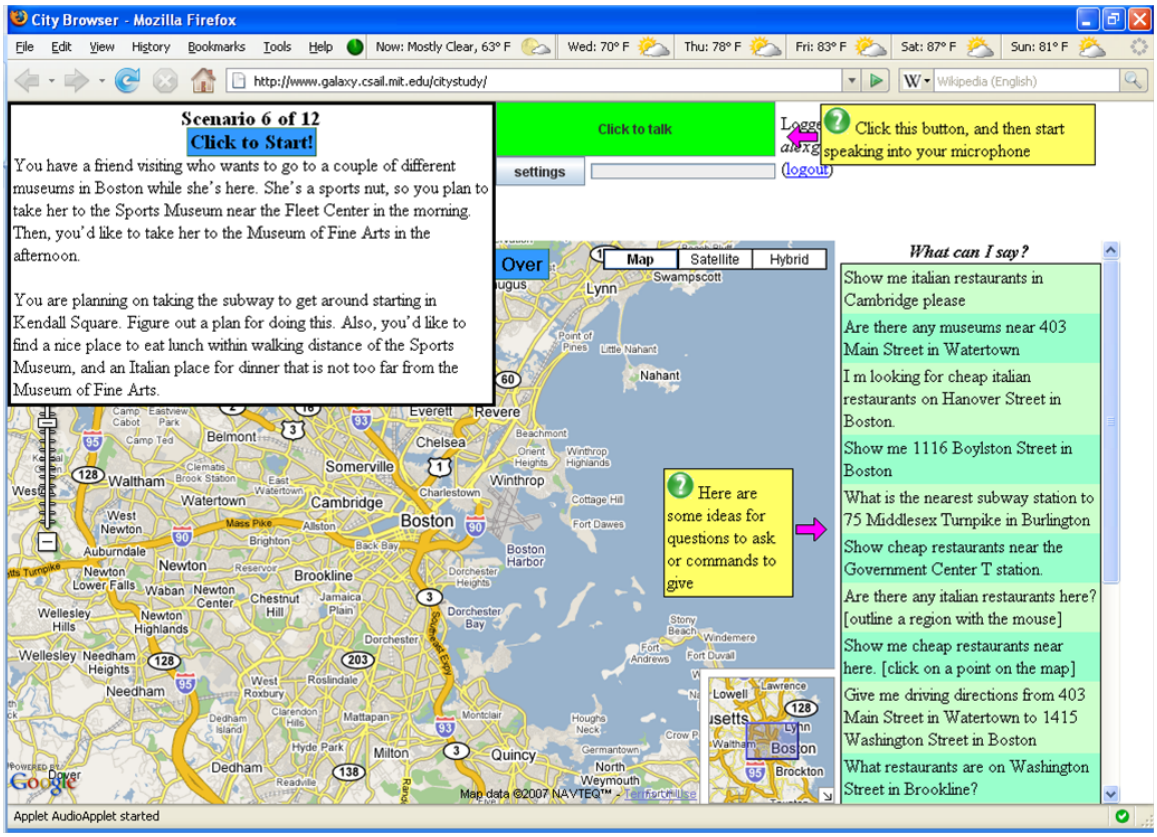



Figure 4-2: Screenshot of the interface used to collect the *Web* dataset. Tasks ("scenarios") were presented visually to subjects in the upper-left corner of the screen. Help bubbles and the context-sensitive suggestions frame are also shown here, which were added to help familiarize new users with the interface.

study and completed it using his or her own computer, microphone and speakers. Because it was web-based, many potential subjects signed up but did not complete the experiment – in large part due to capacity restrictions on the server (unfortunately, at this time, the system only supported two simultaneous users). Many subjects who were turned away due to capacity restrictions did not return to the site. Other potential subjects experienced technical difficulties in getting their microphone or speakers to work properly.

After signing up, subjects completed a warmup task to ensure that their microphones and speakers were functioning. They then completed a series of 11 tasks, listed in Table 4.2. Tasks were presented in sequence, as shown in the screenshot in Figure 4-2. Subjects were then allowed to engage in "free play", interacting with *City Browser* however they liked. Finally, they completed a survey in which they rated the system. Subjects who made an effort to complete all or almost all of the tasks were compensated with a $20 gift certificate.

Thirty eight subjects completed all, or nearly all, of the 11 assigned tasks, resulting in 2,138 subject utterances, which were transcribed. 16% of utterances were accompanied by a contemporaneous drawing gesture, though users were also free to

| Task | Description |
|---|---|
| 0 | Instructions on how to move through the tasks, and a microphone checking task in which users say "Hello City Browser". |
| 1 | You want to make reservations at a Korean restaurant which is located in Cambridge. You can't remember the name, but you do remember that it's on Prospect Street. Look up the *name*, *phone number*, and *address* of this restaurant so that you can call to make a reservation. |
| 2 | You're supposed to meet a friend in Somerville to go out to dinner at an Italian restaurant. Unfortunately, you can't remember its name. You do remember, however, that your friend said it had some of the best Italian food in Somerville! Try to find the Italian restaurant in Somerville with the highest rating (it should be at least 9 out of 10). |
| 3 | You're taking a date to visit the Isabella Stewart Gardner Museum next weekend. You're not sure where it is, so you need to find it on the map. You also want to get their phone number, so you can call to find out about ticket prices and reservations. After the museum, you'd like to be able to walk to a restaurant nearby for dinner. See if you can find one that looks suitable for a dinner date. You're hoping to take the T to the museum, and to get home after dinner on the T as well. Is this feasible? Figure out where you should get off and on at. |
| 4 | You are going to Framingham this weekend to meet a friend for a meal at the Gold Star India restaurant. Find out its address and phone number, and obtain driving directions from the Stata Center (which is in the city of Cambridge, at 32 Vassar St). |
| 5 | You have a friend visiting who wants to go to a couple of different museums in Boston while she's here. She's a sports nut, so you plan to take her to the Sports Museum near the Fleet Center in the morning. Then, you'd like to take her to the Museum of Fine Arts in the afternoon. You are planning on taking the subway to get around starting in Kendall Square. Figure out a plan for doing this. Also, you'd like to find a nice place to eat lunch within walking distance of the Sports Museum, and an Italian place for dinner that is not too far from the Museum of Fine Arts. |
| 6 | You are visiting a friend who is a student at Wellesley, and want to get driving directions from the MIT campus to her house in the town of Wellesley. Her address is 15 Wellesley Ave. MIT is in Cambridge, and you'll be coming from near the main entrance at 77 Massachusetts avenue. Also, see you if you can find a restaurant which is cheap near her house that the two of you can go to. |
| 7 | You're planning on doing some shopping in Boston on Newbury Street next Saturday afternoon. Afterwards you'd like to invite a friend to meet you for dinner somewhere not too far away. Find a convenient Thai restaurant to go to. You'd like to get home on the subway, so you also need to figure out the best subway station to walk to after dinner. |
| 8 | You're going to a Red Sox game at Fenway Park. Find a Mexican place to eat at before the game. Fenway Park is in Boston, located at 4 Yawkey way. |
| 9 | You plan on driving from Boston to Braintree next Saturday to visit a friend. You'll be driving along highway 93, and you'd like to find a Thai restaurant not too far from the highway to stop at along the way. Identify a few Thai restaurants that look convenient to your route to stop at. One way to find the answer is by drawing on the map while you speak. |
| 10 | You're at a party in Waltham at a friend's house at 30 School St. Your buddy is coming to the party too, but doesn't have a car, so he's taking the red line of the subway to the Alewife subway station. Get driving directions from the party, so you can get to the station to pick him up. |

Table 4.2: Tasks assigned to subjects in the *Web* data collection effort in the order in which they were performed.

move the map and click on search results while not speaking. The system operated with a word error rate of 29.2% across all utterances. The class $n$-gram language model during data collection had a core vocabulary of approximately 1,200 words, plus about 25,000 proper nouns.

In addition, the system's response to each utterance was labeled with regards to its correctness. The response error rate (RER) was 52.2%, where it was calculated as:

$$RER = \frac{incorrect + reject}{correct + incorrect + reject} \tag{4.1}$$

and includes utterances only from the non-warmup tasks. While some users fairly easily completed the assigned tasks, others struggled – often due to poor speech recognition performance. This was not surprising, as many users used low-quality microphones, or built-in laptop microphones, which pick up a lot of noise. In addition, users with various accents and technological skill levels enrolled in the study. While these factors may have led to lower accuracy levels, it was quite useful data to obtain, and is surely a much more realistic simulation of real-world usage than a controlled laboratory-based data collection effort.

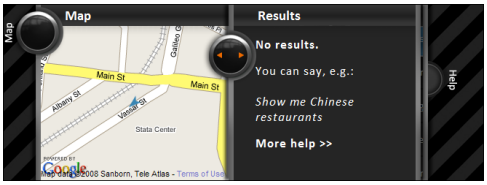
Data and observations from the experiment were used to analyze usage of the context-sensitive suggestions module discussed in Chapter 9, and correctable N-best module (Section 2.4.1). Moreover, the collected data was used to develop the response-based confidence annotation module described in Chapter 7.

At a higher level, the study serves as a first validation that the Web can successfully be used to make multimodal interfaces available to users outside of the laboratory. The logged experiences of users – even, and perhaps especially, ones who had technical difficulties or problems being understood by the system – have proved quite helpful in making the technology needed for web-based multimodal interfaces more robust.

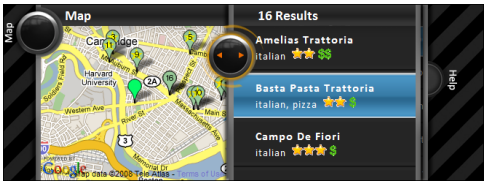
### 4.1.3 *Car* and *Car-Pilot* corpora

The *Car* and *Car-Pilot* corpora were collected from December 2008 - May 2009. Subjects interacted with a version of *City Browser* running inside a BMW 530xi sedan. The car's built-in display and iDrive controller were used to display and control a car-optimized GUI, shown in Figure 4-3. Speech input was captured by an array microphone on the driver's sun visor, and spoken language output was played over the car's speakers. A dedicated speech button near the iDrive controller served as a hold-to-talk button for users to hold while speaking to *City Browser*. In addition, the car's current position was shown on the map, and made available to *City Browser* as a constraint for search and driving directions.

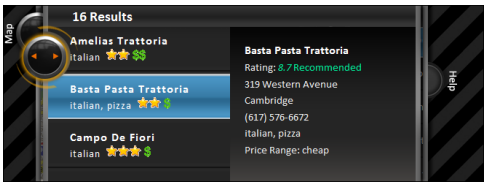
The natural language understanding components of the *City Browser* system were functionally similar to the version deployed in the previous *Web* study, with the exception that a database of hotels had been added. However, despite this functional similarity, the components had undergone a major re-tooling, allowing for a significant increase in speed and robustness. The new architecture made it possible to integrate into the live system the response-based confidence annotation module developed and tested offline using the *Web* corpus (see Chapter 7). The language model used by



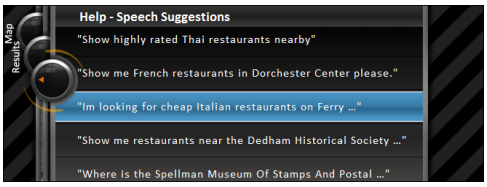

(a) Initial screen showing the position and orientation of the car and short help




(b) Search results screen for *Show me Italian Restaurants*




(c) Detailed information pane for one result




(d) Context-sensitive speech suggestions

Figure 4-3: Screenshots of the *City Browser* interface deployed in the car.

the speech recognizer was nearly identical to the one used while collecting the *Web* corpus, with the exception that approximately 800 hotels in the Boston area were added. This also required augmenting the trigram training data to include some hotel-related example utterances.

Subjects sat in the driver's seat of a parked BMW sedan and interacted with *City Browser* to complete a variety of tasks. In addition, several non-invasive physiological sensors were placed on each subject to measure heart rate, skin conductance level, and respiration over the course of the experiment. These data were gathered so that collaborators in MIT's Age Lab could evaluate the level of arousal of each subject over time. Each subject also completed pre- and post-experiment questionnaires.

Thirty three uncompensated subjects participated in the *Car-Pilot* phase, which was used to iteratively refine *City Browser*: bugs were fixed and the natural language understanding capabilities of the system were enhanced over the course of the data collection. After completing the pre-experiment questionnaire and the tutorial, each subject was presented with the 12 tasks shown in Table 4.3, printed on index cards.

Of the 33 subjects, 31 completed 9 or more tasks – the remaining 2 had great difficulty interacting with the interface, due to poor speech recognition accuracy. 1,651 utterances were collected and transcribed; the word error rate for these utterances was 31.9%. Annotators also marked whether the system responded correctly, as well as if any of its other N-best candidate responses were correct. These annotations indicated a system response error rate of 44.0%.

Following this period of system refinement, development was "frozen" and a follow-on study was performed, resulting in the *Car* corpus. The experiment followed the same protocol, with the exception that two of the tasks were eliminated and the tasks were reordered somewhat as shown in Table 4.3. Task 12 was eliminated because it was perceived as repetitive, task 8 because it was confusing; in addition, experimenters observed subjects becoming restless during the final few tasks, suggesting that the experiment was too long. The tasks were reordered based on qualitative observations

| *Task Order* Pilot | Car | Description |
|---|---|---|
| 0 | 0 | Tutorial task, part of which involves the subject being instructed to say "Show me Chinese Restaurants" |
| 1 | 1 | You need to find a hotel in Somerville. Use the system to find the name of a hotel. |
| 2 | 2 | You'd like to find a restaurant that's just a short drive away for lunch. You are in the mood for Indian food. |
| 3 | 5 | You're showing some friends around Boston and want to go to the Museum of Science. Get directions to the museum. |
| 4 | 3 | You're meeting up with some friends in Cambridge and want to take them to a Chinese restaurant. Find one and get directions to it. |
| 5 | 4 | You're supposed to go to Chelmsford to meet a friend at a restaurant called Garlic Bistro. Find out their phone number, so you can call them to make a reservation. |
| 6 | 6 | You're on your way to a meeting in Boston, located at 28 Huntington Ave. Get directions to this address. You'd like to take your client out to dinner afterwards somewhere within walking distance. Find an Italian restaurant – something upscale that looks good – to suggest, and get its phone number and address. |
| 7 | 7 | Your aunt is in town and you want to take her to the Museum of Fine Arts. Get directions to the museum. |
| 8 | - | You are visiting a friend in Brighton and would like to go with him to a Greek restaurant. Find one and get directions to it. |
| 9 | 8 | You're supposed to meet a friend at a restaurant called Fugakyu, but neither of you remember exactly where it is. You know it's in Brookline, and that it's Japanese. Get the address so you can tell your friend where it is, and get directions there for yourself. |
| 10 | 9 | You've got a friend visiting from out of town who is staying in Boston at the Sheraton hotel. Get directions there so you can pick him up. Find a restaurant that doesn't cost a lot of money to go to for dinner which is near the hotel. |
| 11 | 10 | You're picking up a friend from his apartment in Quincy at 180 Hancock Street. Find the address on the map and find a cheap restaurant near his apartment to have dinner. |
| 12 | - | You've just gotten in the car and plan to head to a restaurant called Thai Moon which is located in Arlington. Get directions to this restaurant. |

Table 4.3: Tasks assigned in the *Car-Pilot* and *Car* datasets, the numbering indicates the order in which the tasks were presented. In the pilot phase, the tutorial plus 12 tasks were used. In the follow-on study, only 10 of the tasks were used, and they were slightly reordered.

and quantitative analysis of their difficulty, as a particular pattern of difficulty was deemed useful for physiological analysis [35]. In addition, subjects were recruited from outside the university community, and the pool was balanced by gender and across three age groups. Subjects were compensated with a $20 gift certificate.

The *Car* corpus contains interactions of 92 subjects, each of whom completed 10 tasks, resulting in a total of 3,492 utterances. The word error rate for these utterances is 36.1%, and the response error rate is 46.9%.

## 4.2 Comparison to Similar Corpora

Collecting data from real users interacting with conversational, multimodal interfaces can be quite a resource-intensive process. First and foremost, developing even prototypes of such systems is challenging, as practitioners must be versed in a variety of technologies: speech recognition, natural language parsing and generation, knowledge representation, database design, and user interface design – to name a few. Moreover, conversational multimodal interfaces are largely unfamiliar to users: while they can be a highly effective means of obtaining information, each is typically restricted to a particular domain of knowledge. Users are much more familiar with using purely graphical interfaces such as search engines to solve similar tasks. Search engines have a simple model of interaction: search for anything you like, browse the results, and then iterate or refine. Conversational interfaces, on the other hand, understand a wide range of natural language, but only over a narrow domain. This disconnect often requires a re-orientation of users' expectations and behavior to match the capabilities of an unfamiliar interface during a brief exposure.

Beyond system development, data collection is quite resource intensive when it is performed in the laboratory: subjects must be recruited, scheduled, consented, and then run through the experiment. The automotive data collection efforts described in Section 4.1.3, for example, were quite resource intensive: collecting and transcribing, the approximately 50 utterances that might make up a single subject's interaction, typically required 2 or more hours of effort in recruiting, running the subject, and transcription time. While this study yielded a wealth of valuable data – transcribed utterances, videos, audio recordings, physiological measures, and survey results – it does not scale cheaply.

In contrast, the *Web* data collection described in Section 4.1.2 was designed to be more scalable: subjects were recruited via e-mail announcements, and interacted with *City Browser* at their leisure via a web interface. Once started, a study performed in this manner has a much lower incremental cost: subjects complete the study without any supervision, meaning that the only per-subject cost is transcription, annotation, and analysis. While the *Web* corpus collected here was still relatively small in size, it represents a pilot effort to collect multimodal user interface data via the web – the first of its kind.

Taken together, the data collection efforts in this thesis represent a significant contribution to the set of available corpora collected from subjects' interaction with multimodal conversational interfaces. At the moment, there are very few such cor-

| Corpus | Subjects | User Utterances | Domain | Location |
|---|---|---|---|---|
| ***City Browser*** | | | | |
| Tablet | 10 | 546 | Search | Lab |
| Web | 38 | 2,138 | Search + Navigation | Web |
| Car-Pilot | 33 | 1,651 | Search + Navigation | Car |
| Car | 92 | 3,492 | Search + Navigation | Car |
| **Multimodal Conversational Interfaces** | | | | |
| WITAS [45] | 20 | * | UAV Control | Lab |
| WITAS [21] | 6 | 303 | UAV Control | Lab |
| AdApt [44] | 26 | 4,388 | Apartment Search | Lab |
| MATCH [49] | 5 | 338 | Search + Navigation | Lab |
| MATCH [50] | 39 | 3,116 | Media Selection | Lab |
| CHAT [91] | 20 | * | Restaurant Search | Lab |
| CHAT [91] | 20 | * | Music Selection | Lab |
| **"Wizard-of-OZ" Multimodal Conversational Interfaces** | | | | |
| AdApt [40] | 33 | 1,845 | Apartment Search | Lab |
| CHAT [91] | 50 | * | Search, Navigation, Music | Lab |
| SAMMIE-2 [54] | 42 | * | Music Selection | Lab |
| **Telephony Conversational Interfaces** | | | | |
| Jupiter [101] | * | 180,000 | Weather | Telephone |
| Communicator [88] | * | 53,394 | Travel Planning | Telephone |
| Let's Go [70] | * | 100,000+ | Bus Information | Telephone |

†Estimated from statistics as of 11/4/2008 at `http://cmuletsgo.org`
* Indicates unknown or unreported

Table 4.4: Characteristics of corpora of similar multimodal conversational interfaces, as well as of several telephone-based conversational interfaces for comparison. The ubiquity, ease-of-use, and familiarity of telephones has allowed research labs (with small staffs and limited budgets) to collect orders of magnitude more data over the telephone than via multimodal interfaces, which has led to commercialization of such interfaces.

pora, and most are very small in size. Table 4.4 compares the data collection efforts presented in this chapter with other similar efforts. Note that the multimodal corpora cited in Table 4.4 all involve tens of subjects, and at the most several thousand utterances. These numbers are orders of magnitude lower than those of corpora collected using telephone-based conversational systems, a few of which are also shown for comparison in Table 4.4.

Telephone-based conversational interfaces are now widespread, in part because the telephone is ubiquitous and easy to use – allowing researchers to collect large amounts of data and companies to deploy systems that are easy for their customers to access. Such data is critical for speech recognition, as acoustic and language models require large amounts of training data. It's also critical for spurring new types of research – for example, the response-based confidence scoring approach described in the thesis could never have been developed and validated without this chapter's corpus collection efforts. If conversational multimodal interfaces are to become mainstream, then data

collection efforts that generate orders of magnitude more data – like the web-based ones piloted here – must become more widespread. Toward that goal, the technologies used to make *City Browser* available on the web for this study have been packaged into the open-source WAMI toolkit [34], which has already been used for other web-based data collection efforts (*e.g.*, [62]).

## 4.3 Annotation Tools

Thus far, this chapter has focused on the problem of collecting interaction data using multimodal interfaces; however, once this data is collected, it often must be transcribed and annotated in order for it to be useful. Moreover, system developers also need a sensible way of reviewing the collected data, in order to understand the ways in which subjects interacted with the system. For telephone-based interfaces, this boils down to the ability to play back the conversation; by hearing both the human's and computer's utterances in sequence, it is possible to accurately transcribe the human participant's speech, to judge whether the system accurately responded, and to understand the "flow" of the interaction.

In contrast, pure auditory information does not suffice when studying multimodal interfaces, as users may spend a lot of time using the graphical user interface (GUI) – both on its own and while speaking. When studies are done in the laboratory, this interaction can be observed by the experimenter (as in the *Tablet* corpus) and/or recorded as video (as in the *Car* and *Car-Pilot* data collection efforts). However, as was noted in the previous section, such techniques do not scale well to large numbers of users. Larger scale studies must involve users interacting with the system on their own, unobserved.

Ideally, system designers should still be able to gain at least some understanding of the way in which users are combining their use of the GUI with speech – even when the interface is deployed remotely, as in the case of the *Web* corpus. To facilitate this, the *City Browser* interface has been instrumented such that the user interactions are captured and logged to the server. This means that all drawing gestures are logged, as are all clicks on displayed points of interest, corrections made via the correctable N-best list, and any window scrolling. In addition the map's bounds are recorded any time they change, and the search results displayed on the map are always recorded. All logging is done by sending messages asynchronously in the background, using standard AJAX techniques, so that there is no noticeable delay to the user.

In order to view this wealth of logged data, an interaction "play back" tool has been developed, in which the system developer, transcriber, or data annotator can view the GUI as the user saw it, watching it change over time as the user speaks, the system responds, and the user interacts with the GUI. Such playback capabilities allow system developers to gain the same sorts of insights that might otherwise be gleaned from observation: How do users respond when the system doesn't understand them perfectly? Do users explore several search results, or just one? Moreover, because interaction events are logged, quantitative characterizations of user interaction can be made, as is done in this thesis for the correctable N-best list and the context-sensitive

suggestions.

The ability to observe an interaction can also be critical in understanding if a system's response was even correct in a given situation. For example, a user might say "give me directions there", where "there" might refer to a previously mentioned location, or a search result the user just clicked on. By watching the log playback, an annotator can understand if the pronoun was resolved correctly; and, if not, why the system may have made an error.

This log playback capability has been integrated with a web-based transcription tool, various versions of which have been used to annotate and transcribe the *City Browser* corpora. Screenshots of the tool are shown in Figure 4-4.

It allows annotators or system developers to view a list of sessions, as shown in Figure 4-4a. After choosing a session to annotate, they are presented with the interface in Figure 4-4b. There they can transcribe each utterance, and view all of the candidate system responses generated by the response confidence scoring module. Each response can be labeled with regards to its correctness. Finally, they can also play back the session, seeing the *City Browser* interface as it originally looked to the user. In this mode, they can hear both what the user and system said, see any of the user's drawing or clicking gestures, and watch as the GUI is updated. This helps annotators understand what was happening in the session, and get a sense for the pace of the interaction.

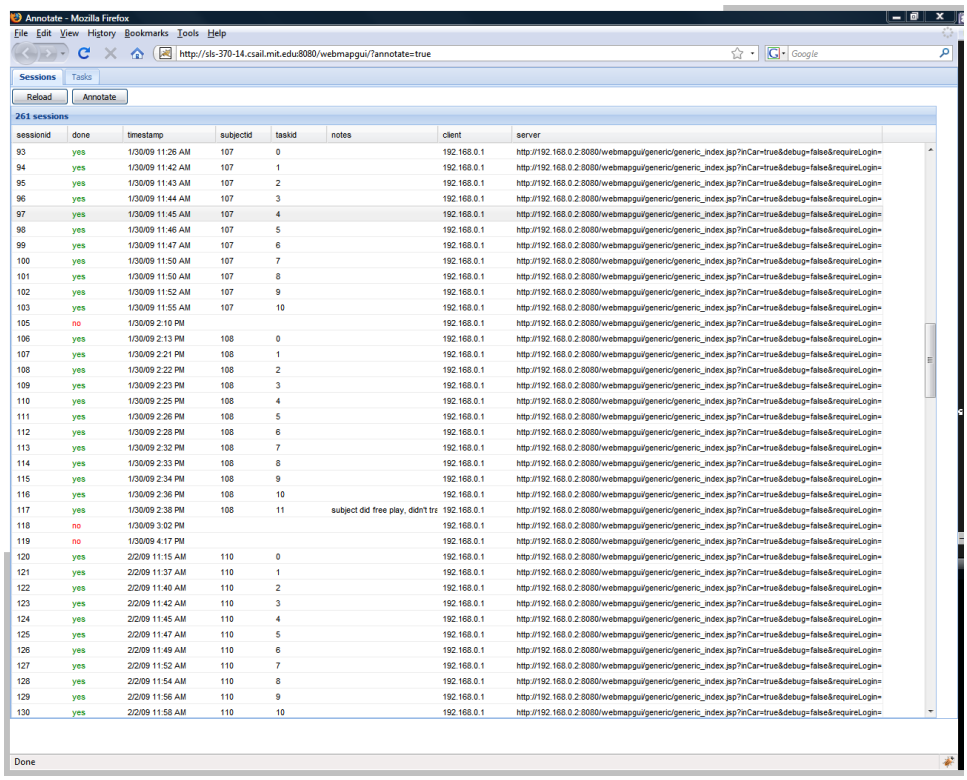
## 4.4 Summary

In this chapter, several data collection efforts were discussed in which subjects interacted with *City Browser*. In particular, three distinct efforts were described in which data was collected in the laboratory, via the web, and in an automobile. These data collection efforts led to four distinct corpora, encompassing a total of 7,827 user utterances, which have been transcribed. The *Web* corpus is the first of its kind, in that it contains the results of a user study with a conversational multimodal interface conducted entirely via the web, meaning that subjects completed the study using their own computers, completely outside of a laboratory environment. While small in size, it serves as a proof-of-concept that such studies are possible, and has paved the way for similar web-based data collection efforts [62].

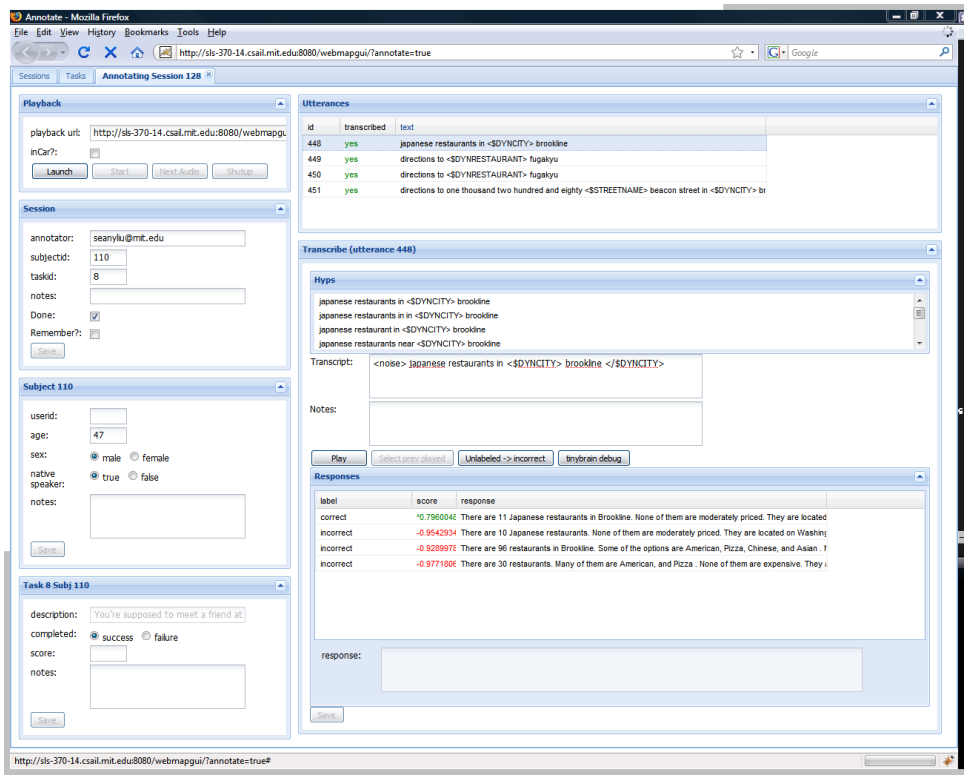
The *Car* and *Car-Pilot* corpora explore another avenue of migrating multimodal conversational interfaces out of the laboratory: in this case, subjects interacted with *City Browser* while in the driver's seat of an automobile. While certainly more controlled than the web-based experiments, they do represent an effort to study users interacting outside of the laboratory. In particular, it is challenging to rapidly acquaint users with the interface, so that they can quickly start using the system.

## Acknowledgments

Data collection, transcription, and annotation is a complicated and labor-intensive process. Many people were instrumental in the efforts discussed in this chapter, which




(a) Session list




(b) Transcription and Annotation Window

Figure 4-4: Screenshots of the web-based transcription tool used to transcribe and annotate the *City Browser* corpora.

would have been impossible without them.

Liz Murnane designed and implemented the "help bubbles" displayed during the *Web* data collection. Sean Liu performed a significant portion of the transcription and annotation of this corpus. Marcia Davidson handled gift-certificate distribution for the *Tablet* and *Web* efforts.

The *Car* and *Car-Pilot* efforts were undertaken by a large team. BMW provided the car; Jeff Zabel modified it and provided the software necessary to deploy *City Browser* in it. Sean Liu designed and implemented the GUI used in these experiments. Jarrod Orszulak, Shannon Roberts, Bruce Mehler, and Bryan Reimer developed the experimental protocol; they were responsible for all physiological monitoring aspects of the experiments. Jarrod and Shannon, with Alea Mehler, Eugenia Gisin, Michael Thompson, Tina Stutzman, Katharine Binder, and Jacob Wamala handled all aspects of running the subjects through the experiments. While I annotated the *Car-Pilot* corpus, the *Car* corpus was annotated by Shannon Roberts, Tina Stutzman, Katharine Binder, Sean Liu, Jarrod Orszulak, and myself. Finally, Jim Glass and Bryan Reimer initiated the collaboration, and, together with my advisor, Stephanie Seneff, provided key guidance throughout the process.

# Chapter 5

# Context-Sensitive Language Modeling

Computational systems that are designed to understand spoken language rely on a speech recognizer to transcribe an audio signal containing spoken language into a string of words. Speech recognizers, in turn, rely on language models, which assign a probability to every possible sequence of words; this constrains the search problem, and allows the speech recognizer to rank acoustically similar word sequences. Typically, conversational interfaces use language models tailored to the specific task domain supported by the system; these language models contain only words relevant to the domain, which means that generally the speech recognizer will only use words appearing in the vocabulary of the language model to transcribe utterances. A good language model will assign high probabilities to sequences of words that the user is likely to say at a given time, and lower probabilities to unlikely sequences of words. Language model scores are then combined with acoustic evidence to rank hypothesized spoken word sequences.

A good language model is critical to the usability of a spoken or multimodal conversational interface because it becomes increasingly difficult for such systems to respond accurately to a user's utterance as the number of errors made by the speech recognizer increases. Indeed, language modeling considerations play a major role in the design and implementation of all conversational systems. For example, many telephone-based systems are designed such that the computer takes the initiative in controlling the flow of the conversation by leading the user through a series of carefully designed prompts for which there are only a few plausible responses. Such systems typically employ a restricted language model for each prompt, containing just the few phrases the user is expected to utter. When users are cooperative, this strategy can yield high accuracy, because the speech recognizer needs only to distinguish among a small set of utterances.

System-driven interactions have proved to have utility in some transactional domains, where calls to an automated system are geared at completing a series of steps involved in completing a particular transaction. However, they often prove awkward in more exploratory domains like *City Browser*, where the user's next step after, say, obtaining a list of restaurants in response to a query, is not always clear: the user

might want to refine this list, get driving directions, find a subway station, and so on. A menu driven system where the user had to always choose the next step from a list of options would surely feel unnatural. Indeed, in transactional domains, users usually find menu-driven systems unnatural as well. As a result, much dialogue systems research has been focused on giving the user greater control over the conversation by allowing him or her to speak naturally about a wide variety of capabilities at any time. Such interfaces, however, can also prove frustrating if the speech recognizer can't accurately transcribe what is being said.

Despite the important role of language modeling on speech recognition accuracy, conversational interfaces in which users are relatively free to control the conversational flow typically make use of a single, large language model meant to cover all possible utterances a user might make during the entire interaction. Such a configuration is easiest to deploy, but fails to make use of contextual knowledge: intuitively, as a conversation unfolds, knowledge about what has been said so far ought to be extremely useful to predict what will be said next – indeed, humans are often able to complete one another's sentences. It is an open problem, however, to determine how to usefully integrate such contextual expectations into the form of language modeling constraints useful for the speech recognition algorithm.

Figure 5-1 illustrates graphically the premise underlying context-sensitive language modeling, as it applies to typical conversational dialogue systems. It shows the typical flow of information in such an interface: users speak and gesture, their words and gestures are recognized and understood in context, and the conversational system then produces an appropriate response, perhaps both graphically and verbally. The premise of context-sensitive language modeling is that, given its knowledge of what has been said by the user and what will be said by the system, the dialogue manager ought to be able to produce expectations about what the user is likely to say next, which in turn should be used to update the language model used during speech recognition.

In this chapter, a novel method of conditioning language model probabilities based on contextual knowledge is described. In particular, while a single class $n$-gram language model is trained and deployed for use in recognizing each user utterance, the weights for certain words and phrases in that model are adjusted as the conversation progresses, via the use of *contextualized semantic classes*. These classes are marked in the training corpus using system logs, and then are populated at run-time prior to each user utterance with phrases drawn from, or relevant to, the current context of the conversation.

## 5.1 Background

A language model assigns a probability $P(w_1, \ldots, w_m)$ to observing a sequence of words $w_1, \ldots, w_m$. Speech recognizers use language models to provide probabilistic constraints about the sequences of words that are likely to be heard. This helps recognizers to differentiate among sequences of words that might sound similar to one another. A classic example is the following pair of sentences:

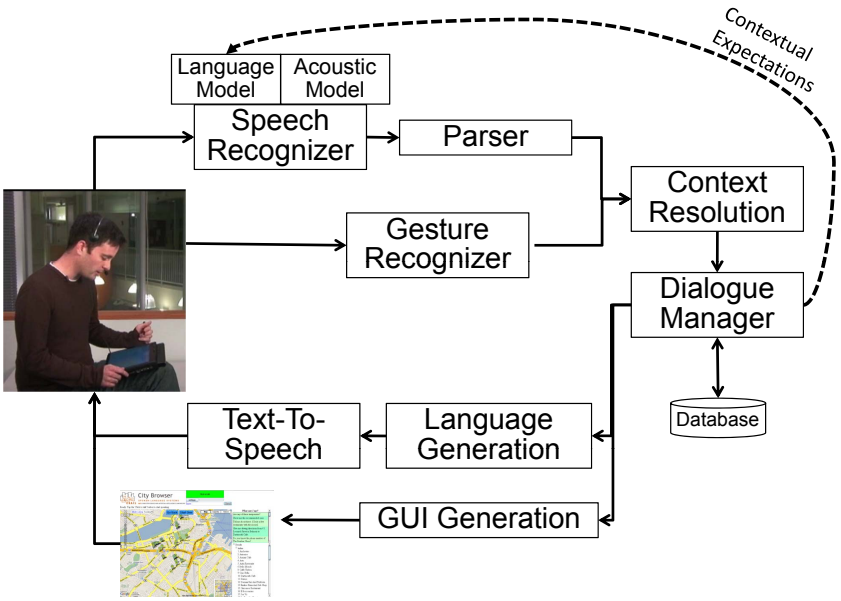



Figure 5-1: Diagram illustrating the typical flow of information in a multimodal conversational interface. The dotted line illustrates the intuition behind context-sensitive language modeling, namely that contextual expectations regarding what the user is likely to say given the current context of the conversation can be inferred from information typically available to the dialogue manager and then transferred to the language model to improve recognition accuracy for subsequent user utterances.

A: *This machine can recognize speech.*

B: *This machine can wreck a nice beach.*

The two sentences sound very similar. However, a language model meant to provide the probability of observing either sequence of words in a thesis about speech recognition should assign a higher probability to sentence $A$.

Speech recognizers combine the probabilistic information in language models with acoustic and lexical models to create a rank-ordered list of hypotheses for a given utterance. In theory, a language model could be defined using any algorithm that, given a sequence of words, assigns a probability to that sequence. However, to be used with a speech recognizer, the algorithm must be compatible with the dynamic search procedure used to find an optimal sequence of words. As a practical matter, then, language models used for speech recognizers almost always take one of two forms:

1. $n$-gram statistical language models,
2. Probabilistic context-free grammars (PCFGs).

This thesis is concerned with $n$-gram language models, which are generally more flexible because they can assign a non-zero probability to any sequence of words by exploiting a back-off model. PCFGs are constrained by grammar rules and therefore subject to hard failure. Since several of the methods discussed as related work use PCFGs, they are also briefly explained.

### 5.1.1 $n$-gram Language Models

$n$-gram language models (see Chapter 6 in [51]) are built on the idea that the probability of observing a sequence of words can be calculated by the product of the probability of seeing each word in the sequence, given the sequence of words preceding it.

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}) \tag{5.1}$$

Each probability in the product is estimated by counting how many times that word sequence appears in a corpus of training data. Since training data is finite, however, $n$-gram language models approximate the probability of seeing a given word by considering only the N-1 preceding words (a Markov assumption):

$$\prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i | w_{i-(N-1)}, \ldots, w_{i-1}) \tag{5.2}$$

#### Classes

Class $n$-gram language models (see [10] and Chapter 8 in [51]) further approximate the probability of seeing a particular word in context by grouping words (or phrases) into classes. The idea is that certain words, for example the days of the week, can easily

be interchanged with one another; that is, a sentence like *I'll see you on Friday* could just as easily have been *I'll see you on Monday*. By grouping together the names of the days of the week, whenever one is seen in the training corpus, information about the others can be gained as well. This means that if a word $w_i$ is a member of class $c_i$, its probability of being observed in a particular context can now be calculated as follows:

$$P(w_i|w_{i-(N-1)}, \ldots, w_{i-1}) = P(w_i|c_i)P(c_i|c_{i-(N-1)}, \ldots, c_{i-1}) \tag{5.3}$$

The above equation assumes each word must be assigned to a class. Of course, many words can be assigned to a simple class, of which they are the only member. In this case $P(w_i|c_i) = 1$ and the class assignment has no effect. Otherwise, $P(w_i|c_i)$ is referred to as the *within-class* probability, and it can be calculated in several ways, as the experiments in the next chapter illustrate. The most common method is to use the maximum likelihood estimate from the training corpus:

$$P(w_i|c_i) = \frac{C(w_i)}{C(c_i)} \tag{5.4}$$

where $C(w_i)$ is the number of times $w_i$ was observed in the training corpus, and $C(c_i)$ is the number of times any member of class $c_i$ was observed.

Finally, the context-sensitive language modeling methods proposed in this chapter assume that a word $w_i$ may be assigned to two classes. In this case, in the language model, $w_i$ is effectively treated as two distinct tokens: $w_{i1}$ and $w_{i2}$. However, in calculating error rates, both tokens are treated equivalently.

#### Dynamic Classes

Many speech recognizers that support the use of class $n$-gram language models provide for *dynamic* classes, in which the words assigned to a particular class, and their within-class weights, can be modified at runtime. Practically speaking, this makes it straightforward and computationally efficient to load a single $n$-gram language model into memory, and then alter this language model by changing the contents of the classes at runtime. This provides a mechanism, for example, to efficiently personalize language models for particular users. The names of all the contacts in a particular user's address book might, for instance, be used to customize a voice dialing application. Or, after acquiring a user's current location, restaurant names near that location might be loaded into a particular dynamic class in a large $n$-gram language model, as in [16]. In this chapter, a novel use of such dynamic classes is explored, in which class membership is determined based on conversational context.

### 5.1.2 Probabilistic context-free grammars

A probabilistic context-free grammar (PCFG) is simply a context-free grammar (see Chapter 9 of [51]) in which a probability is assigned to each production rule. While not used for any of the recognition experiments in this thesis, PCFGs are commonly used in speech recognition applications, including in some of the related work discussed here

and in Chapter 9. The probability of a word sequence is assigned using a PCFG by considering the probabilities assigned to the production rules used to form a "parse" in which the words in the sequence appear as terminal nodes.

Such grammars are often used in speech recognition because they are linguistically motivated. Linguists have created a variety of formalisms for representing the syntactic structure of natural language, (*e.g.*, famously, in [13]). Many of these formalisms can either be reduced to context-free grammars, or approximated by them. Typically, while grammar rules may be written by hand, the probabilities assigned to the production rules are estimated by parsing a corpus.

### 5.1.3 Training Corpora

In order to estimate probabilities, both $n$-gram language models and PCFGs rely on a training corpus. The corpus is a list of utterance transcripts, ideally transcribed from actual utterances collected in a similar situation as the one in which the language model is intended to be used. In the best case, the corpus used to train the language model for a particular conversational interface would consist of thousands or millions of transcribed utterances of real users interacting with that conversational interface. Of course, such a corpus is rarely available, since gathering such a corpus requires a working conversational system, which itself requires a trained language model. Typically then, language models are trained through iterative rounds of data collection. To bootstrap data collection, language models for early rounds of data collection are typically trained using copora generated with one or several of the following techniques:

- Transcribed user utterances from "wizard-of-oz" experiments, in which subjects believe they are interacting with an actual conversational interface, but – in fact – that interface is actually under the control of a human "wizard" [17],
- "Synthetic" utterances generated from templates or grammars, created by system developers, that specify what they believe users are likely to say (*e.g.*, [22, 38]),
- Synthetic utterances generated from the natural language parsing grammar (*e.g.*, [100]),
- Synthetic utterances generated by adapting existing corpora from other domains (*e.g.*, [15, 22]),

Usually, language model creation is an iterative process in which early versions of an interface are deployed using synthetic and/or "wizard-of-oz" data. After users interact with these early versions, their transcribed data is then added to the training corpus, and a new language model is created. As even more data is gathered, the language model is iteratively retrained.

### 5.1.4 Word Error Rate

Speech recognition systems are often evaluated in terms of their word error rate (WER). The WER is calculated by comparing a speech recognition hypothesis to a transcript for a given utterance, and calculating the minimum number of word-by-word changes required to change the hypothesis to the transcript, where the hypothesis may have words deleted, inserted, or substituted. This is the Levenshtein distance (or "edit distance") between the transcript and hypothesis. Word error rate is then defined as:

$$WER = \frac{Substitutions + Deletions + Insertions}{N} \tag{5.5}$$

where $N$ is the number of words in the transcript.

To determine if the differences in WER produced by two different speech recognizers is statistically significant – that is, unlikely to be due to chance – the NIST Matched-Pair Sentence-Segment Word Error (MAPSSWE) [25, 67] is used in this thesis, as is typically done. $p$-values of less than .05 are considered significant.

## 5.2 Related Work

Previous research into effectively leveraging contextual information for the speech recognizer's language model in conversational interfaces has typically focused on creating language models specific to each possible "dialogue state". The most extreme examples can be found in conversational interfaces that have very strong system initiative – that is, in which the computer system drives the flow of the conversation. Many such systems are designed using VoiceXML [86], which allows interface designers to specify a series of dialogue states. In each state, the system queries the user for one, or a few, pieces of information; for example, the name of a city. Associated with each state is a particular context-free grammar, which provides the language model constraints for the expected utterances. For example, if the conversation is currently in a state where the user is expected to say the name of a city, then a grammar with exactly a list of expected city names would be used for utterances in that state. An example VoiceXML program and a conversation supported by this program are shown in Figure 5-2; note that a series of fields are filled in, and that associated with each field is a particular grammar.

Developers using VoiceXML can tightly constrain the language model grammar after each system prompt, because language model grammars and "dialogue state" are tightly coupled. Such coupling can allow for high accuracy rates – when prompts help users to speak within the confines of the tight grammar – but can lead to unnatural, inflexible conversational interfaces.

Interfaces that aim to have a more natural conversational flow typically do not use the "dialogue state" model employed by VoiceXML, where the conversation flows from one distinct form, or field within a form, to another. Instead, such systems usually employ more complex strategies to decide how to interpret what a user has said, and what to say next in response. Strategies vary – see [8, 59, 47, 75] for examples – but what remains constant across them is that "dialogue states" are no

```
<form>
 <field name="city">
  <prompt> Please choose a city. </prompt>
  <grammar type="application/x-nuance-gsl">
    [london paris (new york)]
  </grammar>
  <filled>
    Ok, <value expr="city">!
  </filled>
 </field>

 <field name="type">
  <prompt> What would you like to know about <value expr="city">?
           Say weather or transportation. </prompt>
  <grammar type="application/x-nuance-gsl">
    [weather transportation]
  </grammar>

  <filled>
    Ok, I'll get information about <value expr="type">.
  </filled>
 </field>
</form>
```

*S1*: Please choose a city.
*U2*: London.
*S3*: OK, London! What would you like to know about London? Say weather or transportation.
*U4*: Weather.
*S5*: OK, I'll get information about weather.

Figure 5-2: An example VoiceXML program, and a dialogue that it allows. $S$ indicates system utterance; $U$ indicates user utterances. In VoiceXML, a grammar is explicitly associated with each dialogue state, defined in terms of forms and fields.

longer explicitly defined, meaning that there is no longer an opportunity to associate particular grammars with each state. Thus, despite the rich contextual knowledge used in such systems to understand and respond to natural language appropriately given what has been said so far in the conversation, most flexible conversational interfaces do not bring this contextual information to bear on the speech recognizer's language model. Instead, it is typically the case that a single language model is used by the speech recognizer in all conversational contexts.

Research focusing on improving language models with contextual information in systems like these has, despite rich contextual information, still centered around the idea of a "dialogue state". Techniques that have been explored have generally centered around the idea of *partitioning* the rich contextual knowledge in such systems into dialogue states – akin to the states used in Voice XML – and then assigning a language model to each state. Such techniques have been tried both with grammar language models, like those used in VoiceXML, and statistical $n$-gram models trained on corpus data. The remainder of this section summarizes these two approaches.

**Context-sensitive grammars** Many flexible conversational systems make use of context-free grammars as their speech recognizer language models, like those used in VoiceXML. Often, however, these context-free grammars are derived from linguistically motivated grammars used by the natural language parsing components of the system, as is done in [58, 81]. It is possible to isolate the grammar rules used to understand answers to particular prompts, for example to system questions where a "yes" or "no" answer is expected. In the WITAS system described in [58], the dialogue system's "information state" is used to determine which subset of the grammar rules should be active, given a user's expected range of responses to a particular prompt. That grammar subset is then used as the speech recognizer's language model. If the recognizer fails to produce a hypothesis above a particular confidence threshold, a second recognition pass is performed in which the entire grammar is used as the language model.

Experiments on a small set of subjects showed that the context-specific language model was used for 87.9% of utterances. When compared with a system using only the entire grammar, this provided 11.5% and 13.4% reductions in the word and concept error rates respectively. While this technique is powerful, it has several key drawbacks:

1. It relies on multiple recognition passes, which can cause significant latency.

2. The confidence threshold for initiating a second recognition pass must be tuned.

3. The grammar must be partitioned by hand, which may be difficult to do and may involve a trade-off with modularity in grammar design.

4. A mapping algorithm is required to choose the appropriate sub-grammar given a particular information state.

**Context-sensitive $n$-gram language models** $n$-gram language models are preferred to grammars in many applications because they offer greater flexibility in

dealing with a wide range of utterances. Context-sensitive language modeling techniques have also been developed involving $n$-gram language models. For example, in [1, 79, 85, 92, 95] it is shown how dialogue-state-dependent $n$-gram models can be used to increase accuracy when they are interpolated with the $n$-gram model derived from a larger set of in-domain data. In each case, the training corpus is divided into sub-corpora – one for each dialogue *s*tate, where a set of dialogue states must be determined using the conversational context. Each sub-corpus consists of transcribed utterances collected from users while the dialogue system was in that state.

An $n$-gram language model is then trained on each sub-corpus. Often, this state-specific language model is then interpolated with a larger model, trained using the entire corpus. An important parameter in this approach, then, is the relative weights of the small and large language models in the interpolation. One way to set these relative weights is by minimizing perplexity on a held-out set. Perplexity, however, does not always correlate well with word error rate.

### 5.2.1 Limitations

The family of context-sensitive language modeling techniques presented immediately above have in common that they require system designers to map complex information states into a discrete set of dialogue states, so that a specific language model can be designated for each state. The level of granularity in the design of these states is a design decision that may have a strong impact on recognition accuracy. In the case of $n$-gram models, the system designer must create definitive segmentations that balance the fine-grainedness of the dialogue states with data sparseness. Moreover, as the complexity – and, hence, the number of states – of the dialogue system increases, it becomes increasingly difficult to find the right combination of states to interpolate – though automatic clustering has had limited success mitigating some of the difficulty [92, 95]. Similarly, for grammar-based models, the grammar writer may have to balance writing linguistically motivated rules with domain-specific ones that allow for more optimal subdivisions for recognizer performance.

Perhaps more importantly, neither approach reasonably provides dialogue system designers with the ability to incorporate highly specific contextual information into the language model. For example, it is often the case that particular words or phrases may be salient: a particular list of airlines offered to a user, perhaps, or a list of restaurants that match a user's search request. Accommodating specific expected lexical items such as these would require partitioning the dialogue state into a huge number of states, each dependent on the particular set of salient proper nouns. Correctly recognizing utterances containing such contextually relevant proper nouns from one of a potentially large set can prove to be daunting – an observation confirmed in at least one conversational interface that is quite similar to *City Browser*, the CHAT system [91]. Nonetheless, this is an important problem in a wide variety of dialogue systems, for example multimodal interfaces to music players (*e.g.* [33, 54, 91]). The method developed in this chapter, in contrast, should be generally applicable to many domains involving large sets of proper nouns.

Another important limitation of both approaches is that each requires swapping

among several language models as a conversation with a user progresses. This may cause latency, or require large amounts of memory, as large language models are swapped. Dynamic classes, on the other hand, which are utilized by the approach described in this chapter, can be updated efficiently [73].

## 5.3 Contextualized Semantic Classes

This section presents *contextualized semantic classes* that, when they are incorporated as part of an $n$-gram language model, provide an approach to context-sensitive language modeling that overcomes many of the limitations discussed in the previous section. In particular, they do not require language models to be fragmented based on dialogue state; instead they make use of dynamic classes – which can be updated quickly and efficiently – and they allow system designers to incorporate very specific expectations about words or phrases that a user is likely to utter. Finally, as they are not themselves a state-specific solution, they have the potential to be mixed in with techniques that do create state-specific language models.

Contextualized semantic classes are simply a particular type of dynamic class in a class $n$-gram language model. They provide a mechanism for incorporating contextual information into the language model in real time, as a conversation progresses. Unlike the approaches outlined in the previous section, no dialogue-state-specific language model is required. Instead, a single class $n$-gram language model is used, where some or all of the classes can be dynamically updated. Contextualized semantic classes are "semantic" in the sense that the lexical items used as class expansions are related to one another semantically; *e.g.* each member of a class might be a restaurant name, city name, or the name of an airline. Classes are "contextualized" because tokens in the training corpus are tagged as members of the class whose members vary depending on the dialogue context.

Perhaps the best way to describe contextualized semantic classes is via an example. Figure 5-3 shows a (very small) corpus of transcribed utterances gathered from a conversational interface, and gives an example of several contextualized semantic classes tagged in this corpus: DESTINATION, OFFERED_TIME, and OFFERED_AIRLINE. Figure 5-3a shows the transcribed utterances, as they would appear in the corpus. Figure 5-3b shows how particular tokens and phrases in this corpus may be tagged with semantic class tags prior to training a typical class $n$-gram language model, specifically with the tags CITY, AIRLINE, ORDINAL, and DIGIT. Finally, Figure 5-3c shows how the same tokens can, instead, be tagged as members of contextualized semantic classes. A city name, in this example, may be tagged either as a CITY, or as a SOURCE or DESTINATION if it has already been mentioned as such in the conversation. An airline name may be tagged as an AIRLINE, PROMPTED_AIRLINE, or OFFERED_AIRLINE, depending on if either (a) no reference to an airline has yet been made, (b) the system has explicitly prompted for an airline name, or (c) it has offered a flight on a specific airline already. Finally, the number *two* in this example is tagged as an OFFERED_TIME, rather than as a DIGIT, since, given the context of the conversation, it is known to be a flight time.

| | |
|---|---|
| *U1*: | I'd like to fly from Oakland to Austin on the third. |
| *U2*: | Not Boston, Austin. On Northwest. |
| *U3*: | How about the flight at two on American. |
| | ... |

(a) A (very small) corpus of transcribed utterances.

| | |
|---|---|
| *U1*: | I'd like to fly from Oakland/CITY to Austin/CITY on the third/ORDINAL. |
| *U2*: | Not Boston/CITY, Austin/CITY. On Northwest/AIRLINE. |
| *U3*: | How about the flight at two/DIGIT on American/AIRLINE. |
| | ... |

(b) The same corpus, where some tokens have been replaced with typical semantic classes: CITY, AIRLINE, ORDINAL, and DIGIT.

| | |
|---|---|
| *S1*: | How may I help you? |
| *U1*: | I'd like to fly from Oakland/CITY to Austin/CITY on the third/ORDINAL. |
| *S2*: | Okay, from Oakland to Boston [*misrecognized*] on March third. Can you provide an approximate departure time or airline? |
| *U2*: | Not Boston/DESTINATION, Austin/CITY. On Northwest/PROMPTED_AIRLINE. |
| *S3*: | Okay, from Oakland to Austin on March third on Northwest. There are no flights on Northwest, but I've got a flight on American at two o'clock, would that work? Or I've got one on United at four thirty. |
| *U3*: | How about the flight at two/OFFERED_TIME on American/OFFERED_ARLINE. |
| | ... |

(c) The same corpus, where each transcribed utterance is shown properly *contextualized* as part of the conversation from which it was originally drawn. Tokens have been tagged with contextualized semantic classes, where appropriate. Notice that a city name may now be tagged as either CITY – if it is new to the conversation – or DESTINATION – if it has previously been mentioned as the destination of the flight. Similarly, an airline name may be tagged as either AIRLINE, PROMPTED_AIRLINE, or OFFERED_AIRLINE depending on whether the system prompted for an airline name, or already offered a flight on a specific airline.

Figure 5-3: A (very small) example corpus of transcribed utterances from an actual system interaction, tagged with contextualized semantic classes.

In order to tag contextualized semantic classes in the training corpus, each transcribed utterance must be accompanied by a system log indicating the dialogue system's state at the time of the recognized utterance. When tagging class membership in the training corpus, the log is used to determine how words and phrases should be tagged. For example, when utterance $U3$ is tagged, the log must indicate that a flight at 2:00 has just been offered in order to tag the token *two* as an OFFERED_TIME, rather than a DIGIT. Moreover, when the system is interacting with a user, it must be able to use the current state to include the token *two* in the list of class members for OFFERED_TIME immediately after it offers such a flight.

When the training corpus is tagged in this way, and the contextualized semantic class is populated appropriately at run-time depending on the context of the conversation, the likelihood of seeing utterance *U3* according to the language model changes depending on the context. In contrast, in a static language model, the likelihood of seeing this utterance never changes. The difference is well illustrated by looking at the likelihood of seeing just the single trigram "flight at two". In a static language model, this probability is expressed as follows:

$$p(flight\ at\ two) = p(DIGIT|flight\ at)p(two \in DIGIT) \tag{5.6}$$

The first term in the product is drawn from counting occurrences in the training corpus. The second term, typically, is drawn from a uniform distribution over the ten digits. Alternatively, it may be determined by training data, or set through some other mechanism.

When contextualized semantic classes are used, a new term is added to Equation 5.6:

$$\begin{aligned} &p(flight\ at\ two) = \\ &\quad p(DIGIT|flight\ at)p(two \in DIGIT) + \\ &\quad p(OFFERED\_TIME|flight\ at)p(two \in OFFERED\_TIME) \end{aligned} \tag{5.7}$$

It should be noted that the value of $p(DIGIT|flight\ at)$ will be different in Equations 5.6 and 5.7; it will be smaller in Equation 5.7 because it will have occurred less frequently in the training data. Instead, some of those training instances will have been tagged as OFFERED_TIME, causing some of the probability mass to shift to $p(OFFERED\_TIME|flight\ at)$.

During most of the user's interaction, the OFFERED_TIME class will be empty, as no flight times will have been offered. This means that:

$$p(two \in OFFERED\_TIME) = 0,$$

which, in turn, reduces the probability $p(flight\ at\ two)$. On the other hand, when the system offers several flights, as in utterance $S3$, the class OFFERED_TIME may now contain a few values; in the case of $S3$, the set of values might be:

$$OFFERED\_TIME = \{two, four\}$$

with each class member being equally likely. This increases $p(flight\ at\ two)$.

In this way, contextualized semantic classes such as OFFERED_TIME provide a simple mechanism for changing the probability of particular phrases – and, along with them, entire utterances – depending on the context of the conversation. Moreover, because the classes are tagged in the training corpus, the contextual shifts in probability mass are data driven. While it is up to the system designer to identify candidate contextualized semantic classes, once they have been chosen their distribution is based entirely on the training corpus.

### 5.3.1 Cues

Figure 5-3 shows two distinct types of contextualized semantic classes. In the first type, lexical items that had been verbally mentioned in the conversation were members of the classes DESTINATION, OFFERED_TIME, and OFFERED_AIRLINE. In such a case, only a very small set of lexical items is in the set; OFFERED_AIRLINE, for example, included only two possible airline names: *United* and *American*. On the other hand, the PROMPTED_AIRLINE class could include any airline; it was used instead of the AIRLINE class only because the system had just prompted the user for an airline.

More generally, contextualized semantic classes can be classified based on the expected size of the class; that is, based on the number of lexical items expected to be class members over the course of the conversation. The size of the class depends on the type of contextual *cue* that signals the use of the class. For example, the OFFERED_AIRLINE class is signaled by a *verbal* cue, because the system verbally mentioned the specific airlines, while the PROMPTED_AIRLINE class is cued by a prompt for an airline name. In a multimodal interface, *graphical* cues may also be used; for example, in the *City Browser* system, a list of hotel names might be displayed next to their locations on the map. Finally, in some interfaces, there may be *implicit* cues; for example, if a particular city has been mentioned in the conversation, all of the street names for streets in that city might be implicitly cued, in an application allowing geographical search. Figure 5-4 gives a rough indication of the relationship between the type of cue and the range in the number of lexical items expected to be in a given contextualized semantic class. Clearly, such constraints might not hold across all application domains; however, the ranges shown here correspond roughly to the two domains explored in the experiments described in the chapter that follows.

### 5.3.2 Scalability and Flexibility

An advantage of contextualized semantic classes over other context-sensitive language modeling techniques is that they do not require partitioning training data into state-specific sub-corpora. This section explores some of the advantages conferred because partitioning is not necessary. In particular, the scalability and flexibility of the technique are examined.

Contextualized semantic classes scale well because, as a dialogue system's capabilities are expanded, it is easy for system designers to add new classes as appropriate.

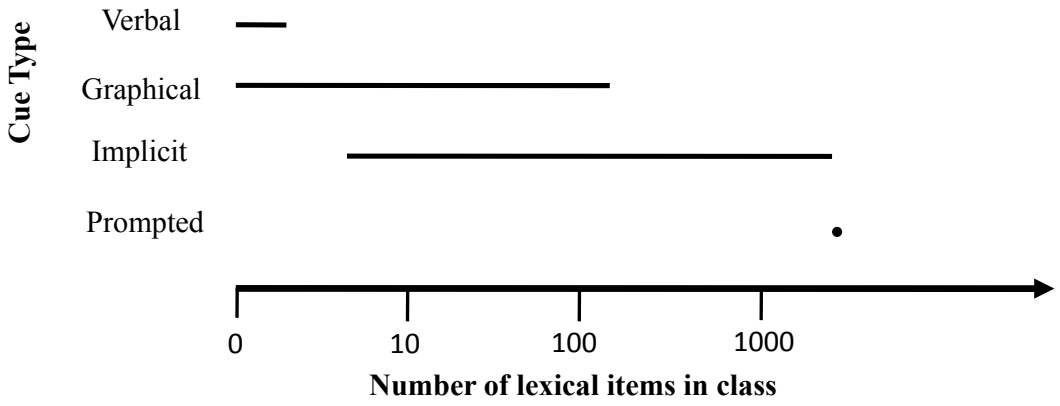



Figure 5-4: Potential ranges for the number of lexical items in a contextualized semantic class, as a function of the type of cue used to populate that class.

Adding new classes does not require changes to existing classes; each class is relatively independent, so adding new classes should generally have limited effect on unrelated classes. Moreover as new training data becomes available, new language models can easily be trained that incorporate new probabilities for $n$-grams containing contextualized semantic classes drawn from the new data.

Beyond scalability issues, contextualized semantic classes are much more flexible than state-specific solutions. Because each class is not tied to a particular dialogue state, classes can provide contextual expectations across a range of situations. For example, a flight-reservation system may prompt for an airline in several different ways; sometimes while prompting for other information as well. Imagine a dialogue system that could use any of the prompts for an airline name shown in Figure 5-5, depending on its decision of which is contextually most appropriate. After each of these prompts, the class PROMPTED_AIRLINE will be active, allowing for contextually relevant counts of $n$-grams involving airline names to be shared across each prompt. At the same time, other contextualized semantic classes may be active as appropriate, for each of the other prompts. In contrast, state-based approaches to context-sensitivity require a decision as to which of these prompts should be grouped together to share a state – forcing a trade-off between data sparsity and contextual expectation.

## 5.4 Conclusion

This chapter motivated the concept of context-sensitive language modeling, appealing to the intuition that contextual expectations about what a user is likely to say should potentially be useful in a speech recognizer's language model, in order to improve recognition accuracy. It discussed several context-sensitive language modeling techniques that make use of state-based language models. A new technique was then

| | |
|---|---|
| *P1* | What airline would you like to use? |
| *P2* | Could you tell me a preferred airline? Or your preferred departure airport in the San Francisco area? |
| *P3* | Could you provide an airline or departure time? |

Figure 5-5: Several prompts that include a request for an airline name. Contextualized semantic classes allow for exactly those $n$-grams containing an airline name to be shared after each of the prompts. State-based approaches, however, fragment each prompt into its own dialogue state, or group them together.

introduced that makes use of $n$-gram language models containing contextualized semantic classes, and therefore does not require the creation of state-based language models. Contextualized semantic classes are normal class $n$-gram language model classes, whose contents are updated dynamically, and whose membership depends on the context of the conversation. They provide a mechanism for integrating very specific contextual information into the speech recognizer's language model as the conversation progresses. In the next chapter, several empirical evaluations of the proposed technique are detailed.

# Chapter 6

# An Empirical Evaluation of Contextualized Semantic Classes

In this chapter, context-sensitive $n$-gram language models that make use of *contextualized semantic classes* are evaluated in a variety of conditions. In particular, two distinct sets of experiments are described, which explore the utility of a number of different contextualized semantic classes in two different conversational interfaces. The accuracies of the context-sensitive language models are compared with baselines that do not make use of contextual knowledge.

The first set of experiments puts into action the contextualized semantic classes given as examples in the previous chapter for the airline reservation domain. In particular, language models are trained and tested using a corpus of utterances collected from users interacting with the MERCURY flight reservation system [75, 77]. Several training data conditions are explored, meant to realistically emulate situations system developers typically confront. The second set of experiments focuses on proper noun usage in the *City Browser* system, and makes use of several of the corpora discussed in Chapter 4.

The experimental conditions explore the cue types shown in Figure 5-4 of Chapter 5. In particular, the MERCURY experiments explore contextualized semantic classes based on both *verbal* and *prompted* cues. In the *City Browser* domain, *graphical* and *implicit* cues are used.

## 6.1 Experiments in the Flight Reservation Domain

The MERCURY flight reservation system [75, 77] is the product of years of development at MIT. Available via the telephone, it is a robust interface that allows callers to make airline flight reservations. While strongly constrained by the transactional nature of making a flight reservation – users who interact with the system typically have a goal in mind involving where, when, and how they want to travel by air – the conversational interface is mixed-initiative. That is, while the system will helpfully prompt the user for information needed to make an airline booking, users may completely or partially ignore these prompts. They may, for instance, provide additional

information beyond what was requested, or even completely different constraints – such as a departure time instead of an airline name.

Its flexible mixed-initiative style makes the MERCURY system an excellent testbed for the use of contextualized semantic classes. MERCURY does not have explicit "dialogue states", so creating dialogue-state-specific language models would require partitioning its "information state" – in which it tracks the current conversational context – intro discrete states. Moreover, because users are free to ignore prompts or offer corrections at any time, such state-specific language models would have to be flexible enough to allow for such interaction.

The two experiments described in the remainder of this section explore the use of *verbal* and *prompted* cues for contextualized semantic classes. Both make use of a corpus of 26,886 utterances collected over several years of real user interaction with MERCURY. Each utterance was transcribed, and, via system log files, could be traced to the particular context from which it came. In particular, the system's "information state" at the time of the utterance was available, providing such information as the previous prompts, and any slot values – such as SOURCE, DESTINATION, or DATE – that had already been provided by the user.

### 6.1.1 Verbal Cues

The MERCURY corpus used here was collected via the telephone with no graphical user interface, meaning that the system must provide quite specific information verbally to the user. For example, the set of flights matching a user's search criteria must be narrowed down until they can be enumerated verbally. Such utterances cause particular city names, airline names, and times to be quite salient to the conversation; and there is a good chance that, after being offered, say, a flight on United Airlines, a user will say the word "united" in his or her response.

The MERCURY corpus was tagged with the following contextualized semantic classes using verbal cues like these:

- SOURCE: the city from which the user would like to travel,
- DESTINATION: the city to which the user would like to travel,
- OFFERED_AIRLINE: airlines providing the flights that have been offered,
- OFFERED_TIME: times of flights that have been offered to the user,
- OFFERED_NTHFLIGHT: phrases such as *the first one* and *the second one* that depend on the number of offered flights.

Figure 6-1 shows an example interaction with MERCURY, with user utterances tagged appropriately.

| | |
|---|---|
| *S1*: | How may I help you? |
| *U1*: | I'd like to fly from Oakland/CITY to Austin/CITY on the third/ORDINAL. |
| *S2*: | Okay, from Oakland to Boston [*misrecognized*] on March third. Can you provide an approximate departure time or airline? |
| *U2*: | Not Boston/DESTINATION, Austin/CITY. On Northwest/AIRLINE. |
| *S3*: | Okay, from Oakland to Austin on March third on Northwest. There are no flights on Northwest, but I've got a flight on American at two o'clock, would that work? Or I've got one on United at four thirty. |
| *U3*: | How about the flight at two/OFFERED_TIME on American/OFFERED_ARLINE. |
| | ... |

(a) Example interaction, $S$ indicates system utterance and $U$ indicate user utterance

```
:reply {
 :prompt "welcome"
 }
```

| | |
|---|---|
| *SOURCE* | -empty |
| *DESTINATION* | -empty- |
| *OFFERED TIME* | -empty- |
| *OFFERED AIRLINE* | -empty- |
| *OFFERED NTHFLIGHT* | -empty- |

(b) Information state subset and class members associated with utterance *U1*.

```
:date "March3"
:source "OAK"
:destination "AUS"
:reply {
 :prompt "airline"
 :prompt
   "departure_time" }
```

| | |
|---|---|
| *SOURCE* | "oakland" |
| *DESTINATION* | "austin" |
| *OFFERED TIME* | -empty- |
| *OFFERED AIRLINE* | -empty- |
| *OFFERED NTHFLIGHT* | -empty- |

(b) Information state subset and class members associated with utterance *U2*.

```
:date "March3"
:source "OAK"
:destination "AUS"
:reply {
 :best_departure {
  :time "2:00pm"
  :airline "AA" }
 :second_departure {
  :time "4:30pm"
  :airline "UA" } }
```

| | |
|---|---|
| *SOURCE* | "oakland" |
| *DESTINATION* | "austin" |
| *OFFERED TIME* | "two", "two o'clock", "two p_m", "two o'clock p_m", "four thirty", "four thirty p_m" |
| *OFFERED AIRLINE* | "united", "united airlines", "american", "american airlines" |
| *OFFERED NTHFLIGHT* | "the first one", "the first flight", "the second one", "the second flight" |

(d) Information state subset and class members associated with utterance *U3*.

Figure 6-1: An example interaction with MERCURY, where user utterances have been tagged with contextualized semantic classes created with *verbal* cues. Class membership is based on MERCURY's internal information state at the time of each user utterance. The information state associated with each user utterance is shown, as are the members of contextualized semantic classes generated from that information state. The class members are used both to tag class members in the training set and to specify class membership during recognition of each test utterance.

| Specific Prompts | Open Prompts | Example |
|---|---|---|
| PROMPTED_AIRLINE_S | PROMPTED_AIRLINE_O | united |
| PROMPTED_CITY_S | PROMPTED_CITY_O | boston |
| PROMPTED_STATE_S | PROMPTED_STATE_O | massachusetts |
| PROMPTED_COUNTRY_S | PROMPTED_COUNTRY_O | france |
| PROMPTED_CITYSTATE_S | PROMPTED_CITYSTATE_O | boston massachusetts |
| PROMPTED_CITYCOUNTRY_S | PROMPTED_CITYCOUNTRY_O | paris france |

Table 6.1: Contextual semantic classes cued by system prompts. Specific prompts are those that prompt explicitly for a city or airline name. Open prompts are open ended, for instance *How may I help you?*

### 6.1.2 Prompt cues

In addition to explicitly mentioning lexical items, MERCURY also prompts users for particular values; for example, in utterance *S2* of Figure 6-1 the user is prompted to provide an airline or time. Similarly, a user is quite likely to provide an airline or city name after encountering the welcome message in *S1*. In both cases, it is difficult to predict which airline or city name a user is likely to say, however it is likely (though not required) that the user's utterance will contain an airline or city name given the prompt. By using these *prompt* cues, contextualized semantic classes that contain all proper nouns in a given class – for example all city or airline names – can be created.

Table 6.1 shows the classes created for the MERCURY domain. Classes were created for airlines, cities, states, countries, city/state combinations, and city/country combinations. Moreover, two prompt conditions were used to create two versions of each class: *specific* and *open*. The *specific* case occurs when the system explicitly prompts for a class member; as in utterance *S2* of the example. The *open* case covers utterances following an open-ended prompt like *S1*. Figure 6-2 shows the same example dialogue as in Figure 6-1. In this case, it has been tagged using contextual semantic classes based on prompt cues. Classes active during the recognition of each user utterance are also shown – where an "active" class is simply one that has a non-empty list of members.

### 6.1.3 Experimental Conditions

The two types of contextualized semantic classes – derived from *verbal* and *prompt* cues respectively – were evaluated by splitting the MERCURY corpus into random training and test sets of 24,815 and 2,071 utterances respectively. A baseline class trigram language model was trained, in which semantic class membership was determined purely based on static lists; the classes included, for example, airlines, cities, and digits such as *one* and *two*. Trigram language models making use of the two types of contextualized semantic classes were trained separately, by tagging the training corpus with class membership using contextual information, as described above. In all cases, the language model had a base vocabulary size of 1,586 words – however, this vocabulary count excludes semantic class members, notably airline and city names, which can add hundreds or thousands of additional words to the vocabulary, depend-

| | |
|---|---|
| *S1*: | How may I help you? |
| *U1*: | I'd like to fly from Oakland / PROMPTED_CITY_O to Austin / PROMPTED_CITY_O on the third/ORDINAL. |
| *S2*: | Okay, from Oakland to Boston [*misrecognized*] on March third. Can you provide an approximate departure time or airline? |
| *U2*: | Not Boston/CITY, Austin/CITY. On Northwest/PROMPTED_AIRLINE_S. |
| *S3*: | Okay, from Oakland to Austin on March third on Northwest. There are no flights on Northwest, but I've got a flight on American at two o'clock, would that work? Or I've got one on United at four thirty. |
| *U3*: | How about the flight at two/DIGIT on American/AIRLINE. |
| | ... |

(a) Example interaction, *S* indicates system utterance and *U* indicate user utterance

```
:reply {
 :prompt "welcome"
 }
```

| **Active classes** |
|---|
| PROMPTED_AIRLINE_O |
| PROMPTED_CITY_O |
| PROMPTED_STATE_O |
| PROMPTED_COUNTRY_O |
| PROMPTED_CITYSTATE_O |
| PROMPTED_CITYCOUNTRY_O |

(b) Information state subset and active classes associated with utterance *U1*, an open prompt.

```
:date "March3"
:source "OAK"
:destination "AUS"
:reply {
 :prompt "airline"
 :prompt
   "departure_time" }
```

| **Active classes** |
|---|
| PROMPTED_AIRLINE_S |

(c) Information state subset and active classes associated with utterance *U2*, a specific prompt.

```
:date "March3"
:source "OAK"
:destination "AUS"
:reply {
 :best_departure {
  :time "2:00pm"
  :airline "AA" }
 :second_departure {
  :time "4:30pm"
  :airline "UA" } }
```

| **Active classes** |
|---|
| -none- |

(d) Information state subset and active classes associated with utterance *U3*.

Figure 6-2: An example interaction with MERCURY, where user utterances have been tagged with contextualized semantic classes created with *prompt* cues. Class membership is based on the previous prompt, as determined by MERCURY's information state. The information state associated with each user utterance is shown, as is the list of active *prompt*-cued contextualized semantic classes.

| | |
|---|---|
| **Corpus/Medium** | Within-class weights for *CITY*, *CITY_STATE*, and *AIRLINE* based on **corpus** counts; **medium** vocabulary size of: 516 city names, 329 city_states, 68 airlines |
| **Uniform/Medium** | Within-class weights **uniform**; **medium** vocabulary. |
| **Population/Large** | Within-class weights for *CITY*, *CITY_STATE* based on **population**, *AIRLINE* uniform; **large** vocabulary: 16,956 city names, 25,334 city_states, 68 airlines. |
| **Uniform/Large** | Within-class weights **uniform**; **large** vocabulary. |

Table 6.2: Experimental conditions for MERCURY experiments in which the number of cities supported by the system, and the within-class weights for those city names are varied.

ing on the number of lexical items in the class.

In addition, each language model was tested in four conditions, meant to simulate typical system design situations. The four conditions were created by varying two parameters: first, the manner in which the within-class weights were calculated for some classes, and second, the number of lexical items in a class. Three ways of calculating the within-class weights were explored: the first looked at how often each class member appeared in the training corpus, the second gave uniform weights to each class member, and the third weighted *CITY*, *CITY_STATE*, and the associated contextualized semantic classes based on each city's population. In tandem with varying the within-class weights, the number of lexical items in the *CITY* and *CITY_STATE* classes was varied. The conditions are summarized in Table 6.2.

The four conditions were chosen because they simulate conditions commonly confronted by dialogue-system designers. The corpus/medium condition is a developer's ideal situation: a large training corpus well matched with the test set. However, such an ideal situation is not common. More typically, class membership is assumed to be uniform, as in the uniform/medium condition. Moreover, data collected with an initial version might only include a few cities, while a newer system might support many more. This situation is reflected in the uniform/large and population/large conditions; in the first, the system designer allows all cities in the larger set to occur with equal probability; in the second, the designer users population as a proxy to estimate the within-class weights.

### 6.1.4 Experimental Results

Table 6.3 reports word error rates on the test set for the four conditions. All conditions show that both *verbal* and *prompt* cues offer an improvement compared to the baseline; however, this difference is not statistically significant in the Corpus/Medium condition – where the training and test data are perfectly matched and the class vocabulary size is not large. When within-class weights are not available from the corpus and/or the vocabulary size increases, the contextualized semantic classes lead to larger, statistically significant reductions in word error rate.

In addition to the overall word error rate, it is useful to break the test set into

| Condition | | Cue Type | | |
|---|---|---|---|---|
| **Weights** | **Vocabulary** | **None** | **Verbal** | **Prompt** |
| Corpus | Medium | 17.8 | 17.7 | 17.6 |
| Uniform | Medium | 25.2 | 24.4* | 20.8** |
| Population | Large | 27.1 | 26.7* | 26.0* |
| Uniform | Large | 46.7 | 45.0* | 42.1** |

Table 6.3: Word error rates for the baseline class $n$-gram language model, and two context sensitive language models incorporating contextualized semantic classes derived from *verbal* and *prompt* cues respectively. Word error rates on the test set are shown in four conditions. Starred entries (*) are a statistically significant improvement over the baseline. Double stars (**) indicate instances where the prompt cues show a statistically significant improvement over the verbal cues.

| | Condition | | Active | | Inactive | |
|---|---|---|---|---|---|---|
| | **Weights** | **Vocabulary** | **None** | **Verbal** | **None** | **Verbal** |
| **Source, Destination** | Corpus | Medium | 19.5 | 19.2 | 16.0 | 16.0 |
| | Uniform | Medium | 29.4 | 27.8* | 20.6 | 20.7 |
| | Population | Large | 30.1 | 29.1* | 23.9 | 24.0 |
| | Uniform | Large | 52.5 | 50.1* | 40.4 | 39.4* |
| **Time, Airline, Nth Flight** | Corpus | Medium | 18.6 | 16.8* | 17.7 | 17.8 |
| | Uniform | Medium | 32.0 | 25.3* | 24.1 | 24.2 |
| | Population | Large | 30.4 | 27.6* | 26.6 | 26.5 |
| | Uniform | Large | 52.9 | 47.1* | 45.7 | 44.6* |

Table 6.4: Word error rates of the baseline class $n$-gram language model and one incorporating contextualized semantic classes derived from *verbal* cues. Error rates are broken down by sets of utterances in which verbally-cued classes are active or inactive. Starred entries (*) are a statistically significant improvement.

two groups, based on whether the contextualized semantic classes were *active* or *inactive*. The active group, for a particular contextualized semantic class, is the group of utterances where that class was non-empty; the inactive group is the ones where it was empty. For example, in the example dialogue in Figure 6-1, user utterances would be grouped as follows considering the classes derived from *verbal* cues:

U1: All contextualized semantic classes inactive.

U2: SOURCE and DESTINATION active, others inactive.

U3: SOURCE, DESTINATION, OFFERED_TIME, OFFERED_AIRLINE, and OFFERED_NTHFLIGHT active

It should be noted that, in these examples, some of the class members appear in the actual utterances – *Boston*, *two*, and *American* – however, this is not requisite. Indeed, in *U2*, SOURCE is active, despite the fact that *Oakland* does not appear in the utterance. Such utterances are those where it is anticipated that salient phrases will appear, not utterances in which they necessarily do appear; they are not chosen by an "oracle".

Table 6.4 compares results over two subsets of utterances where verbal cue classes are active: those where one or both of SOURCE and DESTINATION are active, and those where one or more of OFFERED_TIME, OFFERED_AIRLINE, and OFFERED_NTHFLIGHT are active. This is a useful distinction because SOURCE and DESTINATION tend to be less specific contextual indicators than the other classes. Indeed, throughout much of a typical conversation, either SOURCE or DESTINATION will be active, as the source and destination of a flight are usually settled early on. Hence, while active utterances in this category comprised 60% of the test set, they accounted for only relatively small improvements in word error rate. Conversely, OFFERED_TIME, OFFERED_AIRLINE, and OFFERED_NTHFLIGHT tend to be active most often just after the system has offered the user a selection of multiple flights. Therefore they are strongly salient, and the user is highly likely to invoke one of the phrases in these dynamic classes to select a flight. Thus, while active utterances in this category constituted only 13% of the test set, they saw a 14.4% decrease in error rate in the Corpus/Medium case, a statistically significant difference. This suggests that a system with many strong contextualized cues like these – and hence a greater number of utterances where such contextualized classes are active – would experience a larger reduction in overall word error rate. This is confirmed by the effects seen in the inactive group: each condition saw either a statistically significant reduction in word error rate, or no significant change – confirming that the gains for the utterances in the active set do not come at the expense of those in the inactive set.

Table 6.5 reports the results of the same analysis, as applied to the *prompt*-cued classes. In this case, improvements were generally seen in both sets, with larger reductions in word error rate seen in the inactive set. This can be explained by the fact that, here, the active set is most like the baseline in that airlines and locations are fairly common in both; conversely, in the inactive set, locations and airlines are

| Condition | | Active | | Inactive | |
|---|---|---|---|---|---|
| **Weights** | **Vocabulary** | **None** | **Prompt** | **None** | **Prompt** |
| Corpus | Medium | 14.7 | 14.4 | 19.5 | 19.3 |
| Uniform | Medium | 18.9 | 17.0* | 28.7 | 22.9* |
| Population | Large | 23.6* | 27.2 | 29.1 | 25.4* |
| Uniform | Large | 38.4 | 34.8* | 51.2 | 46.1* |

Table 6.5: Word error rates of the baseline class $n$-gram language model and one incorporating contextualized semantic classes derived from *prompt* cues. Error rates are broken down by sets of utterances in which prompt cued classes are active or inactive.

*not* likely to be mentioned, appearing relatively infrequently in the training data. Without the contextualized versions of these classes active, the recognizer is much less likely to hypothesize one of these proper nouns, leading to a significant reduction in word error rate.

The Population/Large condition, however, is anomalous. Analysis shows that almost all of the increase in word error rate came when context-sensitive classes cued by "specific" prompts were active. It may be the case that the population model heavily weights some cities that appear very infrequently in the test set – it is an imperfect "proxy" for true corpus counts. Since most of the data was collected from users in the Boston area, this led to a bias. Since the contextualized classes in this case boost the probability of seeing a location, like a city name, over the baseline, it may be this case in which mismatched within-class weights decrease accuracy the most.

## 6.2 *City Browser* Experiments

Several language modeling experiments using data gathered from the *City Browser* system were performed as well, in this case looking at contextualized classes based on *graphical* and *implicit* cues. In each case, as in the MERCURY experiments described in the previous section, a single $n$-gram language model was created that incorporated contextualized semantic classes. However, *City Browser* differs from MERCURY in three key respects:

1. The task is more open-ended: users might be looking for a particular restaurant or hotel, or they might want to explore what's available in a given city, or near a particular address. They might want driving directions, or they might not. In the MERCURY system, on the other hand, all users call to book a flight reservation.

2. The interface is multimodal, which allows users to browse large lists of restaurants and hotels, for example, rather than just consider a few options listed verbally.

3. There are few prompts, and very little directed dialogue; the interface is designed such that the user almost always takes the initiative in the conversation. Instead of prompting the user to fill in values, *City Browser* provides a context-sensitive list of suggested utterances (see Chapter 9).

Given these interface differences, the *prompt-* and *verbal*-cued contextualized semantic classes used in MERCURY are not readily applicable to *City Browser*. Instead, the experiments in this section explore the use of *graphical* and *implicit* cues.

### 6.2.1 Graphical and Implicit Cues

Graphical cues are based on what the user sees in the graphical user interface. In the case of *City Browser*, the most prominent graphical cue is the list of database results shown to the user, for example the list of hotels or restaurants displayed to the user in response to a spoken search query. These search results are depicted in a list in the interface, as well as shown on the map. Users can then ask for more information about particular search results, get directions to one, and so forth. Thus, the names of the displayed results are excellent candidates for a contextualized semantic class.

The *City Browser* system also provides an *implicit* cue: search results are often clustered geographically – *e.g.*, in a particular city. The location of the search results, then, can be used to cue the names of streets that are near the search results. In particular, *City Browser* uses a contextualized semantic class to contain the names of all streets in all cities that contain visually displayed search results. Generally, this should roughly correspond to streets that are near the set of returned search results.

Figure 6-3 shows an example dialogue with *City Browser*, with contextualized semantic classes tagged appropriately. The dialogue illustrates how users may naturally refer to the name of a restaurant or hotel visible on the screen; similarly, an implicitly cued street name is used to narrow down the search results.

### 6.2.2 Experiments on the *Tablet* corpus

Several exploratory experiments were performed using the small *Tablet* corpus, which is described in Chapter 4. In this corpus, subjects used an early prototype of *City Browser* to find four appealing restaurants, in various metropolitan areas. Each time the user switched to a new metropolitan area, proper names for that area were loaded into the language model; this means that instead of a strictly "static" baseline language model like that used in the MERCURY experiments, experiments with *City Browser* data use a "per-metro" baseline language model, where proper names for the metropolitan area in use are loaded.

The early *City Browser* prototype did not support hotels or museums, so the contextualized semantic classes were used only for restaurant and street names. The set of restaurant names is actually larger than the number of restaurants in a particular metropolitan area, as alternative ways of referring to each restaurant are also included. For instance, *Caprice Restaurant and Lounge* can also be referred to as

| | |
|---|---|
| *S1*: | How may I help you? |
| *U1*: | Show me Chinese restaurants near 77 Massachusetts/STREETNAME avenue in Cambridge/CITY. |
| *S2*: | There are 10 Chinese restaurants near 77 Massachusetts avenue in Cambridge. [shows a list] |
| *U2*: | Tell me about the one on Main/I_STREETNAME street. |
| *S3*: | Royal East is located on 792 Main Street in Kendall Square in Cambridge. It serves reasonably priced Chinese food. |
| *U3*: | Give me directions to the Royal East/G_RESTAURANT. |
| *S4*: | Here are directions from this location to the Royal East. [shows directions on the map] |
| *U4*: | Can you show me hotels in Cambridge/CITY? |
| *S5*: | There are 28 hotels in Cambridge. The particular neighborhoods are Harvard Square, Central Square, and Kendall Square. [shows a list] |
| *U5*: | Give me the phone number of the Charles/G_HOTEL. |

Figure 6-3: An example interaction with *City Browser*, where user utterances have been tagged with the appropriate contextualized semantic classes, based on *graphical* and *implicit* cues. G_RESTAURANT, and G_HOTEL are graphically-cued contextualized semantic classes, while I_STREETNAME is implicitly cued.

*Caprice Restaurant* or simply *Caprice*. This set of aliases is semi-automatically created, as described in [38], yielding name sets containing 5,860 to 11,247 aliases, depending on the metropolitan area. The within-class probability for the set of restaurant name aliases was uniform.

The set of street names in each metropolitan area was created by taking the names of streets and dropping suffixes such as "street" and "road". The fact that street names are often reused significantly reduces the size of the set, yielding from 1,177 to 2,243 street names per metropolitan area. The within-class weights used for street names were created using a heuristic formula that first assigned a count of 1 to each street, then added the number of restaurants on that street, and finally normalized the weights to sum to 1.

Because the *Tablet* corpus contained only 546 utterances from 10 users, language model experiments were performed via a jack-knife procedure where training data collected during system development was combined with transcripts from 9 out of 10 subjects to create a trigram language model to be tested on the held-out utterances from the single user. Table 6.6 compares the word error rates and the proper noun error rates for the two language models. It is quite interesting that, although the word error rate is higher for the language model with contextualized semantic classes, its proper noun error rate is much lower. Note that as this was an exploratory experiment and the set of data was quite small, statistical significant testing was not performed.

One hypothesis for this discrepancy is that, while the contextualized semantic classes do help when users mention proper names, they also cause a fragmentation of the language model training data. When there is very little training data available, as in these experiments, this fragmentation may well lead to reduced accuracy. In an attempt to understand if more training data would help, a second experiment

| | Per-Metro | Contextualized |
|---|---|---|
| **Word Error Rate** | 24.4 | 27.1 |
| **Proper Noun Error Rate** | 59.0 | 49.5 |

Table 6.6: Word error rate and proper noun error rates for the per-metro language model and the language model with graphically- and implicitly-cued contextualized semantic classes.

| | Per-Metro | Contextualized |
|---|---|---|
| **Word Error Rate** | 22.1 | 21.6 |
| **Proper Noun Error Rate** | 56.4 | 44.1 |

Table 6.7: Word error rate and proper noun error rates – when synthetic training data is included – for the per-metro language model and the context-sensitive language model with graphically- and implicitly-cued contextualized semantic classes.

was performed in which the transcripts were augmented with *synthetic* training data, generated using hand-crafted templates created by examining the syntactic patterns appearing in each user's collected data. Synthetic utterances were then created during the leave-one-out procedure, using only templates derived from transcripts in the training set.

Table 6.7 shows error rates in this condition; the word error rates are now comparable, and, while the proper noun error rate improves in both conditions, the improvement is larger when using the language model with contextualized semantic classes. This indicates that the effectiveness of the contextualized semantic classes is likely to increase when more real training data is available.

### 6.2.3 Experiments on the *Car-Pilot* and *Car* corpora

A follow-up set of experiments was performed using the larger amount of data available in the *Car-Pilot* and *Car* corpora, described in Chapter 4. In particular, language models were trained using the *Car-Pilot* corpus and then tested using the *Car* corpus – meaning that the training and test set were quite well matched. As in the previous section, two language models were compared: a static "per-metro" language model, and one built using contextualized semantic classes like those shown in Figure 6-3. In particular, graphically-cued semantic classes were used for restaurant, museum, and hotel names; and an implicitly cued class was created for street names. Finally, since this corpus was collected using only tasks in the Boston metropolitan area, the "per-metro" language model was, in fact, truly a static language model. There were 8,534 restaurant names, 831 hotel names, 89 museum names, 191 city names, and 1,177 unique street names.

Table 6.8 shows error rates for the two language models. The contextualized semantic classes led to a modest, statistically significant reduction in word error rate and a larger reduction in proper noun error rate. This indicates that the repaired

| | Per-Metro | Contextualized |
|---|---|---|
| **Word Error Rate** | 26.6 | 26.0* |
| **Proper Noun Error Rate** | 50.1 | 44.3 |

Table 6.8: Word error rates and proper noun error rates for the per-metro language model and the context-sensitive language model with graphically- and implicitly-cued contextualized semantic classes. Trigram language models were trained on the *Car-Pilot* corpus, and tested using the *Car* corpus. A star (*) indicates a statistically significant difference in word error rates.)

word errors were generally the intended ones: proper nouns.

## 6.3 Conclusion

Contextualized semantic classes based on several types of cues were evaluated using data collected from real conversational interfaces: MERCURY and *City Browser*. In several sets of experiments, language models making use of contextualized semantic classes led to improved recognition accuracy compared to baselines, which did not dynamically incorporate contextual expectations into the language model. Moreover, in the case of *City Browser*, it was shown that the reductions in word error rate were generally related to semantically important proper names. Understanding these words is critical to the system responding correctly; however, they are often misrecognized.

As future work, it would be interesting and useful to develop contextualized semantic classes for other domains, as well as to integrate additional ones into each of the two domains studied in this chapter. It would also be useful to explore methods of automatically identifying and tagging such classes using transcripts paired with dialogue system logs. This would free the system designer from having to manually devise each class. Instead, an automatic, or semi-automatic, procedure could be used that would learn to map particular features in the system state to particular words (or existing $n$-gram classes) in the transcript.

# Chapter 7

# Context-Sensitive Confidence Scoring

Despite efforts to help users speak within the bounds of what a conversational system can understand (see Chapter 9) and to update the speech recognizer's language model to reflect expectations about what users are likely to say (see Chapters 5 and 6), speech recognition accuracy in a conversational interface will almost certainly be far from perfect. Even with context-sensitive language modeling techniques, the word error rates reported in Chapter 6 for the MERCURY and *City Browser* conversational interfaces range from about 18% to 42% depending on the condition. Such high word error rates are not unique to these systems: Table 7.1 lists error rates reported for several conversational interfaces of similar complexity. The unfortunate reality is that, quite often, a speech recognizer's best hypothesis as to the words that a user has spoken will contain a significant number of errors.

Because recognition errors are unavoidable, speech understanding systems traditionally rely on *confidence scores* assigned by the speech recognizer to judge the likelihood that errors have been made, and the severity of those errors. Recognition confidence scores are supposed to be a measure of how errorful a particular hypothesis is likely to be: low confidence scores indicate a high likelihood of errors. They are

| **WER** | **System** |
|---|---|
| 64.3%† | [70] Let's Go |
| 40.4% | [49] MATCH City Guide |
| 56% | [50] Home Entertainment Browser |

† Mean session-average WER

Table 7.1: Word error rates (WER) for several research prototype spoken or multimodal conversational interfaces. Note that the systems have different vocabulary sizes, language modeling requirements, and interact with users in different acoustic environments, so their error rates are not directly comparable. Nonetheless, the WERs are collectively indicative of the challenge of accurately recognizing speech in conversational interfaces.

generally produced via a supervised learning process in which a model is trained using acoustic and lexical features, and error rates calculated using transcribed utterances. Typical conversational interfaces choose to either *accept* (that is, attempt to understand and respond to) or *reject* (that is, respond to the user with an indication of non-understanding) a user utterance by comparing the recognition confidence score to a rejection threshold.

If the goal is simply to present a user with a transcription of his or her utterance, then training a confidence model based only on word or sentence error rates makes perfect sense. However, in a conversational interface, speech recognition is just the first in a series of processes aimed at understanding the utterance in context to generate an appropriate response. Typically, the recognition hypothesis is parsed, interpreted in context, and a response is produced, which is spoken – and/or presented visually – to the user. From the user's perspective, the system's response is what is truly important, regardless of whether every word in his or her utterance was accurately transcribed. Sometimes, a recognition hypothesis containing many errors may still result in an appropriate system response; conversely, a recognition hypothesis with just a single error may evoke an incorrect response, if understanding the utterance hinges on the misunderstood word.

Thus, the usefulness of an errorful transcription is highly dependent on the type of error made, the robustness of the conversational interface with regard to that type of error, and the context of the conversation. For instance, imagine the following utterance and associated recognition hypothesis uttered to *City Browser*:

| | |
|---|---|
| Utterance: | *directions to thirty two vassar street* |
| Hypothesis: | *is thirty two vassar please* |

The word error rate of this hypothesis is 50%: there are three errors – two substitutions and a deletion – and six words in the transcript. Given this high error rate, a confidence scoring module trained only on word error rate would, ideally, give this hypothesis a low score.

Nevertheless, a key piece of information is still largely intact in the errorful hypothesis, namely the address to which the user would like directions. Whether and how a conversational interface might make use of that information largely depends both on the capabilities of the interface and the context in which it is uttered. A system that is only able to parse fully grammatical utterances, for example, might not be able to extract the address, as the hypothesis taken as a whole is not grammatical in an obvious way. For such an interface, a low confidence score may be quite appropriate.

On the other hand, *City Browser* uses a robust parser that can find understandable fragments in an otherwise ungrammatical recognition hypothesis; in this case, it will recognize that the fragment *thirty two vassar* is an address. Depending on the context of the conversation, interpreting just the fragment may (or may not) lead to the system providing a correct response. Figure 7-1 shows an example in which the errorful hypothesis shown above is placed in the context of two different conversations. In Figure 7-1(a), interpreting just the address fragment in context leads to the correct system response, whereas in Figure 7-1(b), it leads to an incorrect response. Since

| | |
|---|---|
| *U1:* | Show me Chinese restaurants in Cambridge. |
| *S1:* | There are 10 Chinese restaurants in Cambridge... [shows on map] |
| *U2:* | Directions from the Royal East restaurant please. |
| *S2:* | Please tell me to where you would like directions. |
| *U3:* | Directions to thirty two Vassar street. (**is thirty two vassar please**) |
| *S3:* | Here are directions from Royal East to thirty two Vassar street. [shows directions on the map] |

(a)

| | |
|---|---|
| *U1:* | Show me Chinese restaurants in Cambridge. |
| *S1:* | There are 10 Chinese restaurants in Cambridge... [shows on map] |
| *U2:* | What is the phone number of the Royal East restaurant? |
| *S2:* | The phone number of the Royal East is 555-5555. |
| *U3:* | Directions to thirty two Vassar street. (**is thirty two vassar please**) |
| *S3:* | Here is thirty two Vassar street. [shows address on the map] |

(b)

Figure 7-1: Two example conversations in which utterance *U3* might sensibly appear; in each case, a potential recognition hypothesis with 50% word error rate is shown in bold below it. In (a) the errorful hypothesis still leads to the correct system response of giving directions, whereas in (b) the system's response of showing the address on the map is not correct. Note that $U$ indicates user utterances, while $S$ indicates system responses.

speech recognition confidence scores are assigned to the recognition hypothesis and are trained using word error rates, they provide no means of distinguishing cases (a) and (b). But, if confidence scoring is deferred until after the hypothesis has been interpreted in context, then it may be possible to distinguish these two cases. In particular, if a candidate system response is produced based on the hypothesis, then a confidence score can be assigned to that *response*, which indicates the likelihood that it is acceptable. In this way, the problem of assigning a confidence score to an input hypothesis shifts to one of providing a score for a particular response.

Furthermore, while the previous example considers just the top, or most likely, hypothesis from the speech recognizer's N-best list, generating candidate system responses from multiple hypotheses on the N-best list may provide more evidence that a particular response is correct. Figure 7-2 considers the responses produced by interpreting each of the 5-best hypotheses for the same utterance, given the two conversational contexts given in Figure 7-1. Intuitively, it seems that since each hypothesis evoked the same system response in context (a), the system ought to have a relatively high confidence that this response is correct. On the other hand, in (b), two unique responses were generated, meaning that perhaps the system ought to have lower confidence in the response produced by the top hypothesis. Moreover, perhaps it might consider choosing the most frequent response, rather than the one generated by the top ranking hypothesis.

The remainder of this chapter presents a method for formalizing this intuition that the confidence score ought to be assigned to the system response rather than the recognition hypothesis, and that it may be beneficial to explore multiple candidate

| | Hypothesis | Response in Context (a) | Response in Context (b) |
|---|---|---|---|
| 1 | is thirty two vassar please | Here are directions from Royal East to thirty two Vassar street | Here is thirty two Vassar street |
| 2 | directions thirty two vassar please | Here are directions from Royal East to thirty two Vassar street | Here are directions from Royal East to thirty two Vassar street |
| 3 | directions to thirty two vassar | Here are directions from Royal East to thirty two Vassar street | Here are directions from Royal East to thirty two Vassar street |
| 4 | to thirty two vassar please | Here are directions from Royal East to thirty two Vassar street | Here are directions from Royal East to thirty two Vassar street |
| 5 | thirty two vassar street | Here are directions from Royal East to thirty two Vassar street | Here is thirty two Vassar street |

Figure 7-2: The 5-best recognition hypotheses for the utterance *Directions to thirty two Vassar street* and the system response generated by each one depending on whether each is interpreted in the context of conversation (a) or (b) from Figure 7-1 above.

system responses. In particular, a *response confidence scoring* module will be developed that generates a number of candidate system responses to a user's utterance by processing multiple hypotheses from the speech recognizer's N-best list, and then assigns a confidence score to each candidate response meant to reflect the probability that the response is an appropriate one given the conversational context. Features used to make the decision are obtained not only from the speech recognizer, but also from the process of generating each candidate response, and from the distribution of the candidate responses themselves. These features are used to train a Support Vector Machine (SVM) [84] to identify acceptable responses. When given a novel utterance, candidate responses are ranked with confidence scores output from the SVM. Based on the scores, the system can then either respond with the highest-scoring candidate, or reject all of the candidate responses and respond by indicating non-understanding.

## 7.1 Related Work

There are several areas of research that are relevant to the response confidence scoring approach discussed in this chapter. First, recognition confidence scoring techniques have been explored in the context of many speech recognition systems. Second, research into building conversational interfaces has resulted in confidence scoring approaches that take into account information from the language understanding phase, as does the one proposed in this chapter. Third, stochastic dialogue management strategies overlap in their goals and methods, as they use data-driven approaches to decide how to respond to a user. Finally, the general category of *multimodal disambiguation* techniques for multimodal interfaces can perhaps be seen as including the confidence scoring module discussed in this chapter.

### 7.1.1 Recognition Confidence Scores

There has been much research into deriving utterance-level confidence scores based on features derived from the process of speech recognition. For instance, the utterance-level confidence module used as a baseline in the experiments in the next chapter is described in [43]. In it, confidence scores are derived by training a linear projection model to differentiate utterances with high word error rates from those that have no or few errors. The classifier is trained using 15 features drawn from the speech recognition process. These features are computed using the acoustic score, lexical score, and total score of each hypothesis in the N-best list, as well by counting the words in each hypothesis and examining the "purity" of words in the N-best list (that is, how often a particular word appears in a particular position across all hypotheses). The utterance-level confidence scores are used to decide whether or not the entire utterance should be accepted or rejected.

Other methods of confidence scoring that make use of features from the speech recognition process have also been explored – see, for instances [72, 12]. In [72], features are derived from language model back-off, language model score, and phonetic word length to train both a multi-layered perceptron and a decision tree to distinguish among in-domain and out-of-domain utterances. Some confidence scoring approaches, as in [12, 43], also provide per-word confidence scores. As a result of this research, commercially available speech recognizers today, for the most part, provide some form of confidence scoring.

### 7.1.2 Language Understanding

Research with regards to judging the quality of a recognition hypothesis in the context of a conversational interface has focused on ways to incorporate information from the language understanding phase. For instance, in [60], when confidence score training data is created, the concept error rate of the parsed hypothesis is used to determine the label, rather than its word error rate. Unlike the approach presented in this chapter, only the top recognition hypothesis is considered.

In addition, there has been much work on extracting features to use in the confidence scoring process from the language understanding phase, an idea that has heavily influenced the work presented in this chapter. Each of [7, 14, 21, 87] demonstrates the utility of training a classifier with features derived from the natural language and dialogue management components of a spoken dialogue system to better predict the quality of speech recognition results.

The methodology described in [21] is especially relevant. In this work, each hypothesis on the N-best list is parsed, and multimodal reference resolution is performed. Features from each parsed and reference-resolved hypothesis are extracted, and then a memory-based classifier is used to classify each hypothesis in turn, starting at the top of the N-best list. Each hypothesis can be classified as either *in-grammar*, *out-of-grammar*, or *crosstalk* (speech not directed at the system). The top hypothesis classified as *in-grammar* is chosen as input to the system.

This methodology shares some similarities with the one presented in this chapter;

most strikingly, the idea of processing each hypothesis on the N-best list to form alternative interpretations. However, the response confidence scorer presented in this chapter differs fundamentally in that it processes each hypothesis to produce an actual system response, rather than a reference-resolved parse. By considering the response generated by the system, the problem of confidence scoring is recast in terms of the response accuracy of the system, rather than in terms of its input accuracy. It also allows for features drawn from the full contextual knowledge of the system (*e.g.*, does an input parse lead to any database results?), and for features based on the distribution of unique responses.

In addition, because of the small size of the corpus used in [21], the authors were limited to testing their approach with leave-one-out cross validation. As a result, testing on a particular user's utterance, other utterances from the same user also contributed to the training set. This means that the experiments do not measure how well the technique will work for an unfamiliar user. Their method also does not provide a way to set a rejection threshold that optimizes a particular metric—such as F-measure. Finally, another key difference is that the experiments described in the next chapter make use of an $n$-gram language model with a large vocabulary of proper names, whereas in [21], a context-free grammar with a much smaller vocabulary is used.

### 7.1.3 Stochastic Dialogue Management

Various methods have been devised for incorporating uncertainty into the decision of how to produce an appropriate response to a user of a conversational interface. Typically, this uncertainty derives primarily from speech recognition errors, so it is related to the work described in this chapter since in both cases the goal is to produce an appropriate system response based on errorful input. Typically, this work focuses on maintaining a probability distribution over the concepts a system has understood: for example, an airline reservation system may assess the probability that the user wants to depart from "Boston" at 90%, and from "Austin" at 10%. Based on the firmness of its beliefs, it might decide to either (1) explicitly ask the user if the departure city was, indeed, "Boston", (2) implicitly confirm that "Boston" is the departure city by mentioning it in the next prompt ("OK, from Boston. Where would you like to fly to?"), or (3) not mention it at all: ("Where would you like to fly to?"). A popular way of updating such probability distributions over beliefs as the conversational progresses is via the use of a Partially Observable Markov Decision Process (POMDP) [93].

Optimizing the manner in which the system responds based on these beliefs can be performed by evaluating the trade off between the costs – for example the length of the dialogue – and the payoff of actually obtaining the correct values for a particular belief. In [9], a practical, data-driven method of performing such optimization with logistic regression is described. Other work has focused on the use of reinforcement learning to learn dialogue strategies (*e.g.*, [71]). While promising, such learning techniques require a very large amount of training data, making it implausible to collect training data from real users, and requiring instead the use of "simulated" users [24].

The stochastic dialogue management strategies discussed in this section are very much related to the response confidence scores discussed here; however, they differ somewhat in their goals. These techniques are generally geared toward responding appropriately based on a belief as to whether certain key bits of information have been well understood, while this chapter is concerned more globally with the quality of the system response as a whole. The response feature extraction methodologies described here could potentially play a role in the belief updating phase of stochastic dialogue management algorithms, as could the response confidence scores themselves.

Finally, another approach to propagation of uncertainty in dialogue systems can be found in [65]. In this dialogue system architecture, uncertainty is propagated across each layer of processing through the use of probabilities, eventually leading to posterior probabilities being assigned to candidate utterance interpretations. Unlike the confidence algorithm described here, in which a single classifier is trained using arbitrary features derived from each stage of processing, each component (recognizer, parser, *etc.*) is trained separately and must be capable of assigning conditional probabilities to its output given its input. The method hinges on probabilistic inference, yet it is often problematic to map a speech recognizer's score to a probability as the approach requires. In addition, the method is evaluated only in a toy domain, using a few sample utterances – it is unclear how, or if, it would extend to a larger scale, useful conversational interfaces.

### 7.1.4 Functional Accuracy and Multimodal Disambiguation

As was noted at the beginning of this chapter, response accuracy – as opposed to the accuracy of the input speech hypothesis – plays a key role in the response confidence scoring approach discussed in this chapter. The accuracy of a system's response, as opposed to its input, has been referred to elsewhere as its *functional accuracy* [52, 19]. Functional accuracy has long been noted as a key measure for multimodal interfaces, since interfaces that rely on multiple input streams – *e.g.*, speech and gesture – may be able to perform *mutual disambiguation* by combining information present in those input streams [66]. Such disambiguation occurs when information that has been underspecified in one input stream, for example a spoken deictic expression, is specified in another stream – for example, a drawn circle. While both input streams may contain errors due to individual recognition processes, mutual disambiguation may increase the functional accuracy above what would be predicted by the individual accuracies of the input streams.

When applied to multimodal interfaces like *City Browser*, the response confidence scoring algorithm described in this chapter can be seen as a part of the multimodal disambiguation process, as it takes into account the integration of spoken input with graphical user interface actions like drawing and clicking. Response confidence scoring, then, provides a way to judge the quality of responses generated using complementary input streams, because it is meant to judge functional, rather than input, accuracy. This falls out of the general property that response confidence scoring, by examining the response rather than the input, takes into account all contextual information available. This property makes it particularly useful in multimodal interfaces,

though its usefulness ought to extend speech-only conversational interfaces as well.

## 7.2 Data Exploration

Figures 7-1 and 7-2 discussed above present a hypothetical example in which multiple recognition hypotheses on the N-best list yield a smaller set of unique system responses, if each recognition hypothesis is interpreted individually in the correct context. Moreover, they suggest that, in some cases at least, the system may respond correctly in spite of speech recognition errors. This section uses *City Browser* data to explore these two ideas.

### 7.2.1 Reduced Response Set

Figure 7-3 explores the intuition that multiple recognition hypotheses can yield a smaller set of unique system responses using *City Browser* data in the *Web* corpus discussed in Chapter 4. To produce the figure, N-best lists with up to fifty hypotheses were produced for each utterance in the corpus. Each hypothesis was parsed, and then interpreted in context to produce a system response. The number of hypotheses, unique parses, and unique responses were then plotted as a function of the maximum N-best list length. Generally, the figure shows that about half as many unique parses are generated as recognition hypotheses, and then half again as many unique responses. Since many hypotheses evoke the same response, there is little value in discriminating among these hypotheses by assigning a distinct confidence score to each. Instead, it appears that information may be gained about the quality of a response by pooling knowledge gleaned from each hypothesis evoking that response.

It seems reasonable to expect a similar trend in which multiple hypotheses map to a single parse in any conversational interface where parses contain a mixture of syntactic and semantic structure—as is the case here—or, even more so, where they contain only semantic information (*e.g.*, slot/value pairs [6, 90]). Parsers that retain more syntactic structure would likely generate more unique parses; however, many of these parses would probably map to the same system response, since a response does not typically hinge on every syntactic detail of an input utterance.

On the other hand, many parsers are capable of generating several parses for a given input utterance, due to syntactic and/or semantic ambiguities. For example, although the TINA parser can create multiple possible parses for a given input, to produce Figure 7-3 only the top ranking parse for each hypothesis was used. However, it would certainly be feasible to process multiple parses for each hypothesis. In this case, it could well be that there are, on average, a greater number of unique parses than recognition hypotheses. The confidence scoring technique developed in this chapter ought to still be applicable to the set of unique responses generated, perhaps with parse scores or rankings used as an input feature.

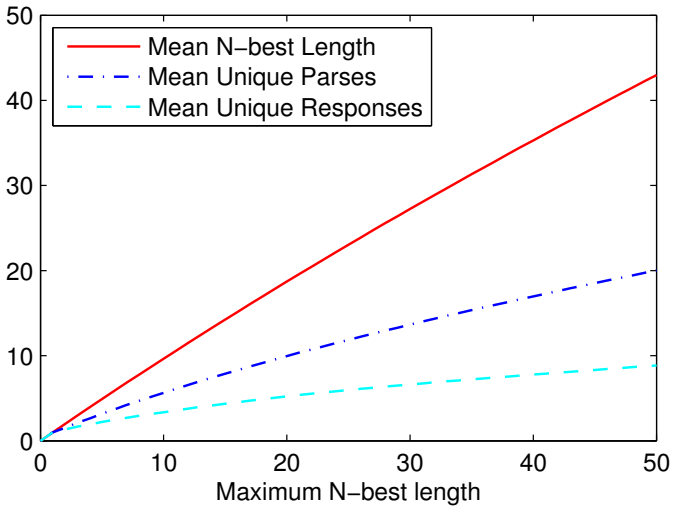



Figure 7-3: The mean N-best recognition hypothesis list length, mean number of unique parses derived from the N-best list of recognition hypotheses, and mean number of unique system responses derived from those parses, given a maximum recognition N-best list length.

### 7.2.2 Hypotheses with Errors

Figure 7-4 explores the intuition that speech recognition hypotheses with errors may still yield appropriate system responses. It shows a histogram of utterances from the *Car* corpus, distributed according to how many word errors – substitutions, insertions, or deletions – occurred in the top ranking hypothesis associated with the top scoring system response (as determined by the response scoring algorithm developed in the next section). Each bar in the histogram is divided into two categories, indicating whether the annotators labeled it as correct or incorrect.

The histogram shows that almost all utterances with no word errors are answered correctly, as would be expected. Those which aren't represent errors made by the natural language understanding or response generation components of the system. More importantly, the histogram indicates that the majority of utterances with a single word error can still lead to an appropriate system response, as well as a significant portion of those with two or three errors. As such, it demonstrates well that errors in the input to the natural language processing pipeline do not necessarily lead to errors in the generated system response.

## 7.3 Response Confidence Scoring Algorithm

With an intuition developed, and some data supporting this intuition in hand, this section describes in detail the proposed response confidence scoring process. Given an input utterance, the final goal of the algorithm is to produce a list of candidate system responses, each associated with a score. The algorithm operates as follows:

1. Given a spoken utterance, use the speech recognizer to produce an N-best list

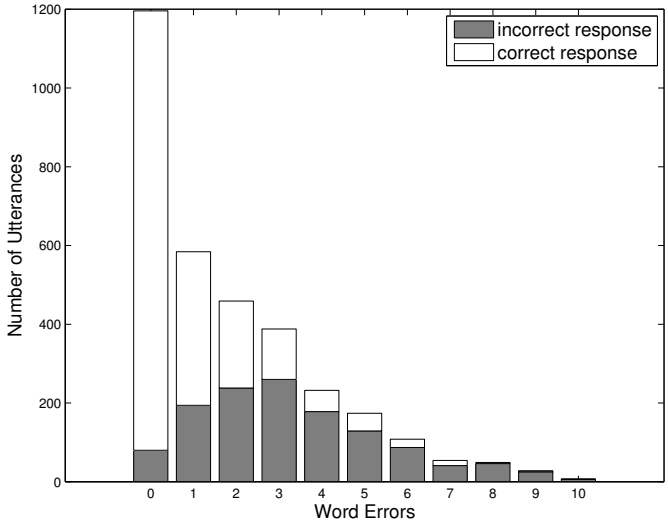



Figure 7-4: Histogram of utterances from the *City Browser Car* corpus for which the top-scoring system response was annotated as correct or incorrect, as a function of the number of word errors made by the speech recognizer in the hypothesis associated with that top scoring response.

of hypotheses $H_1, H_2, \ldots, H_n$, with associated feature vectors $\vec{F}_{H_1}, \vec{F}_{H_2}, \ldots, \vec{F}_{H_n}$ where features include those used in traditional confidence scoring modules, such as the acoustic and language model scores for each hypothesis.

2. Use the dialogue system to interpret each hypothesis $H_1, H_2, \ldots, H_n$ in context and produce responses $R_1, R_2, \ldots, R_n$. Calculate feature vectors, $\vec{F}_{R_1}, \vec{F}_{R_2}, \ldots, \vec{F}_{R_n}$ associated with each response, and the process of creating each response – for example, features might include the type of response or the number of database items displayed on the graphical user interface.

3. Determine the set of *unique* responses, $R'_1, R'_2, \ldots, R'_m$ where $m \leq n$.

4. Merge feature vectors created in the previous two steps so that there is only one vector associated with each unique response – *e.g.*, by choosing the best value appearing for each feature – to produce $\vec{F}'_{H_1}, \vec{F}'_{H_2}, \ldots, \vec{F}'_{H_n}$ and $\vec{F}'_{R_1}, \vec{F}'_{R_2}, \ldots, \vec{F}'_{R_n}$

5. Create distributional features for each unique response that measure, for example, how frequently each unique response was produced: $\vec{F}_{D_1}, \vec{F}_{D_2}, \ldots, \vec{F}_{D_m}$.

6. Use a Support Vector Machine (SVM) to produce a score $S_i$ for each unique response $R_i$, using as input the features $\{\vec{F}'_{H_i}, \vec{F}'_{R_i}, \vec{F}_{D_i}\}$.

7. If the highest scoring response exceeds the rejection threshold, use it. Otherwise, indicate to the user that clarification is necessary.

Figure 7-5 illustrates graphically the above algorithm, which is applicable to a wide range of spoken or multimodal interfaces. It ought to be useful for any system that

**Recognition N-best**

| Hypothesis | Rank | $S_t$ | $S_a$ | $S_l$ | … | *DS* | Response | Parse | … |
|---|---|---|---|---|---|---|---|---|---|
| thirty two vassal street in cambridge | 0 | 45.3 | 28.5 | 26.5 | | → | $R_0$ | *FULL* | |
| thirty two vassar street in cambridge | 1 | 45.0 | 27.1 | 30.5 | | → | $R_1$ | *FULL* | |
| thirty two vassar street in in cambridge | 2 | 44.2 | 26.0 | 30.4 | | → | $R_1$ | *ROBUST* | |
| at thirty two vassar street <noise> | 3 | 40.1 | 26.5 | 29.4 | | → | $R_1$ | *FULL* | |
| at thirty two vassal street in cambridge | 4 | 39.5 | 26.3 | 29.0 | | → | $R_1$ | *FULL* | |
| thirty two vassar street cambridge <noise> | 5 | 38.4 | 25.8 | 28.4 | | → | $R_1$ | *FULL* | |
| thirty two vassar street in canton | 6 | 38.0 | 25.8 | 28.3 | | → | $R_2$ | *FULL* | |
| thirty two vassal street in in canton | 7 | 33.5 | 22.5 | 27.5 | | → | $R_3$ | *ROBUST* | |
| twenty vassar in street in zoom | 8 | 32.4 | 22.3 | 26.3 | | → | $R_4$ | *NONE* | |
| thirty two vassar street in cambridge <noise> | 9 | 32.0 | 19.5 | 26.7 | | → | $R_1$ | *FULL* | |

**Response List**

| Response | Rank | $S_t$ | $S_a$ | $S_l$ | *%Top3* | *%Top5* | *Dist.* | *Parse* | … | *SVM* | Score | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_0$ | 0 | 45.3 | 28.5 | 26.5 | .33 | .8 | 5 | *FULL* | | → | .42 | |
| $R_1$ | 1 | 45.0 | 27.1 | 30.5 | .66 | .2 | 5 | *FULL* | | → | .73 | → $R_1$ |
| $R_2$ | 6 | 38.0 | 25.8 | 28.3 | 0.0 | 0.0 | 5 | *FULL* | | → | .32 | |
| $R_3$ | 7 | 33.5 | 22.5 | 27.5 | 0.0 | 0.0 | 5 | *ROBUST* | | → | .20 | |
| $R_4$ | 8 | 32.4 | 22.3 | 36.3 | 0.0 | 0.0 | 5 | *NONE* | | → | .02 | |

Figure 7-5: The feature extraction and classification process. The top half of the diagram shows how an N-best list of recognizer hypotheses, with associated scores from the recognizer, are processed by the dialogue system (*DS*) to produce a list of responses. Associated with each response is a set of feature values derived from the response itself, as well as the process of evoking the response (*e.g.*, the parse status). The bottom half of the figure shows how the unique responses are collapsed into a list. Each response in the list inherits the best recognition scores available from hypotheses evoking that response; each also has feature values associated with it derived from the distribution of that response on the recognizer N-best list. Each set of feature values is classified by a Support Vector Machine, and the resulting score is used to rank the responses. If the highest scoring response exceeds the rejection threshold, then it is chosen as the system's response.

tries to understand a spoken utterance and produce a response of some sort. In order to use it in a particular domain, the following must be provided by the system designer:

1. A specific set of features to use at each stage of feature extraction – recognition, response generation, and distribution – and a means of calculating values for those features. Recognition and distribution features are meant to be domain general, as they are not dependent on the specifics of the response generation process, while response generation features capture domain-specific properties of the response itself.

2. A feature merging process to be used in step 4.

3. Training data for the SVM used for scoring in step 6.

4. A rejection threshold, used in step 7, that ought to be tuned in a data-driven manner as well.

## 7.4 Training: Data and Annotation

Training data is required for the SVM used to produce scores, and to tune the rejection threshold. The training data should be labeled instances that map feature values associated with each unique response to a label of *acceptable* or *unacceptable*. Ideally, this data should be mined from logs of real users interacting with the conversational interface in question. The logs should either include the candidate responses generated for each hypothesis on the N-best list, or it should be possible to create these responses in an offline process.

For a given utterance, each unique candidate system response must then be labeled as either *acceptable* or *unacceptable*. The most accurate way to do this is to annotate each response by hand: human annotators who are aware of the current context of the conversation should label each response. Alternatively, if transcripts for each utterance are available, it may be feasible to obtain reasonable labels using these. Specifically, each transcript may be processed by the dialogue system to produce a response, which is then compared to the list of responses produced by the recognition hypotheses; matching hypotheses are marked as *acceptable*. This approach is not perfect, however, because the dialogue system may not produce the correct response based on the transcript, or several unique responses may each be acceptable. Nonetheless, this approach is useful if a corpus of transcribed data already exists, as it requires no additional manual annotation.

An interesting alternative to manual annotation would be to allow users to label data themselves. For example, all, or some, of the candidate system responses could be presented to the user after he or she speaks. The user would then choose the correct one, or – if none were correct – try speaking again. This method could be used after every user utterance, or just after ones where, for instance, the top ranking response has a score near the rejection threshold. In this way, the system could gather training

data for the cases it is most unsure about, a form of *active learning*. Moreover, showing alternative responses could be a useful clarification strategy, allowing users to pick the correct alternative rather than having to repeat or rephrase the previous utterance.

Once a labeled set of training data is available, it can be used to train the SVM. A logistic regression model fit on the training data is used so that the SVM yields a score between 0 and 1, where higher values indicate a greater likelihood that a particular instance is *acceptable*.

The rejection threshold can then be tuned using either a held-out set, or the training data itself, using an objective function based on the desired characteristics of the dialogue system. In a mission-critical application, for example, it may be preferable to accept only high-confidence responses, and to clarify otherwise. It might also be possible to optimize the threshold in a more sophisticated manner, such as that developed in [9] where task success is used to derive the cost of misunderstandings and false rejections, which, in turn, are used to set a rejection threshold.

While a thresholding approach is straightforward to implement and tune, other approaches are feasible as well. For instance, a second classifier could be used to decide whether or not to accept the top ranking response. The classifier could take into account such features as the spread in scores among the responses, the number classified as *acceptable*, the drop between the top score and the second-ranked score, *etc.*

## 7.5 Implementation Considerations

The response confidence scoring method described in this chapter is potentially quite computationally demanding. Natural language parsing, dialogue management, and generation may require significant computation in their own right. So, processing even a single speech recognition hypothesis may well be a time-consuming process; processing 5, 10, 100, or 1000 hypotheses, then, requires proportionally more computation. Users will not be interested in interacting with a system that has a high response latency, which means that the potential improvement in response accuracy yielded by the confidence scoring technique is not worthwhile if the increase in latency is too large.

Latency, however, need not increase linearly with the number of hypotheses in the N-best list. Since each hypothesis is considered in isolation, each response may be generated in parallel. This means that, in theory, processing can be distributed across an arbitrary number of servers, which generate and score each response. Average response latency, in this case, may still increase somewhat, as it will now be bounded by the longest processing time required for any one hypothesis. Assuming there is not great variation in the time required to produce a response, this effect ought to be small.

This analysis assumes that the computation involved in feature extraction and classification are negligible. Recognition features such as acoustic and language model scores must be calculated in the recognition process anyway, so there is essentially no

cost in including these features. Distributional features of the responses simply involve counting, and can be calculated quite quickly. The response generation features, however, are at the discretion of the system designer; in the experiments in the following chapter, each feature used is simply a by-product of the response-generation process, and requires no additional computation. Finally, classification with an SVM is fast, as it simply involves summing a set of weighted feature values.

In situations when computation is performed on a server, as when conversational interfaces are deployed via the telephone or the web, latency need not significantly increase, as parallelization is simply a matter of hardware costs. If computation is to occur on local hardware – the user's own computer or mobile device, for instance – then the computational overhead of the approach outlined here may become more of a consideration (though, as multicore CPUs become increasingly common, parallelization on consumer hardware will become more common). In the chapter that follows, performance implications of the confidence scoring algorithm are investigated on a laptop computer running the *City Browser* system in the trunk of the car used to collect the *Car* and *Car-Pilot* corpora detailed in Chapter 4. While response latency does increase, it remains low enough that the interface remains quite usable.

## 7.6 Conclusion

This chapter has explored the motivation behind response confidence scoring, which recasts the traditional problem of assigning confidence scores to speech recognition hypotheses to one where system responses are scored instead. A specific algorithm for producing such scores was detailed, in which an N-best list of speech recognition hypotheses is used to create a unique set of candidate system responses for a particular utterance. Each response is assigned a score by a Support Vector Machine, which is trained using labeled data with features culled from the speech recognition and response generation processing stages, as well as drawn from the distributional properties of the responses themselves. The acceptability of responses used for training data may be labeled by hand, by comparing each response to that generated by the transcript for a particular utterance, or by users themselves as part of their interaction.

The confidence scoring process requires linearly more computation in the length of the N-best list. However, each response may be generated independently, so the computation is straightforward to parallelize. This means that for server-based conversational interfaces deployed over the telephone or the web, increases in response latency may potentially be negligible, so long as multiple servers or CPUs are available.

# Chapter 8

# Response Confidence Scoring Algorithm Evaluation

In the previous chapter, a response confidence scoring algorithm was developed, which allows for contextual information to be integrated into a system's ability to judge the quality of its response, and hence the likelihood that it understood the user correctly. In this chapter, the usefulness of confidence scores generated via this approach is compared to that of traditional recognition confidence scores.

Two sets of experiments are reported, both of which make use of corpora discussed in chapter 4 in which user interactions with *City Browser* were recorded. In each experiment, the Weka toolkit [94], version 3.4.12, is used to build the Support Vector Machine. The first set of experiments were performed offline, using the *Web* corpus. These experiments validate the intuition that response confidence scoring can lead to a significant improvement in system response accuracy. In the second set of experiments, a response confidence scoring model was trained using the *Car-Pilot* corpus, and then actually deployed during the collection of the *Car* corpus. These experiments demonstrate both that the response confidence scoring algorithm proposed in the previous chapter is feasible to deploy in a live system, and that it yields a significant improvement in response accuracy.

## 8.1 Baseline

The experiments in this section make use of a state-of-the-art utterance-level confidence model [43] as the baseline. The module uses a linear projection model to produce a confidence score based on 15 features derived from recognizer scores, and from comparing hypotheses on the N-best list. In each evaluation, the module was trained and tested on the same data as the SVM model. In addition, its rejection threshold was optimized following the same procedure used for the response confidence scoring algorithm in each experiment. Whenever the utterance confidence score exceeded the rejection threshold, the response generated by the top hypothesis on the N-best list was chosen.

| Recognition | | | Distributional | Response |
|---|---|---|---|---|
| **(a) Best across hyps:** | **(b) Drop:** | **(c) Other:** | percent_top_3 | response_type |
| total_score_per_word | total_drop | mean_words | percent_top_5 | num_found |
| acoustic_score_per_bound | acoustic_drop | top_rank | percent_top_10 | POI_type |
| lexical_score_per_word | lexical_drop | n-best_length | percent_nbest | is_subset |
| | | | top_response_type | parse_status |
| | | | response_rank | geographical_filter |
| | | | num_distinct | |

Table 8.1: Features used to train the acceptability classifier. Nine features are derived from the recognizer; seven have to do with the distribution of responses; and six come from the process of generating the candidate response.

## 8.2 Feature Sets

The recognition, response generation, and distributional feature sets that were used to train the SVM classifier in the experiments reported in this chapter are shown in Table 8.1, with all possible values for non-numerical features listed in Table 8.2. The recognition and response features associated with each unique response take their values from the best values of each associated with a recognition hypothesis that led to that response.

As Table 8.1a indicates, the recognition features include the acoustic, language model, and total scores calculated during the speech recognition process. In addition, the drop in score between the response's score for each recognition feature and the top value occurring in the N-best list is used as a feature (see Table 8.1b). Finally, the rank of the highest hypothesis on the N-best list that evoked the response, the mean number of words per hypothesis evoking the responses, and the length of the recognizer's N-best list are used as features (see Table 8.1c). Each of these features is domain independent, and could be used with any conversational interface.

The distributional features, which are also domain independent, are calculated by examining the distribution of hypotheses on the N-best list that evoke the same unique response. The frequency with which a particular response is evoked by the top 3, top 5, top 10, and by all hypotheses on the N-best list are used as features. Features are generated, as well, based on the distribution of unique responses. These features are: the initial ranking of this response on the list, the number of distinct responses on the list, and the type of response that was evoked by the top hypothesis on the recognizer N-best list.

Finally, domain-dependent features derived from the response itself, and from natural language processing performed to derive that response, are also calculated. The high-level type of the response, as well as the type and number of any points of interest returned by a database query are used as features if they exist, as is a boolean indicator as to whether or not these results are a subset of the results currently shown on the display. If any sort of "geographical filter", such as an address or circled region, is used to constrain the search, then the type of this filter is also used as a feature. Finally, the "best" parse status of any hypotheses leading to this response is also used, where $full\_parse \succ robust\_parse \succ no\_parse$.

| Feature | Possible Values |
|---|---|
| response_type<br>top_response_type | geography, give_directions, goodbye, greetings, help_directions_did_not_understand_from_place, help_directions_did_not_understand_to_place, help_directions_no_to_or_from_place, help_directions_subway, hide_subway_map, history_cleared, list_cuisine, list_name, list_street, no_circled_data, no_data, no_match_near, non_unique_near, ok, panning_down, panning_east, panning_south, panning_up, panning_west, presupp_failure, provide_city_for_address, refined_result, reject_or_give_help, show_address, show_subway_map, speak_properties, speak_property, speak_verify_false, speak_verify_true, welcome_gui, zooming, zooming_in, zooming_out |
| POI_type | none, city, museum, neighborhood, restaurant, subway_station |
| parse_status | no_parse, robust_parse, full_parse |
| geographical_filter | none, address, circle, line, list_item, map_bounds, museum, neighborhood, point, polygon, restaurant, subway_station, city |

Table 8.2: The set of possible values for non-numerical features, which are converted to sets of binary features.

## 8.3 Evaluation Metrics

Since the goal of the response confidence module is to increase the accuracy of the system response, an appropriate metric must be used to measure this accuracy. For each utterance, the response confidence module will either choose a particular response, or choose to *reject*, if no response has a high enough score. In the ideal case, if at least one *acceptable* candidate response exists, then an *acceptable* response should be chosen and presented to the user. If no *acceptable* candidate response has been generated, then the system should choose to *reject*. Sometimes, however, the system will make an error: it will choose an *unacceptable* response; or it will choose to *reject*, even though an *acceptable* response appears on the list. All of this is further complicated by the fact that some system responses may, in fact, themselves be *reject*, since sometimes the system may not be able to make sense of the input hypothesis.

In a standard binary decision problem, F-measure, the harmonic mean of precision and recall, can be used to judge the accuracy of the classifier. It is calculated as follows:

$$F = \frac{2(precision * recall)}{precision + recall} \tag{8.1}$$

Precision and recall, in turn, are calculated based on the number of true positives ($TP$), false positives ($FP$), true negatives ($TN$), and false negatives ($FN$).

$$precision = \frac{TP}{TP + FP} \tag{8.2}$$

$$recall = \frac{TP}{TP + FN} \tag{8.3}$$

Though the problem at hand is not quite a binary decision problem, it can be mapped fairly well onto one, so that F-measure may be used as a measure of accuracy. In particular, Figure 8-1 summarizes each possible situation, and shows whether the system action in each case should be labeled as a true positive, false positive, true negative, or false negative. Given this matrix, an F-measure can be calculated, and used to compare the accuracy of each system.

In addition, by varying the rejection threshold, a receiver operating characteristic (ROC) curve can be plotted for each system, which shows the trade off between *false positive rate* and *true positive rate*, defined as follows:

$$TP_{rate} = \frac{TP}{TP + FN} \tag{8.4}$$

$$FP_{rate} = \frac{FP}{FP + TN} \tag{8.5}$$

Unlike in a typical binary decision problem, however, it may be the case then when the rejection threshold is set to the minimum value, the true positive rate does not reach 100%. The reason for this is that, while the system will indeed choose a response from the list, it may be the case that it chooses an *unacceptable* one, even though an *acceptable* one appears elsewhere on the list. Or, the top ranking response may actually be *reject*.

### 8.3.1 Statistical Significance

McNemar's test [63] is used in this chapter to calculate the probability that predictions made by two classifiers on the same set of data were due to chance. Given two classifiers $C_1$ and $C_2$, McNemar's test is computed using a 2x2 contingency table containing counts for the number of times:

- $a$ = number of instances for which $C_1$ and $C_2$ were both correct,
- $b$ = number of instances for which $C_1$ was correct, $C_2$ was incorrect,
- $c$ = number of instances for which $C_1$ was incorrect, $C_2$ was correct,
- $d$ = number of instances for which $C_1$ and $C_2$ were both incorrect.

The statistic used in McNemar's test is $\chi^2$, calculated as follows:

$$\chi^2 = \frac{(b - c)^2}{b + c} \tag{8.6}$$

## 8.4 Offline Experiments with the *Web* Corpus

The first set of experiments were performed using the *City Browser* corpus collected in the *Web* condition. In this corpus, users interacted with the *City Browser* system via the web to complete 11 scenario-based tasks. Each user utterance was transcribed,

**Case I**

$R_0$ is *acceptable* and is not *reject*

$S_0 \geq T \rightarrow$ T.P.
$S_0 < T \rightarrow$ F.N.

| Response | Score | Type | Label |
|---|---|---|---|
| $R_0$ | $S_0$ | speak_property | acceptable |
| $R_1$ | $S_1$ | list_cuisine | unacceptable |
| $R_2$ | $S_2$ | speak_property | unacceptable |

**Case I: Example Ranked Response List**

**Case II**

No candidate responses *acceptable*, or *acceptable* response is *reject*

| **(a)** | **(b)** |
|---|---|
| $R_0$ is not *reject* | $R_0$ is *reject* |
| $S_0 \geq T \rightarrow$ F.P. | $S_0 \geq T \rightarrow$ T.N. |
| $S_0 < T \rightarrow$ T.N. | $S_0 < T \rightarrow$ T.N. |

| Response | Score | Type | Label |
|---|---|---|---|
| $R_0$ | $S_0$ | speak_property | unacceptable |
| $R_1$ | $S_1$ | list_cuisine | unacceptable |
| $R_2$ | $S_2$ | speak_property | unacceptable |
| $R_3$ | $S_3$ | reject | unacceptable |
| $R_4$ | $S_4$ | zooming_out | unacceptable |

**Case II: Example Ranked Response List**

**Case III**

$R_n$ (with $n > 0$) is *acceptable* and is not *reject*

| **(a)** | **(b)** |
|---|---|
| $R_0$ is not *reject* | $R_0$ is *reject* |
| $S_0 \geq T \rightarrow$ F.P. | $S_0 \geq T \rightarrow$ F.N. |
| $S_0 < T \rightarrow$ F.N. | $S_0 < T \rightarrow$ F.N. |

| Response | Score | Type | Label |
|---|---|---|---|
| $R_0$ | $S_0$ | speak_property | unacceptable |
| $R_1$ | $S_1$ | list_cuisine | acceptable |
| $R_2$ | $S_2$ | speak_property | unacceptable |
| $R_3$ | $S_3$ | reject | unacceptable |
| $R_4$ | $S_4$ | zooming_out | unacceptable |

**Case III: Example Ranked Response List**

Figure 8-1: Algorithm for calculating the F-measure confusion matrix of True Positives (T.P.), False Positives (F.P.), True Negatives (T.N.), and False Negatives (F.N.). The ranking technique described in this chapter creates a list of candidate system responses ranked by their scores. The top scoring response is then *accepted* if its score exceeds a threshold $T$, otherwise all candidate responses are *rejected*. As such, the problem is not a standard binary decision. All possible outcomes from the ranking process are shown with an indication of whether each is counted as a T.P., F.P., T.N., or F.N. Note that given this algorithm for calculating the confusion matrix, no matter where the threshold $T$ is set, F-measure will always be penalized if Case III occurs.

and system logs associate the appropriate contextual information with each utterance. In order to use the data for evaluation, it was necessary to:

1. Produce an N-best list associated with each utterance,
2. Use the N-best list to generate a list of candidate system responses to associate with each logged utterance,
3. Extract feature values for each candidate response,
4. Label each response as *acceptable* or *unacceptable*.

In an offline process, 10-best lists were created for each utterance, using system logs to determine the appropriate language model for the utterance – as the language model can dynamically change depending on the context of the conversation. Each hypothesis was then passed through the dialogue system, along with associated context and gesture information from the system logs, to produce a set of unique responses and associated feature values. 1,912 of the 2,138 total utterances in the corpus were used; utterances from the warmup task were excluded, as were several for which the dialogue state had not been properly logged.

In order to label each response as *acceptable* or *unacceptable*, the transcript associated with each utterance was processed by the dialogue system to produce a response, which was used as the gold standard. If this response was generated from any of the N-best hypotheses, it was marked as *acceptable*; all other generated responses were marked as *unacceptable*.

With the labeled data, 38-fold cross validation was performed, where in each case the held-out test set was comprised of all the utterances from a single user. This ensured an accurate prediction of a novel user's experience, although it meant that the test sets were not of equal size. In each case, the rejection threshold was tuned by optimizing the F-measure on the training set.

### 8.4.1 Results

Table 8.3 compares the baseline recognizer confidence module to the response confidence scoring in terms of F-measure. The response confidence scoring method was evaluated using several subsets of the features listed in Table 8.1. Using features derived from the recognizer only, it is possible to obtain results comparable to the baseline. Adding the response and distributional features yields a 16% improvement over the baseline system, which is statistically significant according to McNemar's test, with $p < .01$. Generally, the response features are more useful than the distributional features, both on their own and in conjunction with the recognition features.

Interestingly, the response and distributional features perform quite well without any recognition features – in fact, nearly as well as when recognition features are included. This is quite an interesting result, as it shows that scores from the speech recognition process are not necessarily needed to produce accurate response confidence scores. This is an important consideration, since not all speech recognizers make such internal scores available. Distributional features capture some of the same information

| Features | F-measure |
|---|---|
| *Baseline* (Recognition Confidence Scores) | .619 |
| Distributional Features Only | .556 |
| Response Features Only | .567 |
| Recognition Features Only | .621 |
| Recognition + Distributional | .675 |
| Recognition + Response | .707 |
| Response + Distributional | .720 |
| Recognition + Response + Distributional | .723 |

Table 8.3: Average F-measures obtained via per-user cross-validation of the response-based confidence scoring method using the feature sets described in Section 8.2, as compared to a baseline system that chooses the top hypothesis if the recognizer confidence score exceeds an optimized rejection threshold. The classification results from each classifier are statistically significant compared to the baseline ($p < .01$) according to McNemar's test.

as acoustic and lexical scores: a high variation in the N-best list indicates uncertainty. Indeed, the baseline speech recognition confidence module uses an N-best "purity" feature as one of its features (see [43]).

Figure 8-2 plots ROC curves comparing the performance of the baseline model to the best response-based model. The curves were obtained by varying the value of the rejection threshold used for each test set, and then averaging the resulting individual ROC curves. The curves confirm that no matter the rejection threshold, the response confidence scoring module outperforms the baseline when using at least two sets of features in conjunction.

The above results were obtained by using an SVM with a linear kernel, where feature values were normalized to be on the unit interval. A quadratic kernel was also tried, as was retaining the raw feature values, and reducing the number of binary features by manually binning the non-numeric feature values. Each change resulted in a slight decrease in F-measure.

## 8.5 Online Experiments

While the results from the offline experiments in the previous section are promising, they can be improved upon in two important ways. First, they were performed offline on data that was previously collected, meaning that the feasibility of the response confidence scoring approach remains to be shown in an online system. Second, transcripts were used to label responses as *acceptable* or *unacceptable*, rather than high-quality human annotation. This section describes experiments in which the response confidence scoring module was deployed as part of an actual multimodal interface with which users interacted, and was trained on data annotated by hand.

In particular, the confidence scoring module was integrated into the *City Browser*

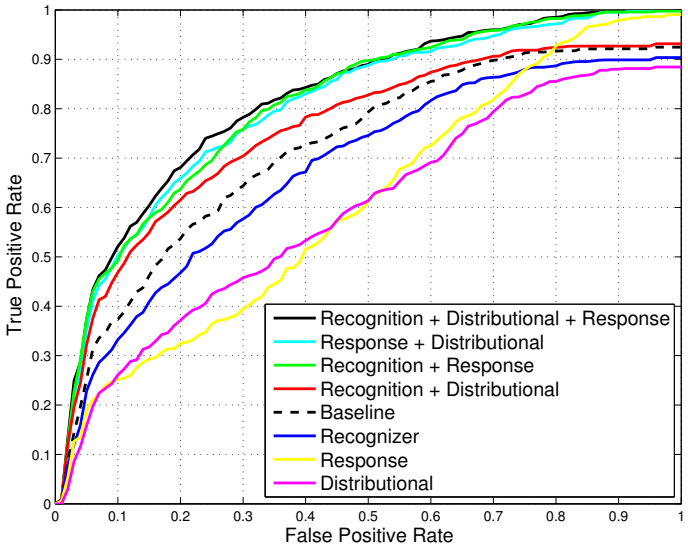



Figure 8-2: Receiver Operator Characteristic (ROC) curves (averaged across each cross-validation fold) comparing the baseline to response confidence models using combinations of each feature set. Note that due to the way that F-measure is calculated, some systems never reach a true positive rate of 100% since, in some cases, they always choose an unacceptable response even though an acceptable response appears elsewhere on the list; see Section 8.3 for details.

system, which was then deployed to a BMW 530xi sedan and used to collect the *Car-Pilot* and *Car* corpora described in chapter 4. The *Car-Pilot* phase served as an initial feasibility test that the response confidence scoring algorithm could be made to operate quickly enough to be useful in a live system, and gave system designers an opportunity to detect and fix bugs in the module. During this phase, the classifier was trained, and the rejection threshold was tuned, using data from the *Web* corpus initially; it was then retrained several times as new data was collected.

Once the *Car-Pilot* phase was complete, the SVM was trained using 868 utterances from 19 subjects in the *Car-Pilot* phase, and the rejection threshold was tuned with 422 utterances from an additional 9 subjects. Figure 8-3 shows the ROC curves produced on the tuning set, with the chosen rejection thresholds marked. For each system, a rejection threshold was chosen that yielded a 90% true positive rate: meaning that 90% of the time if an acceptable response appeared in the list of candidate responses, the system would choose it. At this operating point, the false positive rate for the response confidence module was 47%; it was 60% for the recognition confidence scoring baseline. The trained model and tuned rejection threshold were deployed in the live system, and used during the collection of the *Car* corpus.

### 8.5.1 Results

Figure 8-4 shows the ROC curves produced by the response confidence scores and the baseline recognition confidence module produced by the live deployment in the *Car*

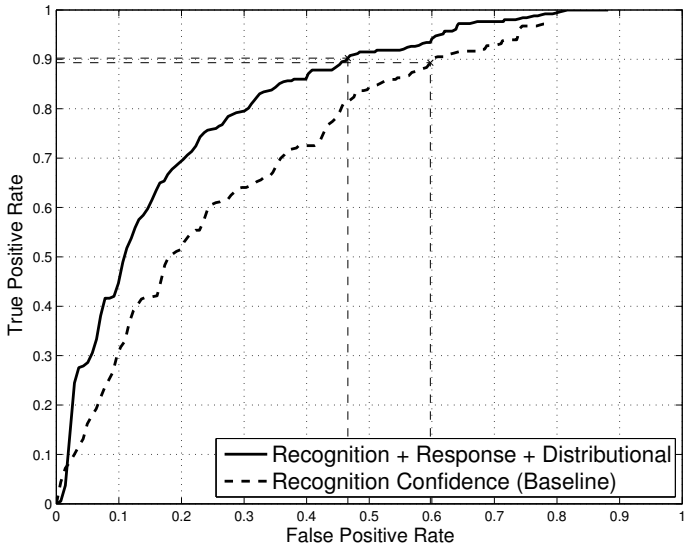



Figure 8-3: Receiver Operator Characteristic (ROC) curves on the tuning set. The marked points occur at a 90% true positive rate. The threshold corresponding to this operating point was used in the response confidence scoring module during the collection of the *Car* corpus.

corpus. Again, across all false positive rates, the response scoring module outperforms the baseline. The marked points on each curve show the operating points at which each system performed, given that the threshold was set to achieve a 90% true positive rate on the held-out tuning set. The response scoring module operated very nearly at this true positive rate, while the baseline system achieved a rate of about 74% – indicating that the tuned threshold led to a more consistent operating point in the case of the response confidence scoring module. At these operating points, the response confidence scoring system was a statistically significant improvement over the baseline according to McNemar's test, with $p << .01$. Moreover, to achieve a 90% true positive rate, the baseline confidence scoring module would have to incur about a 62% false positive rate, compared to about 40% for the response confidence scoring module.

It is also interesting to look at the distribution of scores assigned to the top-ranked response by the confidence module. Ideally, it should be a bimodal distribution, with responses annotated as being correct clustered together with high scores, and responses marked as incorrect clustered together with low scores. Figure 8-5 plots this distribution, showing two score histograms: one for correct response, and one for incorrect ones. While the scores do generally follow this pattern, it is interesting to note that there is a large range of scores (ranging from approximately .3 to .6) for which it is about equally likely that the response could be correct or incorrect. This suggests that a useful strategy for the conversational interface in these cases might be to *clarify* or *explicitly confirm* in some way, perhaps by asking the user to confirm the system's interpretation.

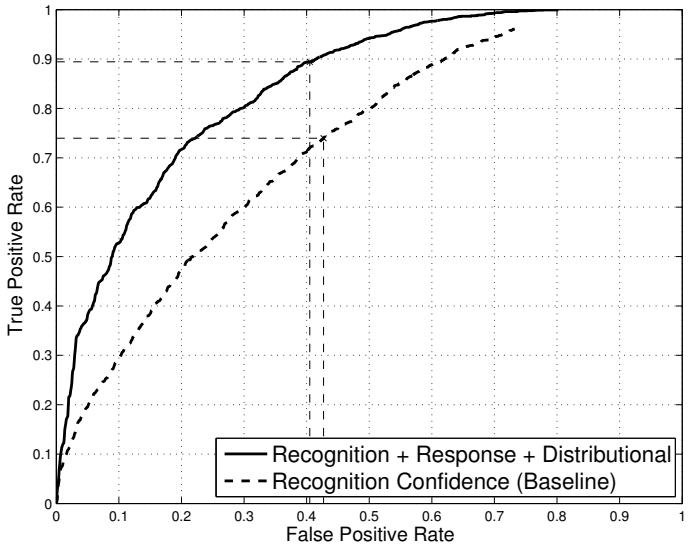



Figure 8-4: Receiver Operator Characteristic (ROC) curves for the response confidence module and the utterance confidence baseline module. The marked points shows the operating points achieved on the test set using the thresholds derived from the tuning set to achieve 90% true positive rates. In the case of the response confidence module, the operating point shown is the actual one experienced by users of the system. In the case of the baseline, it is the point users would have experienced, had they been using the baseline system. The two operating points represent classification results that are statistically significant according to McNemar's test, with $p << .01$.

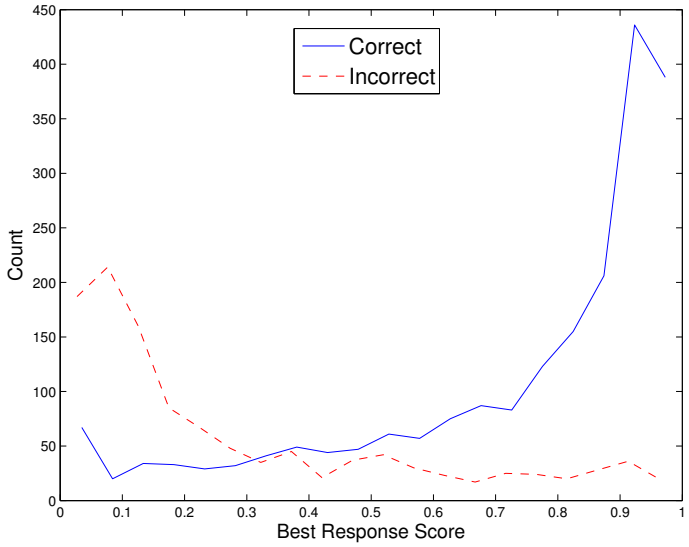



Figure 8-5: Histograms of scores for the top-ranked response for responses annotated as correct or incorrect.

### 8.5.2 Latency Considerations

As was discussed in Section 7.5 of the previous chapter, the response confidence scoring module is computationally expensive, when compared to an approach in which only the top hypothesis on the speech recognizer's N-best list must be processed by the dialogue system. Though, as was discussed above, computation can be parallelized in a straightforward manner, parallelization is not always possible. In the experiments discussed in this chapter, for instance, the dialogue system ran on a single laptop in the trunk of a car. As such, response latency was a real consideration.

Figure 8-6a shows a histogram of response latencies in the *Car* corpus, where response latency is defined as the elapsed time between when an N-best list of recognition hypotheses is available, and when the system actually responds to the user. Latency due to speech recognition – the elapsed time between when a user finishes speaking and the N-best list is available – is not included here, simply because it was not logged; however, this latency is very small in *City Browser*, as speech recognition occurs in near real time.

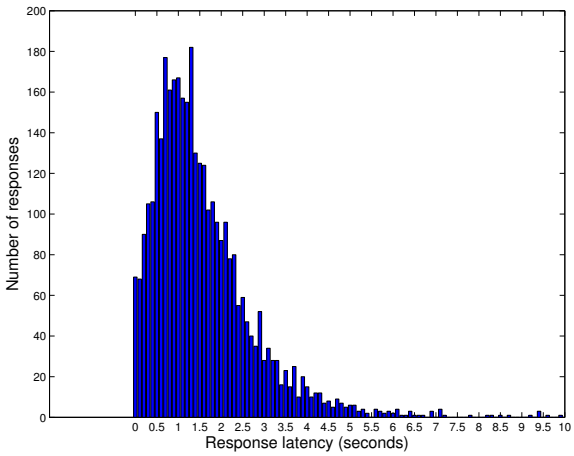
The median response latency was 1.3 seconds. The long tail involving long latency is presumably because in some cases, the system must make a request to an external web service to geocode addresses. Since it is connected to the Internet via a mobile broadband card, web access can sometimes be quite slow.

An important pre-processing optimization involved "cleaning" the N-best hypotheses before processing each one. In particular, hypotheses often include special tokens such as <*noise*> or <*pause*>. Since these are ignored by the natural language understanding components, two hypotheses that differ only by a <*noise*> tag will always evoke the same system response. As such, the response associated with each of these hypotheses can be calculated by only processing one of them. Depending on the N-best list, this can greatly decrease the amount of required computation. Moreover, it provides an opportunity to measure the relationship between the number of hypotheses that must be processed, and the time required to process them. Figure 8-6b plots the response latency as a function of the number of unique hypotheses on the N-best list, after "cleaning". Since computation could not be parallelized on this hardware, the response latency grows linearly with the number of unique hypotheses.

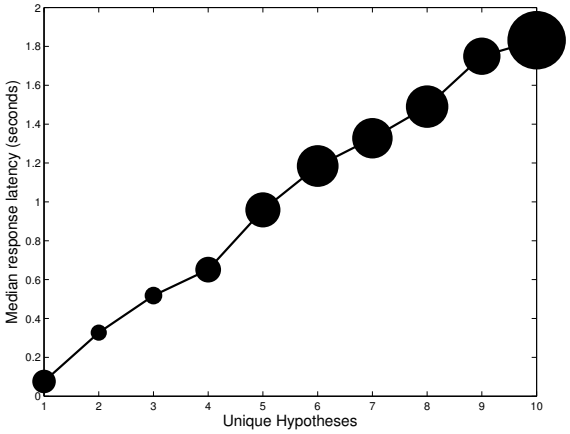
Taken together these results make clear the trade off between response accuracy and response latency for *City Browser*, when it is not run in a client-server environment. There is clearly a significant increase in response time when more recognition hypotheses must be considered. As such, experiments investigating the utility of various size N-best lists would make interesting and useful future work. On the other hand, a response latency of 1 to 2 second should be within the bounds of usability.

## 8.6 Conclusion

In this chapter, the response confidence scoring method described in the previous chapter was compared to a baseline system, in which only recognition confidence scores were considered. In both offline and online experiments with the *City Browser*




(a) Histogram of response latencies. The mean is 1.6 seconds, the median is 1.3 seconds.




(b) Response latency as a function of the number of unique "clean" recognition hypotheses. The radius of the circle marker at each point is proportional to the number of utterances that yielded that number of unique clean hypotheses.

Figure 8-6: Response latency in the *Car* corpus.

interface, the response confidence scorer was shown to significantly increase system response accuracy. Moreover, it was shown that various combinations of features drawn from speech recognition, response generation, and response distribution analysis yielded classifiers nearly as accurate as one making use of the entire feature set. Of particular interest was the fact that even without features from the speech recognition phase, the system performed nearly as well as when those features were available. This means that the response confidence scorer need not be tied to a particular speech recognition engine. Finally, response latency was analyzed in the *Car* corpus, confirming that, although latency grows linearly when parallelization is not possible, it is still almost always possible to produce a response in 1 to 2 seconds or less.

### 8.6.1 Future Work

Perhaps the most significant limitation of the confidence scoring module evaluated in this chapter is that it is only used to produce a binary accept/reject decision. In conversational interfaces, however, there is often a large gray area in which the interpretation of the input and the associated response are "almost correct" in a meaningful and useful way. For instance, a spoken address such as *32 Vassar Street Cambridge Massachusetts* may have been understood correctly with the exception that the street number was understood as *31*. In such cases, it is almost surely more useful to provide the response generated with the small mistake, but provide an easy way to correct it – for example, using the multimodal error correction technique discussed in Section 2.4.1. The confidence scoring module, in its current form, provides no mechanism for learning about such cases or handling them.

However, the architecture developed to support the confidence scoring module does provide a useful starting point for investigating solutions to this problem – as does the data gathered in the *Car* and *Car-Pilot* corpora. Because multiple recognition hypotheses are processed, and multiple candidate responses are produced, it may be possible to evaluate the variation in the produced responses to create concept-level confidence scores. For example, if each generated response agrees on the street name, city, and state of an address, it might be possible to assign high concept-level scores to these concepts, but a lower score to the street number. This information could then be used by the conversational interface to decide how best to respond.

# Chapter 9

# Contextual Utterance Shaping

The previous four chapters have focused on mechanisms for using contextual information to improve the natural language understanding capabilities of spoken and multimodal interfaces, both by predicting what users are likely to say (Chapters 5 and 6), and by entertaining multiple hypotheses as to how best to respond (Chapters 7 and 8). While using contextual information to improve the system's ability to understand and respond to natural language is, as has been shown, quite useful, this chapter focuses on a completely different mechanism for using contextual information to improve accuracy: shaping user behavior.

One reason that word error rates may be high in conversational interfaces is that users use unexpected words or phrasings. This may happen because a user believes the system is capable of performing an action that it is not; for example, a user might try to use a flight reservation system to book a hotel. Or, a user simply might not know how to access a capability in a way that the system can understand; for example, a user might ask *City Browser* for restaurants "in the vicinity" of an address. While it might know how to look for restaurants near that address, it might not understand the phrase "in the vicinity".

Such situations are difficult for system designers: they can't anticipate every word or phrasing that a user might use, nor can they anticipate each user's beliefs about system capabilities. Moreover, the more facile an interface becomes with its use of natural language, the more users may tend to overestimate its capabilities, leading to higher error rates [27]. Speech recognizers are unable to correctly transcribe words that are not in their vocabulary, and statistical language models give low scores to previously unobserved sequences of words. Moreover, even if a user's utterance is transcribed perfectly by the speech recognizer, the natural language understanding components of a conversational interface may not be able to make sense of an unfamiliar construction. In short, while natural language interfaces aim to support a wide variety of natural language constructs, they are only actually capable of understanding a fraction of what a user might reasonably expect to be able to say.

The problem of users knowing the bounds of a system's capabilities and the natural language used to access those capabilities is particularly acute in interfaces like *City Browser*. When interacting with *City Browser*, users are in full control of the conversation – there are very few prompts that guide a user down a particular path.

While this freedom may be quite useful for experienced users, it can be daunting for those who are not as familiar with the interface, because it may be difficult for them to even understand the extent of what they can ask for. During the *Tablet* data collection effort described in Chapter 4, it became clear that even if subjects were given a general idea of the system's capabilities, they were often unsure as to how they should access them via natural language. A common problem was not being sure how to properly decompose a task in a way that *City Browser* could understand. For example, getting directions to a cheap Italian restaurant in Cambridge might involve first searching for restaurants matching this constraint, finding a good one, and then asking for directions to it from a particular address. Even given this task decomposition, users might have difficulty finding the right words and gestures to accomplish these actions—by, for example, saying "I'd like a cheap Italian restaurant in Cambridge", clicking on one, and then saying "Give me directions to this restaurant from 32 Vassar Street, Cambridge". Finally, when users did manage to speak in a way that the system *should* be able to understand, errors sometimes nonetheless prevented the system from responding appropriately. In this case, subjects often either assumed the system did not have the capability they were interested in, or that they needed to access that capability using different natural language constructs. They didn't know whether to repeat, rephrase, or rethink.

More succinctly, the challenge can be stated as follows: without resorting to system-directed dialogue, how can users come to know a conversational interface's capabilities, and the sorts of natural language constructs that can be used to access them? When users gain an idea of what they can say and do, their behavior becomes more predictable, and this leads to higher speech recognition and natural language understanding accuracy. It's perhaps too obvious to point out, but human users are far more intelligent and flexible than any conversational interface with which they might interact. Why not take advantage of this flexibility to help them behave in a way that can be understood?

This chapter explores an unobtrusive, context-sensitive method for helping users of multimodal conversational systems like *City Browser* understand what they can say (or do) at any point in the conversation. In particular, the interface's graphical modality is used to provide context-sensitive utterance suggestions, which give the user an idea of what he or she might want to say or do next at any given point in the conversation. The goal is to "shape" user behavior – that is, influence its structure and content – so that users can understand the full range of capabilities of the system, and how best to access them with natural language. The multimodal *suggestions module* described in this chapter leverages contextual information in order to provide highly relevant, yet unobtrusive, suggestions that are available to guide users at each conversational turn.

## 9.1 Related Work

Several areas of research share similar aims to the context-sensitive suggestions module presented in this chapter. First, multimodal help systems that provide high

quality, on-demand help to users of multimodal conversational systems also aim to help users understand what they can do next. Second, targeted help modules attempt to steer users in the right direction at a given point in the conversation when their utterances appear to be out of the bounds of what the system can understand. Third, speech interfaces that require users to speak using artificial or formulaic language attempt to reduce input errors by requiring that speech input always follow a particular form. In the remainder of this section, each of these techniques is considered in turn.

### 9.1.1 Multimodal Help Modules

The multimodal help system developed for the MATCH urban information system [42] sets out to solve the same problem as the context-sensitive suggestions module discussed in this chapter: helping users understand what they can say and/or do at any given time. The MATCH system [49] is quite similar to *City Browser*: it provides access to information about restaurants and public transportation information in an urban environment through the use of a multimodal interface that allows a combination of speaking, writing, and drawing. System designers found that users had a difficult time understanding how to use both input modalities, and understanding the features provided by the system. In response, they created a context-sensitive multimodal help system, which can be accessed by asking for help, writing "help", or pressing the help button. Based on the context of the conversation, an appropriate help sub-dialogue begins, where users can find out more about system capabilities. Moreover, the help system provides multimodal examples indicating not only how to speak, but how to draw or write to interact with the system.

The MATCH multimodal help system is somewhat similar to the suggestions module proposed here both in implementation and intent. However, there are several key differences. First, MATCH relies on the user explicitly asking for help, while the suggestions module discussed here unobtrusively offers "always-on" help. Moreover, while it is context-sensitive, the MATCH system is less aggressive about incorporating information from the various databases that provide information content for the application. The suggestions module discussed here is also more sensitive to the semantic content of previous queries, allowing it to offer more targeted *subsetting* and *anaphoric* suggestions. On the other hand, the MATCH system's capability to actually demonstrate how to draw or write during a multimodal command is quite useful.

### 9.1.2 Targeted Help

So-called "targeted help" modules have been developed for several spoken and multimodal dialogue systems. In [45], a targeted help module is developed for the WITAS dialogue system, a multimodal interface to a robotic helicopter [58]. A similar targeted-help module is described in [31], where the domain is the *On/Off House (OOH)* in which a user can control roughly 20 home appliances over the phone. In both cases, the targeted help systems attempt to detect user utterances that fall

| | | |
|---|---|---|
| $U_1$ | User: | Hello? |
| $S_2$ | System: | Hello. How can I help? |
| $U_3$ | User: | Um, can I have some lights on? |
| $S_4$ | System: | Sorry, I didn't quite catch that. To turn a device on or off, you could try something like "turn on the kitchen light". |
| $U_5$ | User: | Turn on the kitchen light |
| $S_6$ | System: | The kitchen light is now on |
| $U_7$ | User: | What other sorts of things are there? |
| $S_8$ | System: | I didn't quite catch that. To find out what devices there are, you could try something like "What devices are there", or "what devices are there in the lounge". |

Figure 9-1: Example dialogue including targeted help, reproduced from Figure 1 in [31].

outside the coverage of the dialogue system, and provide spoken *targeted help* demonstrating relevant examples of in-domain utterances. Figure 9-1 reproduces a short example dialogue from [31].

Given an input utterance, the targeted help module makes two sequential decisions. First, the module decides whether or not the utterance is outside of the coverage of the dialogue system. Such an utterance is typically one that, even if it had been recognized correctly (*i.e.*, a perfect transcription), then the dialogue system would still have had difficulty processing it. For example, a dialogue system that sells train tickets would typically not be able to understand an utterance such as:

*My mother in Boston is sick so I need to go and visit her tomorrow.*

While, to a human, such an utterance would simply imply that the user wants to buy a train ticket to Boston tomorrow, such an utterance is typically not covered by a spoken dialogue system built to sell train tickets. Even presuming the speech recognizer perfectly transcribed this utterance, the system would likely have difficulty parsing it and providing an appropriate response.

In order to determine if an utterance falls outside the dialogue system's coverage, the targeted help module first performs speech recognition using an in-domain language model. In [31] and [45] this is a context-free grammar. The grammar may be hand-crafted, as in [31], or it may be an approximation of the grammar used for natural language parsing as in [45], where the grammar was developed with GEMINI [18]. If the confidence score of the top recognition hypothesis is above a particular threshold, then this recognition result is processed directly by the system. However, if the confidence score falls below the threshold, then the utterance is classified as being outside of the system's coverage. A second recognition pass is then performed, using an $n$-gram language model. In [31], the $n$-gram is trained on a combination of approximately 400 transcribed utterances of users interacting with the dialogue system and recognizer hypotheses for 200 additional in-domain utterances.

If the confidence score resulting from the second recognition pass is high enough, then the hypothesis is passed on to a targeted help agent. The agent formulates a hypothesis about the user's goal, and formulates a sample utterance to show the user

how to accomplish that goal within the confines of the system's language understanding capabilities. Utterances $S_4$ and $S_8$ in Figure 9-1 demonstrate system responses driven by the targeted help agent.

In [45], the targeted help agent distinguishes among three possible sources of error: endpointing errors, utterances containing out of vocabulary words, and unsupported verb subcategorizations (*e.g.*, "zoom in on the red car" when the dialogue system supports only "zoom in"). A rule-based system is used to distinguish among these sources of error, and in-coverage example utterances are provided to the user that match as much as possible the vocabulary and dialogue move type (*e.g.*, wh-question *vs.* yes-no question) detected in the user's utterance.

The targeted help agent implemented in [31] consists of a decision tree that maps hypotheses from the $n$-gram language model to 12 different error classes, for which scripted targeted help messages have been created. Hand labeled recognizer hypotheses were used to train the decision tree. Features drawn from each hypothesis included: words and their confidence scores, the utterance confidence score, utterance length, as well as myriad domain-specific boolean features testing for the existence of lexical items such as "on" or "off".

Evaluation in both systems involved providing the targeted help to some users, and not to others. In [31], a reduced word error rate (39% *vs.* 55%) and an increase in system-covered utterances (47% *vs.* 36%) are reported in the targeted help condition. Importantly, almost all users immediately repeated one of the system's suggested utterances, indicating that it was likely relevant and helpful. In [45], it was determined that users in the targeted help condition gave up on tasks less frequently than those in the control group – in particular, in the first assigned task, 11% *vs.* 45% of users gave up. It was also found that users in the targeted help condition completed tasks more quickly, especially the first one.

Such results are promising because they indicate the ease with which humans can adapt to speaking within the bounds of what the system can understand, so long as they are provided with some indication of how to do this. The suggestions module proposed in this chapter is similar to the targeted help systems discussed in this section as, in both cases, the goal is to provide examples of how to use syntactic structure and vocabulary that can readily be understood by the interface. One key difference is that the targeted help systems only become active *after* an error has been made, while the suggestions module attempts to prevent errors from happening in the first place. Moreover, targeted help relies on the ability to accurately infer the user's intent, so that a limited number of examples may be spoken to the user, while the suggestions module described here is more passive. This means that the targeted help systems may provide "the wrong" suggestion if it misunderstands the user's intent. On the other hand, the suggestions module will *always* provide the user with suggestions that may not be of interest – with the hope that a useful suggestion does appear somewhere in the list.

| | |
|---|---|
| *U1* | Start Over |
| *S1* | Starting over |
| *U2* | Theater is the Manor, title is Abandon. |
| *S2* | The Cinemagic Manor Theatre, Abandon. |
| *U3* | What are the show times? |
| *S3* | 2 Show Times: 1:25, 5:30. |
| *U4* | Start Over |
| *S4* | Starting Over |
| *U5* | Theater is Maxi Saver, what is the address? |
| *S5* | Carmike Maxi Saver 12 Theater: 2001 Mountain View Dr. In West Mifflin |
| *U6* | Start Over. |
| *S6* | Starting Over. |
| *U7* | Options |
| *S7* | You can specify or ask about title, theater, show time, ... |

Figure 9-2: Snippet of an example interaction between a user and a Speech Graffiti interface for obtaining movie information, reproduced from Figure 1 of [83].

### 9.1.3 Artificial Language

While the above approaches have focused on ways to help users communicate with natural language constructs that can be understood by the system, an alternative approach proposed by the Speech Graffiti project [83] is to do away with the use of natural language altogether. Instead of placing the burden on system designers to recognize and understand a variety of language constructs, the burden is placed on the users who must learn an *artificial* or *restricted* language in order to communicate with the system. Such an approach has the key advantage that speech recognition accuracy can be greatly increased, as the vocabulary size will be smaller and the language model less complex, as compared to a similar interface that allows natural language. Moreover, if the artificial language is standardized so that many different speech interfaces make use of it, then users who are familiar with it may quickly – with little or no training – be able to use an unfamiliar speech interface in a new domain. Figure 9-2 gives a short example of an interaction between a user and a system that understands Speech Graffiti. In [83], it is reported that a majority of university students trained in the use of Speech Graffiti prefer it to a similar natural language interface, and that the word error rate of users interacting with the Speech Graffiti system was lower.

## 9.2 Context-Sensitive Suggestions Interface

After observing users interact with the version of *City Browser* used to collect the *Tablet* corpus in the lab, it became clear that if *City Browser* was to be made available on the web, it would have to provide more support to new users to help acquaint them with both the capabilities of the system and the natural language constructs that could be used to access them. The module was designed with the principle that the best user interfaces are often those with which a user can start interacting immediately, without undergoing a lengthy tutorial first. This holds especially on the

web, where users generally expect to browse to a web page and immediately begin to interact with it. Most people, however, have never interacted with a multimodal conversational interface that makes use of speech, a graphical user interface, and the ability to draw. So, it is challenging to design the *City Browser* interface in such a way that users can quickly understand how to interact with it.

The context-sensitive suggestions module presented here is an important component of the graphical user interface aimed at making it easy for users to be able to quickly (a) begin using *City Browser*, and then (b) learn more about the capabilities (and limitations) of the interface during the course of their interactions. The remainder of this section describes the module from the point of view of the user. In the following section, the implementation details of this interface are discussed in detail.

### 9.2.1 Look and Feel

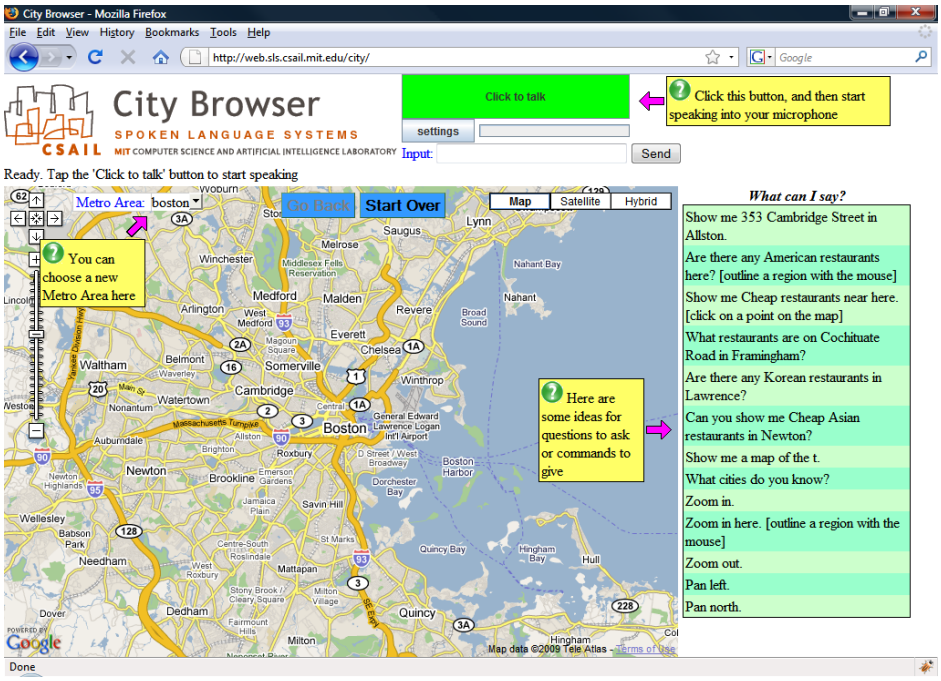
Figures 9-3a and 9-3b show how the context-sensitive suggestions module fits into the user interface of the web version of *City Browser*. As Figure 9-3a shows, when users navigate to the page, they are immediately provided with a list of suggestions on their right-hand side under the header of *What Can I Say?*. In fact, the suggestions are examples not only of what a user can say, but also describe gestures that may be coordinated with certain utterances. After every spoken turn, the list on the right-hand side is updated with a new set of contextually-relevant suggestions. It subtly fades in and out so that the update is obvious, but not distracting. When search results are shown, the frame containing the suggestions reduces in height, as shown in Figure 9-3b. While a few suggestions are still visible at all times, the user can scroll this window to see the complete list (a fact exploited in the usage analysis in Section 9.4).

Figure 9-4 shows how the context-sensitive suggestions list appears in the automotive version of *City Browser* used to collect the *Car-Pilot* and *Car* corpora. In this case, due to the very limited screen resolution available, the list of suggestions is presented as a dedicated screen in the interface. Users can navigate to this screen using the controller. It also automatically appears after two consecutive utterance rejections, where the system indicates non-understanding with an expression like "pardon me".

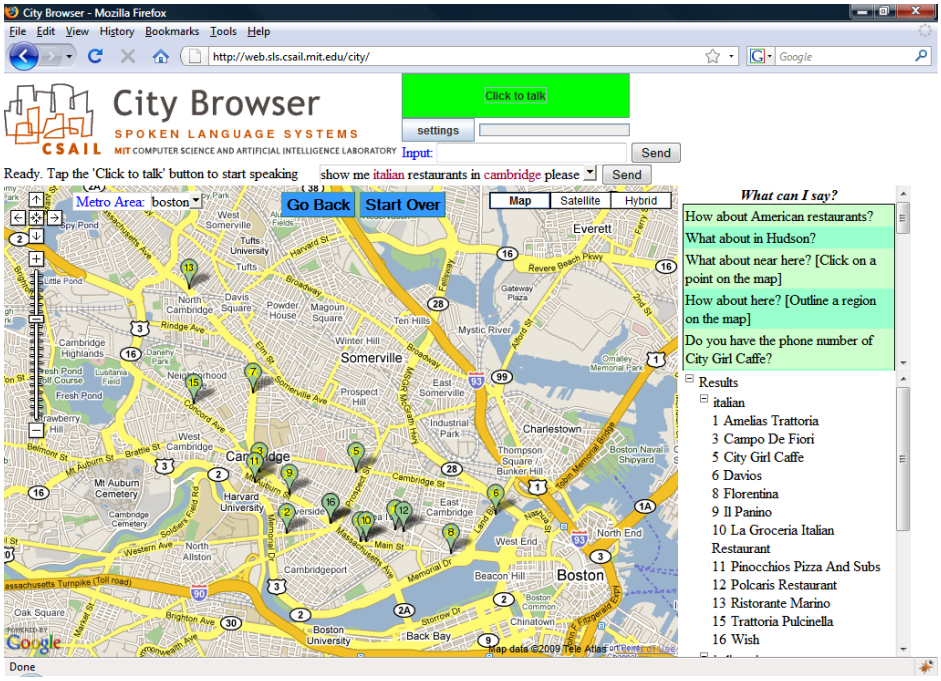
Finally, Figure 9-5 shows how the suggestions list changes depending on the context of the conversation. In Figure 9-5a, the list at the start of the conversation is shown. Then, the user says *Show me Chinese restaurants in Cambridge*, resulting in several search results and the suggestions list shown in Figure 9-5b. Finally, the user asks about a specific restaurant by saying *Where is the Royal East restaurant?*, which leads to the suggestions list in Figure 9-5c. In each case, the context of the conversation heavily influences the type and content of the suggestions shown.

### 9.2.2 Goals

Taken as a whole, the context-sensitive list of suggestions is intended to be *complete*, while each suggestion individually should be *varied*, *valid*, and *productive*. The




(a) At the start of an interaction.




(b) Following the user's request to *Show me Italian restaurants in Cambridge.*

Figure 9-3: Screenshots of *City Browser* showing context-sensitive suggestions in the interface as it appeared during the collection of the *Web* corpus (see Chapter 4). The suggestions appear on the right-hand side of the screen, and are updated to reflect the context of the conversation. For instance, in (b), they incorporate the context provided by the user's previous search constraints, and specific search results from that query. Note that once search results are shown, the frame containing the suggestions list becomes shorter, and a scrollbar appears to allow browsing the complete list.

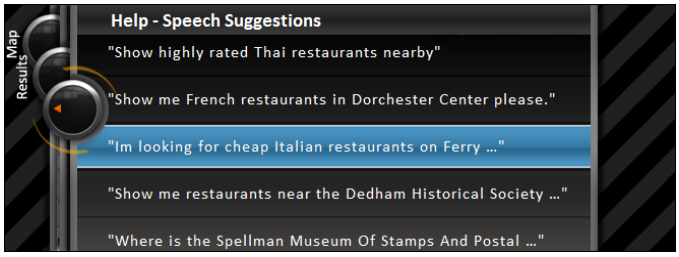



Figure 9-4: Screenshot of the suggestions screen shown as part of the automotive version of *City Browser* used to collect data for the *Car-Pilot* and *Car* corpora. Subjects could navigate to this screen on their own, or were brought automatically to it after two rejections.

remainder of this section discusses each of these concepts in turn.

**Complete** If a user explores the suggestions list as it changes over time, he should eventually obtain a *complete* understanding of the capabilities available in the interface. Moreover, he should understand how to use natural language and other modalities to access these capabilities.

**Varied** For each system capability – for example, the capability to find restaurants near a particular address – the user should encounter a *varied* set of suggestions for accessing that capability. In this way, over time, a number of natural language constructs and vocabulary items that can be understood by the system should become apparent over the course of an interaction. This capability is critical, simply because different speech patterns and lexical choices seem to work better for different people, depending on their accent, speaking rate, pitch, and so forth. If users are exposed to a number of different phrasings, eventually they may find one that works well for them.

In addition to varying in syntactic structure, suggestions should also vary in content. For example, the cuisine and city in a suggestion like *Show me Chinese restaurants in Cambridge* should rotate over time to include other cuisine types and cities.

**Valid** Suggested utterances should always represent *valid* utterances that a user could actually read back verbatim to the system, with a reasonable hope of being understood. This means not only that their grammatical structure should be valid, but they should incorporate real items drawn from the system's database resources. While *Show me Italian restaurants in Paris* may serve as a good example of the type of utterance that *City Browser* understands, it is not valid according to this definition, because it can not actually be understood by the system: the city name *Paris* is not in the speech recognizer's vocabulary, nor does the database contain any restaurants in Paris.

Moreover, the constraint that a suggestion be *valid* also must include contextual considerations. For instance, consider the suggestion *Tell me its phone number*. This utterance makes perfect sense if the user has just been told the address of a particular

Show me Pizza restaurants in Westborough please.
I'm looking for cheap Barbecue restaurants on East Main Street in Milford.
Show me 672 Truman Highway in Hyde Park.
Are there any museums near 110 Huntington in Boston.
What is the nearest subway station to 200 Westgate Drive in Brockton?
Are there any American restaurants here? [outline a region with the mouse]
Show me cheap restaurants near here. [click on a point on the map]
What restaurants are on Market Square in Lynn?
Are there any American restaurants in Bedford?
Can you show me cheap Japanese restaurants in Norwood?
Show me a map of the t.
Hide the map of the subway
What cities do you know?
Zoom in.
Zoom in here. [outline a region with the mouse]
Zoom out.
Pan left.
Pan north.

(a) Initial suggestions list

How about American restaurants?
How about in Brookline?
Tell me the phone number of New Mary Chung Restaurant.
Can you tell me the address of Yun Yun Kitchen?
Do you have hours for Rangzen Tibetan Restaurant?
Show me the web page for Guangzhou Restaurant.
Show me American restaurants in Needham please.
I'm looking for moderately priced Nouvelle Cuisine restaurants on East Main Street in Gloucester.
Give me directions to Qing Dao Garden Restaurant
Show me 82 Concord Street in Framingham.
Are there any museums near 725 Boylston Street Floor 1 in Boston.
Are there any t stations near Yun Yun Kitchen?
What is the nearest t station to 205 L Street in South Boston
Tell me about these [circle a few restaurants with the mouse]
Show me the moderately priced ones.
Are any of these recommended?
Are there any American restaurants here? [outline a region with the mouse]
Show me cheap restaurants near here. [click on a point on the map]
What restaurants are on Holland Street in Somerville?
Are there any American restaurants in Watertown?
Can you show me cheap Pizza restaurants in Boston?
Show me a map of the subway
Hide the map of the subway
What cities do you know?
Zoom in.
Zoom in here. [outline a region with the mouse]
Zoom out.
Pan left.
Pan north.

(b) Suggestions list following the user utterance: *Show me Chinese restaurants in Cambridge*

Figure 9-5: Suggestions list evolving over the course of a dialogue (continued on next page).

| |
|---|
| Show the web page for Royal East. |
| Show me Chinese restaurants in Brighton please. |
| I'm looking for cheap Pizza restaurants on Chatham Street in Lynn. |
| Give me directions to Royal East |
| Show me 212 Holland Street in Somerville. |
| Do you have the phone number? |
| Do you have the hours? |
| Show me the web page for this restaurant |
| What is its address? |
| Show me museums close by. |
| Are there any museums near 105 Market Street in Lowell. |
| What hotels are near here? |
| Show me restaurants close by. |
| What t stations are near here? |
| What t stations are near Royal East? |
| What is the nearest t station to 190 High Street in Boston |
| Give me driving directions to here from 259 Newbury Street in Boston. |
| Are there any Italian restaurants here? [outline a region with the mouse] |
| Show me cheap restaurants near here. [click on a point on the map] |
| What restaurants are on Commonwealth Avenue in Boston? |
| Are there any Pizza restaurants in Framingham? |
| Can you show me cheap Pizza restaurants in Brockton? |
| Show me a map of the t. |
| Hide the map of the t. |
| What cities do you know? |
| Zoom in. |
| Zoom in here. [outline a region with the mouse] |
| Zoom out. |
| Pan left. |
| Pan north. |

(c) Suggestions list following the user utterance: *Where is the Royal East restaurant?*

Figure 9-5: Suggestions list evolving over the course of a dialogue (continued from previous page).

hotel. However it is not valid if the user has just obtained driving directions between two addresses.

**Productive** Each suggestion should be *productive*; specifically, it should produce some sort of useful or interesting result if the user does choose to read it verbatim. For instance, a suggestion like *Show me Armenian restaurants in Somerville* is not productive because it leads to the uninteresting response *There are no Armenian restaurants in Somerville.* On the other hand, *Show me Chinese restaurants in Somerville* is a productive suggestion, because several matching restaurants will be shown to the user.

A suggestions module that produces suggestions having the properties discussed above has three major potential benefits. First, it provides a way for a user with a particular goal in mind to quickly scan a list of suggestions, which ought to give an idea of how to use natural language and/or gestures to accomplish a goal – or, perhaps, indicate that there is no way to accomplish their goal using the interface. Second, because it is context-sensitive, it allows users to easily learn about system capabilities over the course of using the interface; for example, a user who sees a suggestion involving driving directions to a displayed search result, might learn that *City Browser* can, in fact, display driving directions. Or, because varied content words are used – like city names, street names, and cuisine types – users get a general impression of the range of what's available in the system's database. For example, a user might think only major cities are supported, and then be surprised to see the suburb they live in mentioned in a suggestion. Third, by viewing varied productive suggestions, users who have trouble with speech recognition accuracy may be able to experiment until they find syntactic constructs that work well for them; if they speak a productive suggestion verbatim and are understood, their success will be reinforced by the fact that they receive an interesting result.

### 9.2.3 Suggestion Types

A major focus of the *City Browser* interface is enabling users to explore sets of database entries that match a set of constraints. As such, many of the suggestions produced by the suggestions module are aimed to help users understand how to first specify a set of search constraints, and then further refine or revise them. For the most part, the suggestions generated by the module fall into one of the following context-sensitive categories.

**Globally relevant suggestions** These are utterances that are relevant in any context, such as map commands (*Pan right* or *Zoom in*), queries about addresses (*Show me 32 Vassar street in Cambridge*), driving directions (*Directions from 32 Vassar street in Cambridge to 55 Boylston street Boston*), public transportation (*Show me subway stations in Somerville*), and search (*Show hotels in Palo Alto*).

**Subsetting suggestions** These are utterances that allow the user to narrow down a list of results returned from a previous search query. There are two forms of *subsetting* suggestions. First, multimodal ones, such as *Tell me about these [Circle a few restaurants with the mouse]*, allow the user to zero in on a smaller set. Second, suggestions that subset by *attribute* show how properties of database entries can be used to narrow down the set, as in, *Show me the highly rated ones*. Properties that were not mentioned in the user's previous query are suggested, since these will further refine the search results.

**Anaphoric suggestions** A user will often want to get more information relating to a particular search result. Two types of suggestions are produced for these cases. If a single search result is currently salient, then *anaphoric* suggestions are provided that relate to an attribute of that entity, such as *Tell me its phone number*. When several search results are available, suggestions are offered that relate to a specific one, such as *Can you tell me the address of the Museum of Fine Arts?*. In addition to querying about a particular property, users may also use one of the search results as a reference point for performing another action, as in *Are there any subway stops close to the Royal East restaurant?* or *Please give me driving directions from 77 Massachusetts avenue to the Royal East restaurant*.

**Contrastive suggestions** A nice aspect of using natural language to explore a database is that it is quite natural to build on a dialogue by retaining some attributes of a search query and replacing others. For example, if a user has just said *Can you show me the subway stations in Cambridge*, it is quite natural to follow up with a query such as *What about in Brookline?*. Search constraints specified previously by the user are used to guide the creation of such contrastive suggestions, which suggest alternative values for the constraint. Multimodal contrastive suggestions may also be produced, as in *What about near here? [Click on a point on the map]*.

## 9.3 Templates

In order to produce the high-quality suggestions described in the previous section, a module was created that fills in a set of hand-authored templates. Templates make use of the following contextual information:

- Any current database search constraints (*e.g.*, `cuisine=italian,city=cambridge`),
- Any currently displayed ("in-focus") database search results (*e.g.* the set of Italian restaurants in Cambridge),
- The databases available to the application (*e.g.*, containing restaurants, hotels, *etc.*).

Specifically, each group of natural language templates is associated with a set of constraints, listed in Figure 9-6. First, *matching constraints* are used to decide if

the group of templates are applicable, given the current context of the conversation. Then, *language generation constraints* are used to decide how to fill in a randomly chosen template. Each template may contain keys, such as :CUISINE, which must be filled in with values from a database entry. This database entry may either be taken from those in the currently "in-focus" set of search results, or it may be chosen at random – subject to particular search constraints – from one of the databases available to the system.

### 9.3.1 Examples

The simplest natural language suggestions to generate are the *global* ones, which are meant to give examples of utterances that can be said in any context. Figure 9-7 gives two example global template-group constraints, each of which makes use of a randomly chosen database entry. By filling in values from an actual database entry, each is guaranteed to be *valid* and *productive*. The address mentioned in *Show me 1500 Church street in San Francisco* will be an actual, valid address. The query *Show me Chinese restaurants on Church Street in San Francisco* will produce at least one search result.

Figure 9-8 shows examples of how both *subsetting* and *anaphoric* suggestions are created. In this case, rather than using a database entry at random, these templates are filled in using values from one of the database entries from the "in-focus" set created from the user's search constraints. In the examples shown, it is assumed that the user has just said *Show me cheap Indian restaurants in Cambridge*. This leads to the search constraints shown in Figure 9-8a, resulting in several matching database entries – an example of one of which is shown in Figure 9-8b. The subsetting suggestion, *Show me the recommended ones*, is used because, although the search results contain entries of type RESTAURANT, the user did not specify a constraint based on whether or not these restaurants have a value set for their *recommendation* attribute. Since a database entry in the set of results does have this attribute, its value for the *recommendation* key is used to fill in the template. Similarly, the anaphoric suggestion, *Tell me the phone number of India Castle*, can be filled in by choosing the *name* attribute of any matching database entry.

Note that the subsetting template is only used if there are at least 10 database entries visible – if there are fewer, then this template is not likely to be very useful. Moreover, the entries must be of type RESTAURANT – otherwise, this suggestion wouldn't make much sense since there are no recommendations in the hotels and museums databases. The anaphoric template, on the other hand, will be used when there is at least one database result, and it may be used for restaurants, hotels, and museums – all of which have phone numbers.

Finally, Figure 9-9 shows how the contrastive suggestion *What about in Boston?* is produced in the same conversational context. In particular, the search constraints in Figure 9-9a are for a type RESTAURANT and contain the key *city*, which means that the template group in Figure 9-9b will match. Since this template is marked as being *contrastive*, a new database query will be produced (shown in Figure 9-9c) that is identical to the original query, except that it requires the *city* attribute to have

| *Constraint* | *Description* | *Example* |
|---|---|---|
| **Matching Constraints: Search Query** | | |
| *required_keys* | Requires that each of the specified keys appear in the user's search constraints | CUISINE, CITY |
| *excluded_keys* | Requires that none of the specified keys appear in the user's search constraints | CUISINE, CITY |
| **Matching Constraints: "In-focus" Search Results** | | |
| *min_focus* | Requires that at least this many search results be "in-focus" | 5 |
| *max_focus* | Requires that no more than this many search results be "in-focus" | 1 |
| *types* | Requires that one or more "in-focus" search results have at least one of the specified types | RESTAURANT, HOTEL |
| *exclude_types* | Requires one or more "in-focus" visible search results that do not have any of the specified types | RESTAURANT, HOTEL |
| **Matching constraints: Databases** | | |
| *required_features* | Requires that the system have access to a database containing entries of this type | HOTEL |
| **Language Generation Constraints** | | |
| *from_db* | If true, template will be filled in with values from an entry in the database; otherwise, it will be filled in with values from one of the user's search results | TRUE |
| *database_name* | Specifies that random database entries should be drawn from the database containing this type of entry | HOTEL |
| *contrastive* | If true, a database entry will be chosen that fits the current search constraints, except that any *required_keys* constraints will be negated; otherwise, a database entry will be chosen at random | TRUE |
| **Language Generation Templates** | | |
| *templates* | List of templates to use if all constraints are satisfied, one of which will be chosen at random | *What about* :CUISINE *restaurants?* |

Figure 9-6: Constraints used to choose an appropriate natural language generation template based on context, and then fill in that template. Matching constraints are used to determine if the template group is relevant, given the context of the conversation. They may be based on the user's search query, the results of that search query, and the set of databases available to the application. If conditions are met, then one of the language generation templates will be chosen and filled in, using either values from one of the "in-focus" search results, or from an appropriately chosen database entry, depending on the specified language generation constraints.

**Random Database Entry**

| | |
|---|---|
| type: | RESTAURANT |
| name: | ERIC'S RESTAURANT |
| price_range: | CHEAP |
| cuisine: | CHINESE |
| city: | SAN FRANCISCO |
| street_num: | 1500 |
| street: | CHURCH STREET |
| neighborhood: | NOE VALLEY |
| phone: | (415) 555-5555 |
| recommendation: | RECOMMENDED |

(a) A randomly chosen database entry used to fill in values in both templates below.

**Global Template Group**

| | |
|---|---|
| from_db: | TRUE |
| templates: | *Show me* :STREETNUM :STREET *in* :CITY. |
| | *Where is* :STREETNUM :STREET *in* :CITY. |

(b) Global suggestion template, which when paired with the randomly chosen database entry produces the suggestion: *Show me 1500 Church street in San Francisco.*

**Global Template Group**

| | |
|---|---|
| from_db: | TRUE |
| required_features: | RESTAURANT |
| templates: | *Show me* :CUISINE *restaurants on* :STREET *in* :CITY. |
| | *Are there any* :CUISINE *restaurants on* :STREET *in* :CITY. |

(c) Global suggestion template, which when paired with the randomly chosen database entry – which must be of type RESTAURANT – produces the suggestion: *Show me Chinese restaurants on Church Street in San Francisco.*

Figure 9-7: Example global suggestion template groups that fill in values from a randomly selected database entry. Since actual database entries are used, the address in (b) should really exist, and there must be at least one search result for the query in (c).

**Constraints**

| | |
|---|---|
| type: | RESTAURANT |
| price_range: | CHEAP |
| cuisine: | INDIAN |
| city: | CAMBRIDGE |

(a) Original search constraints, generated by the user's utterance *Show me cheap Indian restaurants in Cambridge.*

**Example Matching Database Entry**

| | |
|---|---|
| type: | RESTAURANT |
| name: | INDIA CASTLE |
| price_range: | CHEAP |
| cuisine: | INDIAN |
| city: | CAMBRIDGE |
| street_num: | 928 |
| street: | MASSACHUSETTS AVENUE |
| neighborhood: | HARVARD SQUARE |
| phone: | (123) 456-7890 |
| recommendation: | RECOMMENDED |

(b) One of the matching database entries.

**Subsetting Template Group**

| | |
|---|---|
| types: | RESTAURANT |
| min_focus: | 10 |
| excluded_keys: | :RECOMMENDATION |
| templates: | *Show me the* :RECOMMENDATION*ones.* |
| | *Are any of these* :RECOMMENDATION*?* |

(c) Subsetting template group that matches because there are 10 or more matching restaurant database entries, and because the search query did not include a recommendation attribute. Using the matching database entry, the following suggestion can be produced: *Show me the recommended ones.*

**Anaphoric Template Group**

| | |
|---|---|
| types: | RESTAURANT |
| | HOTEL |
| | MUSEUM |
| min_focus: | 1 |
| templates: | *Tell me the phone number of* :NAME. |
| | *What's the telephone number for* :NAME*?* |

(d) Anaphoric template group that matches because at least one restaurant matches the query. Using the matching database entry, the following suggestion can be produced: *Tell me the phone number of India Castle.*

Figure 9-8: Examples of subsetting and anaphoric suggestions templates. (a) The user's search constraints, which match a set of database entries; one of which is shown in (b). Values from this database entry are used to fill in the templates in (c) and (d) to produce suggestions.

a different value (as indicated by !CAMBRIDGE). One of the database entries that matches this new search is shown in Figure 9-9d. The *city* attribute of this matching entry is then used to fill in the template. In this way, the suggestion is guaranteed to be *productive*, since it will find at least the entry used to fill in the template.

## 9.4 Evaluation

The suggestions module figured most prominently in the collection of the *Web* corpus of *City Browser* interaction data, where its context-sensitive suggestions were always visible to the user on the right-hand side of the screen. While users were generally able to complete most or all of the tasks completely on their own, it's hard to know exactly how much help the suggestions gave them. Short of doing an experiment in which some users have access to the suggestions and others don't, it's difficult to directly measure their usefulness. In this section, the results of two types of evaluation are presented using the *Web* corpus. First, survey results are presented that show that subjects generally noticed and found useful the context-sensitive suggestions module. Second, an analysis of usage data from the corpus shows how frequently subjects interacted with the suggestions module, and what effect this had on *City Browser's* response accuracy.

### 9.4.1 Survey Results

After completing their interaction with *City Browser*, subjects in the *Web* study completed a survey about their experience. Two of the questions had to do specifically with their impressions of the suggestions module. In each case, subjects responded with a rating on a seven-value Likert scale ranging from *strongly disagree* (1) to *neutral* (4) to *strongly agree* (7). Examining the results of these survey questions indicates that subjects generally both noticed the context-sensitive suggestions and found them useful.

The first question asked subjects whether they agreed with the statement: *I noticed the suggestions listed on the right-hand side of things I could say.* The histogram in Figure 9-10a shows that nearly all users took note of it – it's hard to judge what the few "neutral" responses mean, perhaps some subjects did not understand the question. The statement immediately following on the survey was *I found these suggestions pertinent and useful*, with which the vast majority of users agreed, as shown in Figure 9-10b.

### 9.4.2 Usage Data

In addition to looking at survey data, it's also possible to get an idea of the usefulness of the suggestions module by looking at some of the usage data from the *Web* corpus. In particular, one simple quantitative measure is available: the number of times users scrolled the frame containing the list of suggestions. As the screenshot in Figure 9-3b above shows, once search results are displayed, the user must scroll to

**Constraints**

| | |
|---|---|
| type: | RESTAURANT |
| price_range: | CHEAP |
| cuisine: | INDIAN |
| city: | CAMBRIDGE |

(a) Original search constraints, generated by the user's utterance *Show me cheap Indian restaurants in Cambridge.*

**Contrastive Template Group**

| | |
|---|---|
| types: | RESTAURANT<br>HOTEL<br>MUSEUM |
| required_keys: | CITY |
| contrastive: | TRUE |
| templates: | *What about in* :CITY*?*<br>*How about in* :CITY*?* |

(b) Contrastive template group.

**Constraints**

| | |
|---|---|
| type: | RESTAURANT |
| price_range: | CHEAP |
| cuisine: | INDIAN |
| city: | !CAMBRIDGE |

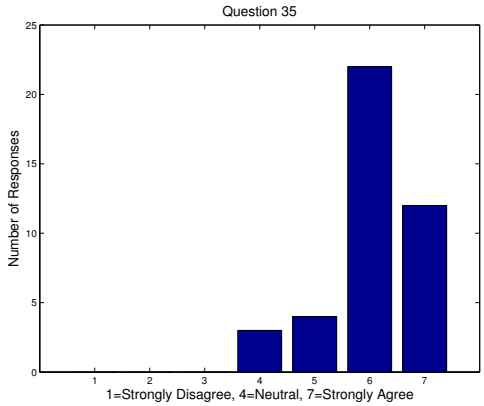
(c) Search constraints used to find a database entry to provide values for a contrastive suggestion.

**Example Matching Database Entry**

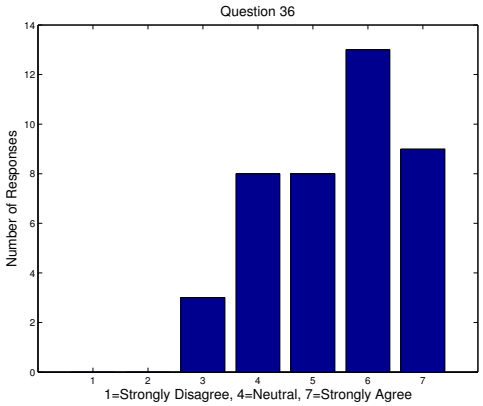
| | |
|---|---|
| type: | RESTAURANT |
| name: | GOURMET INDIA |
| price_range: | CHEAP |
| cuisine: | INDIAN |
| city: | BOSTON |
| street_num: | 800 |
| street: | BOYLSTON STREET |
| neighborhood: | BACK BAY |
| phone: | (555) 555-5555 |
| recommendation: | HIGHLY RECOMMENDED |

(d) One of the matching database entries.

Figure 9-9: Example of the contrastive suggestion *What about in Boston?*. The user's original search constraints are shown in (a), which match the contrastive template shown in (b). Based on this template, the search constraints in (c) are generated, which match a set of database entries – one of which is shown in (d). This entry is used to fill in the key :CITY in the contrastive template group shown in (b).




(a) Histogram of subject responses to the question *I noticed the suggestions listed on the right-hand side of things I could say.* mean=6.05, std=.84




(b) Histogram of subject responses to the question *I found these suggestions pertinent and useful.* mean=5.4, std=1.2

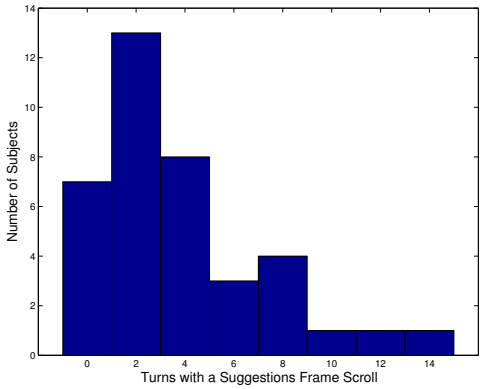
Figure 9-10: Survey results from the *Web* corpus pertaining to the context-sensitive suggestions module.

see any suggestions beyond the two or three ones at the top of the frame. Thus, users who scroll the suggestions frame must, at least, have noticed the suggestions, and suspected that a suggestion that is not currently visible might be useful. Tracking the number of subjects who scroll the suggestions frame should provide an indication of the number who at least noticed that this help feature was available. Of course, some subjects might have made use of the suggestions visible at the top of the frame, which doesn't require scrolling. Others may have scrolled the frame, but not found a useful suggestion.

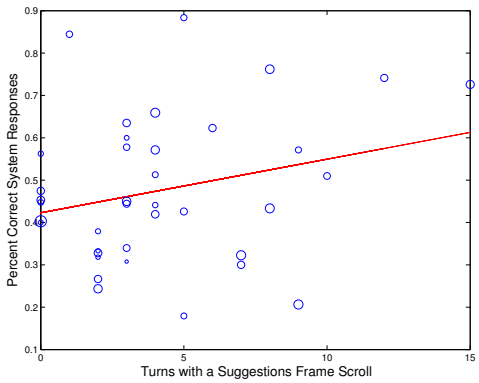
Figure 9-11a analyzes how often subjects scrolled the suggestions frame in the *Web* corpus. Each turn in which a subject scrolled the suggestions frame was counted, where a turn was defined as the span of time between system utterances. The histogram counts the number of subjects who scrolled the frame a specific number of times. Nearly all subjects scrolled the frame more than once, and most scrolled it several times. This indicates that, at the very least, most users were aware of the suggestions, and interested enough to look through them a few times.

Figure 9-11b explores one measure of the usefulness of the suggestions. It plots the percent of instances in which the system responded correctly to a user's utterance – as labeled by annotators – as a function of the number of times the user scrolled the suggestions frame. As the plot indicates, users who scrolled the suggestions list frequently had successful interactions with the system. Users who scrolled less frequently, or not at all, had a more mixed experience.

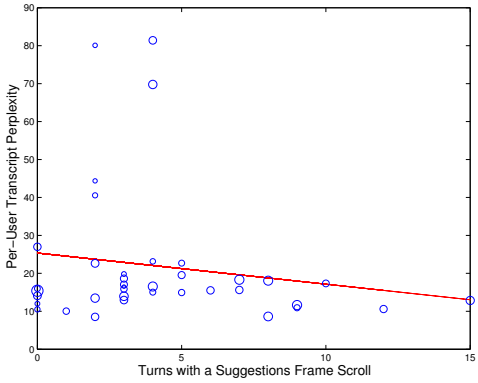
The correlation coefficient for the two variables plotted in Figure 9-11b is 0.26, indicating a weak positive correlation between the number of scrolls and the percent of correct responses. However, if a t-test is used to test the null hypothesis that this correlation is due to chance, then $p = 0.12$, indicating the correlation is not statistically significant. If only subjects who scrolled the frame two or more times




(a) Histogram of the number of users who scrolled the suggestions frame during a specific number of turns, with a bin size of 2.




(b) The circles indicate the percent of system responses that were labeled as correct for a specific user as a function of the number of turns during which that user scrolled the suggestions frame. The size of the circle is proportional to the total number of that user's utterances. The line is a best-fit line.




(c) The circles indicate the perplexity of each user's transcribed utterances as a function of the number of turns during which that user scrolled the suggestions frame. The size of the circle is proportional to the total number of that user's utterances. The line is a best-fit line.

Figure 9-11: Usage and effect of the context-sensitive suggestions in the *Web* corpus.

are considered – in other words, subjects who not only noticed the suggestions list but found it useful enough after a preliminary examination to use it again – then the correlation coefficient is 0.39. In this case, $p = 0.03$, indicating statistical significance.

Finally, Figure 9-11c considers the perplexity of each user's transcribed utterances with regards to the speech recognizer's language model as a function of the number of turns where that user scrolled the suggestions frames. Perplexity gives a rough measure of how well matched the structure and content of a user's utterance is to what the system expects to hear. Subjects who scrolled the suggestions frame frequently tended to speak in a way that resulted in relatively low perplexity, while other subjects had more mixed results. In this case, the correlation coefficient is $-0.16$, indicating a slight, though not statistically significant ($p = 0.35$), negative correlation.

Overall, the usage analysis indicates that subjects do, indeed, find the suggestions list. However, it's not clear that those who scrolled it more frequently derived a particular benefit. It may simply be that scrolling does not correlate well with *looking* at it. Or, it may simply be that a larger sample size is necessary to draw more firm conclusions.

## 9.5 Conclusion

This chapter explored a novel way in which contextual information can be used unobtrusively to help users understand the capabilities of a multimodal conversational interface, as well as how to interact with the interface in a way that it can understand. Specifically, a context-sensitive suggestions module was described, which makes use of the user's previous search constraints, the "in-focus" search results currently visible on the display, and the databases available to the multimodal interface. The module provides high quality, context-sensitive natural language utterance suggestions for the user, which are updated over the course of the conversation.

The suggestions module was integrated into the *City Browser* system to help users interact with the system who may have never used it, or seen it used, before. In the web-based version, suggestions produced by the module were displayed unobtrusively on the side of the screen, and updated after each user utterance. It was determined that almost all subjects in the *Web* corpus discovered these suggestions and found them useful enough to scroll through them on two or more occasions. Moreover, most subjects indicated in a post-experiment questionnaire that they found the suggestions pertinent and useful.

An interesting avenue of future work would be to consider the problem of ranking the suggestions produced by the module described in this chapter. While the hand-crafted templates allow for the creation of high-quality, relevant suggestions, the problem of ordering these suggestions was glossed over. At the moment, the order in which the template groups appear determines the order of the produced suggestions, allowing the developer only limited control over their ordering. However, it would certainly be useful if the most relevant suggestions appeared at the top of the list, and perhaps were even highlighted in some way. Such an ordering could be performed by hand, but a perhaps more fruitful strategy would be to use corpus data to determine

what users often *do* say next in a particular context; these statistics could then influence the ordering of the generated suggestions. A "simulated user", commonly used in reinforcement learning for dialogue system strategies (*e.g.*, [24]) could perhaps be adapted to this task.

# Chapter 10

# Conclusion and Future Work

This thesis has reported on steps toward a pragmatic goal: transitioning multimodal conversational interfaces from prototype systems that make interesting demonstrations, to truly useful tools available to a wide audience of users. Contributions were made toward overcoming two important barriers to making conversational multimodal interfaces more widespread: *availability* and *usability*.

Generally speaking, multimodal conversational interfaces are designed and tested by researchers in the lab, and are not made available to a large number of users. This thesis presents a new framework for delivering multimodal interfaces via the World Wide Web, making them accessible to anyone with a web browser and a microphone. The framework was developed in the context of *City Browser*, a rich conversational interface that allows users to use natural language and gestures to obtain urban information. *City Browser* is a proof-of-concept that conversational interfaces may be deployed as web applications. Moreover, it has been used successfully to collect usage data, both from users who happen to visit the site, and via controlled experiments conducted entirely via the web.

Usage data collected with *City Browser* has been used, in turn, to study the challenges of making multimodal conversational interfaces truly usable. Perhaps the most daunting usability challenge facing conversational interfaces is the fact that speech recognition and natural language understanding errors are inevitable. A successful interface must provide mechanisms to prevent these errors in the first place, detect when they nonetheless occur, and provide strategies for recovering from them. Several techniques falling along this spectrum were discussed in this thesis, which focused on techniques that made critical use of *conversational context*.

Context-sensitive language modeling, via the use of contextualized semantic classes that can be updated based on conversational context, provides a way to alter the system's expectations about what a user is likely to say as the conversation progresses. As this thesis has shown, such techniques can prevent speech recognition errors, leading to a more accurate hypothesis of what words a person said. In several conditions, the use of contextualized semantic classes was shown to lead to significant reductions in word and concept error rates.

Response confidence scoring provides a mechanism both for preventing and detecting errors. The speech recognizer's N-best list is used to produce a set of candidate

system responses based on interpreting each hypothesis in context. Each candidate response is scored using acoustic information and information drawn from the process of producing the response. Errors may be prevented, because the module may choose the highest scoring response, not simply the one generated by the top scoring recognition hypothesis, as is traditionally done. Moreover, speech recognition and natural language understanding errors may be detected when all candidate responses are low scoring; in this case, the system can choose none of the responses, and behave in a more appropriate manner.

Context-sensitive utterance suggestions provide a way to shape user behavior. They are updated as the conversation proceeds, so that they are always relevant, and are shown unobtrusively to the user via the graphical interface. They both help to prevent errors from occurring in the first place, and assist users in recovering when errors do occur by offering them alternate strategies for achieving a goal. Suggestions help a user to understand what capabilities a natural language interface has, and how to access those capabilities in a way that the system is likely to understand.

## 10.1 Future Work

In previous chapters, interesting avenues of future work have been mentioned that pertain to each of the newly developed modules in isolation. This section discusses future work at a higher level, considering the goal of making multimodal interfaces – whether conversational or not – available to, and usable by, a wider audience.

Perhaps the single most important reason that spoken natural language interfaces are not more widespread is that the significant number of errors made by the speech recognizer severely restrict the usability of such systems. While this thesis has presented several contributions toward reducing the number of errors made, and toward mitigating problems caused by errors when they do occur, these technique in no way come anywhere close to solving the problem. Indeed, viewed from a wide-angle lens, they represent incremental improvements to the status quo, when a paradigm shift [55] (albeit a modest one) in how speech recognition is approached may be what is ultimately requisite before systems that interact in a truly natural way can be created.

Despite the limitations of speech recognition technology, it is possible – as this thesis has demonstrated – to build relatively robust, multimodal natural language interfaces that provide some useful functionality. However, if such interfaces are to be widespread, it must become possible for developers without expertise in speech technologies to be able to build them as well. If expertise in the underlying network technologies were necessary to construct a simple web page, the World Wide Web would not exist in its present form. Similarly, it ought to be possible for someone with little or no knowledge of how speech recognition works to build at least a simple multimodal speech interface in a matter of hours. If this were possible, the number of compelling multimodal speech interfaces ought to increase by several orders of magnitude from the handful that exist today.

*City Browser* has demonstrated that multimodal interfaces can potentially be

made available via the web. However, it does little to reduce the expertise required to build systems like it. Given its network-centric architecture, however, it is quite feasible to make its speech and natural language processing components available to other web developers. A first step, which is currently being pursued, is to make the speech recognizer available as a web service. So, with a few lines of code, a developer can incorporate speech capabilities into a web page. Already, this greatly lowers the bar in terms of expertise and software required to build a multimodal interface using speech – an interface, moreover, which is instantly available to a world-wide audience via the web. If other natural language processing components can also be made available as services, such as the parser and natural language generation components used by *City Browser*, web developers could similarly integrate such technology into their sites.

Opening the technology to a wider set of developers via the web should benefit all parties. Developers would be able to integrate speech technologies into applications in ways that speech researchers have never considered, reaping benefits for users of their sites. Speech researchers, in turn, could collect large amounts of speech data from these applications, since they provide their speech technologies as services, and can log the usage data. Such data could be a great benefit to the research community, as large amounts of data are often necessary to devise and test new algorithms. Moreover, if methods to improve the error rates on such data were developed, they would directly benefit the users of the newly speech-enabled web applications. Better performing technology would, in turn, draw more users, and lead to even more data. Such a virtuous cycle would greatly accelerate the pace of innovation in multimodal interface design.

# Bibliography


[1] A. Aaron, S. Chen, P. Cohen, S. Dharanipragada, E. Eide, M. Franz, J-M Leroux, X. Luo, B. Maison, L. Mangu, T. Mathes, M. Novak, P. Olsen, M. Picheny, H. Printz, B. Ramabhadran, A. Sakrajda, G. Saon, B. Tydlitat, K. Visweswariah, and D. Yuk. Speech recognition for DARPA communicator. In *Proc. of ICASSP*, pages 489–492, 2001.

[2] Hua Ai, Antoine Raux, Dan Bohus, Maxine Eskenazi, and Diane Litman. Comparing spoken dialog corpora collected with recruited subjects versus real users. In *Proc. of SIGdial*, pages 124–131, 2007.

[3] Gregory Aist, James Allen, Ellen Campana, Carlos Gomez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus. Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods. In Ron Artstein and Laure Vieu, editors, *Proc. of the 11th Workshop on the Semantics and Pragmatics of Dialogue*, pages 149–154, 2007.

[4] Jonny Axelsson, Chris Cross, Jim Ferrans, Gerald McCobb, T. V. Raman, and Les Wilson. Mobile X+V 1.2. Technical report, 2005. http://www.voicexml.org/specs/multimodal/x+v/mobile/12/.

[5] Lauren Baptist and Stephanie Seneff. Genesis-II: A versatile system for language generation in conversational system applications. In *Proc. of ICSLP*, pages 271–274, 2000.

[6] Dan Bohus, Antoine Raux, Thomas K. Harris, Maxine Eskenazi, and Alexander I. Rudnicky. Olympus: an open-source framework for conversational spoken language interface research. In *Proc. of HLT-NAACL workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology*, pages 32–39, 2007.

[7] Dan Bohus and Alex Rudnicky. Integrating multiple knowledge sources for utterance-level confidence annotation in the CMU Communicator spoken dialog system. Technical Report CS-190, Carnegie Mellon University, 2002.

[8] Dan Bohus and Alexander I. Rudnicky. RavenClaw: Dialogue management using hierarchical task decomposition and an expectation agenda. In *Proc. of EUROSPEECH*, pages 697–600, 2003.

[9] Dan Bohus and Alexander I. Rudnicky. A principled approach for rejection threshold optimization in spoken dialog systems. In *Proc. of INTERSPEECH*, pages 2781–2784, 2005.

[10] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jennifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December 1992.

[11] Chih-yu Chao, Stephanie Seneff, and Chao Wang. An interactive interpretation game for learning chinese. In *Proc. of the Speech and Language Technology in Education (SLaTE) Workshop*, 2007.

[12] Lin Chase. Word and acoustic confidence annotation for large vocabulary speech recognition. In *Proc. of EUROSPEECH*, pages 815–818, 1997.

[13] Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

[14] Ananlada Chotimongkol and Alexander I. Rudnicky. N-best speech hypotheses reordering using linear regression. In *Proc. of EUROSPEECH*, pages 1829–1832, 2001.

[15] Grace Chung, Stephanie Seneff, and Chao Wang. Automatic induction of language model data for a spoken dialogue system. In *Proc. of SIGdial*, pages 55–64, 2005.

[16] Grace Chung, Stephanie Seneff, Chao Wang, and Lee Hetherington. A dynamic vocabulary spoken dialogue interface. In *Proc. of INTERSPEECH*, pages 327–330, 2004.

[17] Nils Dahlback, Arne Jonsson, and Lars Ahrenberg. Wizard of oz studies – why and how. In *Proc. of IUI*, pages 193–200, 1993.

[18] John Dowding, Jean Mark Gawron, Doug Appelt, John Bear, Lynn Cherny, Robert Moore, and Douglas Moran. GEMINI: a natural language system for spoken-language understanding. In *Proc. of ACL*, pages 54–61, 1993.

[19] Farzad Ehsani, Jared Bernstein, and Amir Najmi. An interactive dialog system for learning japanese. *Speech Communication*, 30(2-3):167 – 177, 2000.

[20] Ed Filisko and Stephanie Seneff. A context resolution server for the Galaxy conversational systems. In *Proc. of EUROSPEECH*, pages 197–200, 2003.

[21] Malte Gabsdil and Oliver Lemon. Combining acoustic and pragmatic features to predict recognition performance in spoken dialogue systems. In *Proc. of ACL*, pages 344–251, 2004.

[22] Lucian Galescu, Eric Ringger, and James Allen. Rapid language model development for new task domains. In *Proc. of LREC*, pages 807–812, 1998.

[23] Jesse James Garrett. Ajax: A new approach to web applications, 2005. Adaptive Path Essay, http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[24] Kallirroi Georgila, James Henderson, and Oliver Lemon. User simulation for spoken dialogue systems: Learning and evaluation. In *Proc. of INTERSPEECH-ICSLP*, pages 1065–1068, 2006.

[25] Laurence Gillick and Stephen Cox. Some statistical issues in the comparison of speech recognition algorithms. In *Proc. ICASSP*, pages 532–535, 1989.

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

[27] James R. Glass. Challenges for spoken dialogue systems. In *Proc. ASRU*, pages 307–316, 1999.

[28] Google AJAX APIs. http://code.google.com/apis/ajax/.

[29] Google maps API. http://code.google.com/apis/maps/.

[30] Google mobile app for iphone. http://www.google.com/mobile/apple/app.html.

[31] G. Gorrell, I. Lewin, and M. Rayner. Adding intelligent help to mixed initiative spoken dialogue systems. In *Proc. of ICSLP*, pages 2065–2068, 2002.

[32] Alexander Gruenstein. Response-based confidence annotation for spoken dialogue systems. In *Proc. SIGdial*, pages 11–20, 2008.

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

[34] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proc. of ICMI*, pages 141–148, 2008.

[35] Alexander Gruenstein, Jarrod Orszulak, Sean Liu, Shannon Roberts, Jeff Zabel, Bryan Reimer, Bruce Mehler, Stephanie Seneff, James Glass, and Joseph Coughlin. City browser: Developing a conversational automotive HMI. In *Proc. of CHI*, pages 4291–4296, 2009.

[36] Alexander Gruenstein and Stephanie Seneff. Context-sensitive language modeling for large sets of proper nouns in multimodal dialogue systems. In *Proc. of IEEE Spoken Language Technology Workshop*, pages 130–133, 2006.

[37] Alexander Gruenstein and Stephanie Seneff. Releasing a multimodal dialogue system into the wild: User support mechanisms. In *Proc. of SIGdial*, pages 111–119, 2007.

[38] Alexander Gruenstein, Stephanie Seneff, and Chao Wang. Scalable and portable web-based multimodal dialogue interaction with geographical databases. In *Proc. of INTERSPEECH*, pages 453–456, 2006.

[39] Alexander Gruenstein, Chao Wang, and Stephanie Seneff. Context-sensitive statistical language modeling. In *Proc. of INTERSPEECH*, pages 17–20, 2005.

[40] J. Gustafson, L. Bell, J. Beskow, J. Boye, R. Carlson, J. Edlund, B. Granström, D. House, and M. Wirén. AdApt a multimodal conversational dialogue system in an apartment domain". In *Proc. of ICSLP*, pages 134–137, 2000.

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

[42] H. Hastie, M. Johnston, and P. Ehlen. Context-sensitive Help for Multimodal Dialogue. In *Proc. of ICMI*, pages 93–98, 2002.

[43] Timothy J. Hazen, Stephanie Seneff, and Joseph Polifroni. Recognition confidence scoring and its use in speech understanding systems. *Computer Speech and Language*, 16:49–67, 2002.

[44] A. Hjalmarsson. Evaluating AdApt, a multi-modal conversational dialogue system using PARADISE. Master's thesis, KTH, Stockhom, Sweden, 2002.

[45] B. A. Hockey, O. Lemon, E. Campana, L. Hiatt, G. Aist, J. Hieronymus, A. Gruenstein, and J. Dowding. Targeted help for spoken dialogue systems: Intelligent feedback improves naive user's performance. In *Proc. EACL*, pages 134–137, 2003.

[46] Mikko Honkala and Mikko Pohja. Multimodal interaction with XForms. In *Proc. of the 6th International Conference on Web Engineering*, pages 201–208, 2006.

[47] Amanda Stent James Allen, George Ferguson. An architecture for more realistic conversational systems. In *Proc. of IUI*, pages 1–8, 2001.

[48] Java speech grammar format. http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/.

[49] M. Johnston, S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. Walker, S. Whittaker, and P. Maloor. MATCH: An architecture for multimodal dialogue systems. In *Proc. of ACL*, pages 376–383, 2002.

[50] Michael Johnston, Luis Fernando D'Haro, Michelle Levine, and Bernard Renger. A multimodal interface for access to content in the home. In *Proc. of ACL*, pages 376–383, 2007.

[51] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.

[52] Ed Kaiser, Alex Olwal, David McGee, Hroje Benko, Andrea Corradini, Xiaoguang Li, Phil Cohen, and Steven Feiner. Mutual disambiguation of 3D multimodal interaction in augmented and virtual reality. In *Proc. of ICMI*, pages 12 – 19, 2003.

[53] Kouichi Katsurada, Yusaku Nakamura, Hirobumi Yamada, and Tsuneo Nitta. XISL: a language for describing multimodal interaction scenarios. In *Proc. of ICMI*, pages 281 – 284, 2003.

[54] Ivana Kruijff-Korbayová, Tilman Becker, Nate Blaylock, Ciprian Gerstenberger, Michael Kaisser, Peter Poller, Verena Rieser, and Jan Schehl. The SAMMIE corpus of multimodal dialogues with an mp3 player. In *Proc. of LREC*, pages 2018–2023, 2006.

[55] Thomas Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, 1962.

[56] Raymond Lau, Giovanni Flammia, Christine Pao, and Victor Zue. WebGALAXY: beyond point and click – a conversational interface to a browser. *Computer Networks and ISDN Systems*, 29:1385–1393, 1997.

[57] Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. The WITAS multi-modal dialogue system I. In *Proc. of EUROSPEECH*, pages 1559–1562, 2000.

[58] Oliver Lemon and Alexander Gruenstein. Multithreaded context for robust conversational interfaces: context-sensitive speech recognition and interpretation of corrective fragments. *ACM Transactions on Computer-Human Interaction*, 11(3):241–267, 2004.

[59] Oliver Lemon, Alexander Gruenstein, and Stanley Peters. Collaborative activities and multi-tasking in dialogue systems. *Traitment automatique des langues*, 43(2):131–154, 2002. Special issue on dialogue.

[60] Diane J. Litman, Julia Hirschberg, and Marc Swerts. Predicting automatic speech recognition performance using prosodic cues. In *Proc. of NAACL*, pages 218 – 225, 2000.

[61] Gary M. Matthias. Incremental speech understanding in a multimodal web-based spoken dialogue system. M.eng., Massachusetts Institute of Technology, 2009. (in preparation).

[62] Ian McGraw and Stephanie Seneff. Speech-enabled card games for language learners. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence*, pages 778–783, 2008.

[63] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, June 1947.

[64] Media enclave. http://code.google.com/p/media-enclave/.

[65] Michael Niemann, Sarah George, and Ingrid Zukerman. Towards a probabilistic, multi-layered spoken language interpretation system. In *Proc. of 4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 8–15, 2005.

[66] Sharon Oviatt. Mutual disambiguation of recognition errors in a multimodel architecture. In *Proc. of CHI*, pages 576–583, 1999.

[67] David S. Pallett, William M. Fisher, and Jonathan G. Fiscus. Tools for the analysis of benchmark speech recognition tests. In *Proc. ICASSP*, pages 97–100, 1990.

[68] M. Phillips. Creating speech interfaces for mass market applications. In *Proc. of INTERSPEECH-ICSLP*, 2006. (Plenary Address).

[69] Quizlet. http://quizlet.com.

[70] Antoine Raux, Dan Bohus, Brian Langner, Alan Black, and Maxine Eskenazi. Doing research on a deployed spoken dialogue system: One year of Let's Go! experience. In *Proc. of INTERSPEECH-ICSLP*, pages 65–68, 2006.

[71] Verena Rieser and Oliver Lemon. Learning dialogue strategies for interactive database search. In *Proc. of INTERSPEECH*, pages 2689–2692, 2007.

[72] Rubén San-Segundo, Bryan Pellom, Wayne Ward, and José M. Pardo. Confidence measures for dialogue management in the CU Communicator System. In *Proc. of ICASSP*, pages 1237–1240, 2000.

[73] Johan Schalkwyk, I. Lee Hetherington, and Ezra Story. Speech recognition with dynamic grammars using finite-state transducers. In *Proc. of EUROSPEECH*, pages 1969–1972, 2003.

[74] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-II: A reference architecture for conversational system development. In *Proc. ICSLP*, pages 931–934, 1998.

[75] S. Seneff and J. Polifroni. Dialogue management in the mercury flight reservation system. In *Proceedings ANLP/NAACL Workshop on Conversational Systems*, pages 11–16, 2000.

[76] Stephanie Seneff. TINA: A natural language system for spoken language applications. *Computational Linguistics*, 18(1):61–86, 1992.

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

[78] Stephanie Seneff and Joseph Polifroni. A new restaurant guide conversational system: Issues in rapid prototyping for specialized domains. In *Proc. of ICSLP*, pages 665–668, 1996.

[79] Roger Argiles Solsona, Eric Fosler-Lussier, Hong-Kwang J. Kuo, Alexander Potamianos, and Imed Zitouni. Adaptive language models for spoken dialogue systems. In *Proc. of ICASSP*, pages 27–40, 2002.

[80] speak4it. http://speak4it.com.

[81] Amanda Stent, John Dowding, Jean Mark Gawron, Elizabeth Owen Bratt, and Robert Moore. The CommandTalk spoken dialogue system. In *Proc. of ACL*, pages 183 – 190, 1999.

[82] B. Suhm, B. Myers, and A. Waibel. Multimodal error correction for speech user interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(1):60–98, 2001.

[83] Stefanie Tomko, Thomas K. Harris, Arthur Toth, James Sanders, Alexander Rudnicky, and Roni Rosenfeld. Towards efficient human machine speech communication: The speech graffiti project. *ACM Transactions on Speech and Language Processing*, 2(1):2, 2005.

[84] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[85] Karthik Visweswariah and Harry Printz. Language models conditioned on dialogue state. In *Proc. of EUROSPEECH*, pages 251–254, 2001.

[86] VoiceXML forum, http://www.voicexml.org.

[87] Marilyn Walker, Jerry Wright, and Irene Langkilde. Using natural language processing and discourse features to identify understanding errors in a spoken dialogue system. In *Proc. ICML*, pages 1111–1118, 2000.

[88] Marilyn A. Walker, Alex Rudnicky, Rashmi Prasad, John Aberdeen, Elizabeth Owen Bratt, John Garofolo, Helen Hastie, Audrey Le, Bryan Pellom, Alex Potamianos, Rebecca Passonneau, Salim Roukos, Greg Sanders, Stephanie Seneff, and Dave Stallard. DARPA communicator: Cross-system results for the 2001 evaluation. In *Proc. of ICSLP*, pages 269–272, 2002.

[89] Kuansan Wang. SALT: A spoken language interface for web-based multimodal dialog systems. In *Proc. of ICSLP*, pages 2241–2244, 2002.

[90] Wayne Ward and Sunil Issar. Recent improvements in the CMU spoken language understanding system. In *Prcoc. of the ARPA Human Language Technology Workshop*, pages 213–216, 1994.

[91] F. Weng et al. CHAT: A conversational helper for automotive tasks. In *Proc. of INTERSPEECH*, pages 1061–1064, 2006.

[92] Frank Wessel, Andrea Baader, and Hermann Ney. A comparison of dialogue-state dependent language models. In *Proc. of ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pages 93–96, 1999.

[93] J.D. Williams and S. Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007.

[94] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[95] Wei Xu and Alex Rudnicky. Language modeling for dialog system. In *Proc. of ICSLP*, pages 118–121, 2000.

[96] Yushi Xu and Stephanie Seenff. Mandarin learning using speech and language technologies: A translation game in the travel domain. In *Proc. of ISCSLP*, pages 29–32, 2008.

[97] Yahoo onesearch. http://mobile.yahoo.com.

[98] Brandon Yoshimoto, Ian McGraw, and Stephanie Seneff. Rainbow rummy: A web-based game for vocabulary acquisition using computer-directed speech. (in preparation).

[99] Victor Zue, James Glass, David Goodine, Hong Leung, Michael Phillips, Joseph Polifroni, and Stephanie Seneff. The VOYAGER speech understanding system: Preliminary development and evaluation. In *Proc. of ICASSP*, pages 73–76, 1990.

[100] Victor Zue, James Glass, David Goodine, Hong Leung, Michael Phillips, Joseph Polifroni, and Stephanie Seneff. Integration of speech recognition and natural language processing in the MIT VOYAGER system. In *Proc. of ICASSP*, pages 713–716, 1991.

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