Multimodal Speech Interfaces for Map-based Applications

by

Sean Liu

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2010

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

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

Certified by .....
Alexander Gruenstein
Software Engineer, Google
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Multimodal Speech Interfaces for Map-based Applications

by

Sean Liu

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 2010

In partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis presents the development of multimodal speech interfaces for mobile and vehicle systems. Multimodal interfaces have been shown to increase input efficiency in comparison with their purely speech or text-based counterparts. To date, much of the existing work has focused on desktop or large tablet-sized devices. The advent of the smartphone and its ability to handle both speech and touch inputs in combination with a screen display has created a compelling opportunity for deploying multimodal systems on smaller-sized devices.

We introduce a multimodal user interface designed for mobile and vehicle devices, and system enhancements for a dynamically expandable point-of-interest database. The mobile system is evaluated using Amazon Mechanical Turk and the vehicle-based system is analyzed through in-lab usability studies. Our experiments show encouraging results for multimodal speech adoption.

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

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Scientist

Thesis Supervisor: Alexander Gruenstein
Title: Software Engineer, Google

Acknowledgments

First I'd like to thank Jim Glass. Jim has been a phenomenal advisor throughout my entire four years at MIT. He was willing to support me even as a freshman, and each year he has graciously allowed me to continue researching with the group. An advocate for me inside and outside the lab, Jim has given me the flexiblity and encouragement to pursue personally meaningful projects for an unforgettable experience at MIT.

It has also been a pleasure working with Stephanie Seneff. She has been gracious and patient in teaching me the details of the dialogue system and in helping me augment its features. From fixing cryptic seg faults to outdated grammar files, Stephanie has been invaluable in traversing the many roadblocks, bringing her smiles the entire way.

None of this work would have been possible without Alex Gruenstein, who has been an extraordinary mentor. I came to Alex as a freshman with few skills and experiences, but he was willing to take me on as an undergraduate researcher. He has devoted countless hours to craft manageable and meaningful research opportunities, and even after graduating has continued to offer his guidance. This thesis builds on his previous work, and I am grateful for inheriting his remarkable *City Browser* system.

A huge thank you goes to Ian McGraw who, along with Alex, developed WAMI and has been extremly helpful in making modifications to the *City Browser* system. Ian pioneered the group's work with Amazon Mechanical Turk and has been generous in sharing the tools that he developed along with his insights. Thanks also goes to Chiaying Lee, who was instrumental in getting the transcription engine running.

The work on vehicle systems is the result of Jeff Zabel's tireless support. He made it possible to obtain the vehicle system and explore interfaces on speech-based navigation systems. I am grateful for having had the opportunity to work with Jeff and other BMW associates as an intern at the BMW Technology Office, experiencing first-hand the efforts that go into developing autombile systems.

The vehicle interface usability data was the effort of collaborators at the Age Lab. Special thanks go to Jarrod Orszulak, Bryan Reimer, and Shannon Roberts.

The many coding obstacles would not have been crossed without Scott Cyphers, who delved into the code and helped me resolve numerous errors and library referencing challenges. Lee Hetherington was also instrumental in integrating dynamic updates for the recognizer system.

Thanks to the other friends, students, and staff of SLS including JingJing Liu, Ibrahim Badr, Stephen Shum, Yushi Xu, Yaodong Zhang, and Marcia Davidson for all their endless encouragement.

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advent of the smartphone has sparked the exploration of multimodal systems for mobile devices. These devices offer a rich combination of speech, graphics, and touch inputs, which together enable compelling user interfaces.

Researchers are exploring the use of speech recognition in combination with traditional input mechanisms to create multimodal interfaces. These interfaces have been reshaping traditional interaction paradigms, evolving from simple keyword searching to conversational dialogue. However, the vast majority of these multimodal applications are designed for desktop and tablet devices. Though speech is particularly promising for phone-sized mobile devices, the interfaces presented in related literature have difficulty scaling to the smaller screens and limited inputs of mobile and vehicle systems.

This thesis explores the development of multimodal systems specifically built for mobile devices. We present a redesigned MIT *City Browser*[1] interface sized both for an iPhone and for an automobile navigation system, with additional back-end augmentations for adding new interfaces and growing the point-of-interest database. We report the results of online user testing and in-vehicle usability experiments to show the potential for multimodal speech interfaces on mobile devices.

[1] A system previously designed in [9] which allows users to query for local points-of-interest using speech and manual inputs.

1.1 Challenges of Mobile Interface Design

Mobile devices present a number of user interface design challenges [4]. First, the smaller screen size heavily constrains the amount of displayable information [1]. Applications such as *City Browser*, which are built for desktop and tablet computers, have a large number of features for rich functionality. The challenge of building a mobile interface is retaining functionality without sacrificing usability.

Second, users expect mobile devices to be aware of physical context, since they take devices with them into a wide range of environments [27]. Users take for granted the ability to reference locations and objects relative to their position, such as requesting directions from "here" to "there". This challenge is an important consideration in redesigning desktop interfaces, which generally are geared toward stationary usage.

Third, phone-sized devices have limited input controls. Both the number and size of input keys are restricted by the device's physical dimensions. While desktop applications can expect input from keyboards and pointing devices, mobile smartphone systems are constrained by fewer controls.

Fourth, given the ubiquity of mobile devices, mobile users comprise a widespread population. Systems should be able to handle the diverse input queries which reflect the varying use cases, experiences, and personal backgrounds of users.

Finally, users multitask with their mobile devices, such as while traveling or in between conversations. This interaction method presents the challenge of task interruption and decreased user attention.

1.2 Multimodal Interfaces for Mobile Devices

Speech is compelling in its potential to address mobile interface challenges. Speech recognition capability enhances learnability and is unconstrained by the limited manual controls. For instance, in the *Speech Dasher* system developed in [31], speech was shown to nearly double the input rate when compared to manual entry. In map-based systems, multimodality showed task completion speed-ups of 10%, outperform-

Figure 1-1: The desktop and tablet version of *City Browser* (top-left) juxtaposed with the redesigned mobile (top-right) and vehicle (bottom) versions of the interface.

ing both pure manual and pure speech systems [24]. Combined with natural language understanding, speech pushes interfaces toward effortless conversational interaction.

Many of the existing multimodal interfaces are built for either desktop usage or larger tablet-sized mobile devices. For example, Figure 1-1 shows the original *City Browser* interface, which presents rich information content and controls for desktop web browsers. However, these interfaces are difficult to use when simply scaled down to phone-sized mobile devices, as the screen real estate cannot support the application content. In this work, we present re-designed interfaces of the *City Browser* system, which retain functionality while scaling to the small screen sizes.

1.3 Handling Widespread Populations

One step toward handling widespread user populations is ensuring database coverage for the multitude of potential user queries. The desktop version of *City Browser* utilizes a pre-generated database of points-of-interest (POIs) scraped from the Internet. This method presents the challenge of keeping the data up-to-date and expanding the POI coverage to new cities, as the web-scraping scripts must be constantly adapted to the changing source web pages. In this thesis, we explore an alternative method of accessing online content on-demand through search APIs. This offers the benefit of automatically expanding the database when necessary and leveraging the data kept current by online providers.

1.4 Efficient User Studies

Traditional methods of evaluating systems have focused on in-lab studies or individually reaching out to potential participants. These methods of soliciting participants have high associated time and labor costs. The advent of Amazon Mechanical Turk (AMT) has enabled new methods of rapidly collecting user feedback. Developers can post tasks through the AMT service, which distributes the requests to a large network of workers. We use AMT as a method for evaluating our interface and efficiently collecting feedback to iterate and improve the system.

1.5 Thesis Outline

This thesis is presented as follows: Chapter 2 reviews related work; Chapter 3 discusses the interface framework and development for mobile devices; Chapter 4 reviews system augmentation to enable on-demand content generation; Chapter 5 explains experimentation and evaluation of the mobile interface; Chapter 6 introduces our vehicle system; and Chapter 7 provides a conclusion and future research direction.

Chapter 2

Background

This chapter highlights related work on multimodal mobile interfaces. We then introduce the *City Browser* system architecture and previous work.

2.1 Related Multimodal Speech Interfaces

A number of multimodal mobile interfaces have been created by the community. TravelMan, developed at the University of Tampere, Finland is a multimodal mobile route guidance system [29]. It provides transport information services for buses, metro, and the tram. The system both presents route information and helps plan the journey. The interface displays its graphical information through text, though future aims are to incorporate map interaction.

AT&T's MATCH (Multimodal Access to City Help) system [16] is a mobile interface for the Fujitsu PDA, which is a type of tablet device. The system provides information regarding restaurants, local attractions, and travel. The interface allows for both speech input and pen-based gestures. Overall, the system closely resembles the *City Browser* system, although the accessed database is smaller in MATCH.

Similarly, Google has built a large-vocabulary automatic speech recognition system to interface with their search engine [7] and other services. Other companies, such as Microsoft, VLingo, and Dragon, have deployed applications which similarly showcase the growing trend of integrating speech recognition technologies into mobile

devices.

The MIT Spoken Language Systems group has also explored multimodal mobile interfaces in other domains beyond *City Browser*. For instance, one project built a music player interface, in which users could use speech to query and play their music library. Similarly, the group developed an entertainment system for playing TV and videoclips in which a mobile device served as the control [10].

One challenge of mobile applications that extends beyond the interface is designing for widespread populations [25]. Users come from diverse backgrounds and bring differing experiences and expectations. One piece of this challenge addressed in our work on *City Browser* is dynamically augmenting the database when confronted with user requests in metro regions not in the database. Similar work leveraging third-party content providers has been done by AT&T, in a prototype voice local search system [5]. This system uses automatic speech recognition for parsing user queries, and then extracts search terms to query an online YellowPages search service. *City Browser* follows suit, but builds the database on the fly, trading off between querying capabilities and requiring a database in advance.

2.2 The *City Browser* System

The work in this thesis builds on the *City Browser* system, developed in [9]. *City Browser* was one of the first widely-available web-based multimodal applications. The system allows users to query for local points-of-interest (POIs), such as restaurants, hotels, and museums. Users can interact with the system through natural spoken language in addition to mouse and pen gestures. The system served as a proof-of-concept for enriching web application interaction with speech capability. A screenshot of the interface is shown in Figure 2-1.

2.2.1 Web-Accessible Multimodal Interface Framework

City Browser is built with the Web-Accessible Multimodal Interfaces (WAMI) toolkit [9]. WAMI provides the skeleton to attach speech recognizer, synthesizer, and lan-

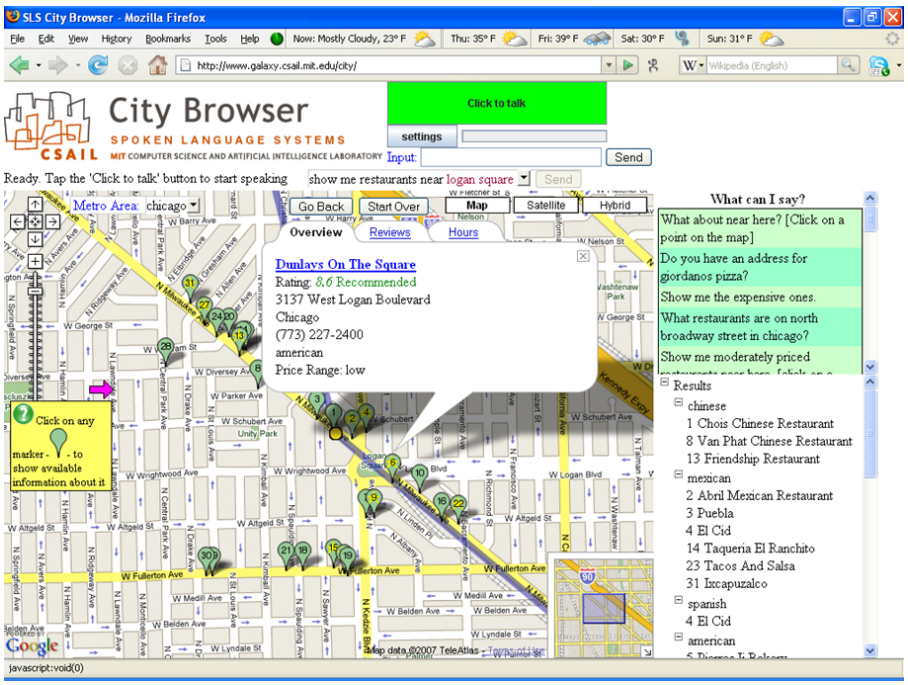

Figure 2-1: A screenshot of the *City Browser* interface. Here the user is asking for restaurants near Logan Square.

guage understanding servers. The MIT Spoken Language Systems group also hosts a WAMI server and recognizer, allowing developers to similarly integrate speech into their web applications through a Javascript API.

The *City Browser* architecture uses a customized WAMI architecture, shown in Figure 2-2. The shaded, gray boxes represent components provided by the default WAMI toolkit. The other components, such as the natural language processing server, were built specifically for the map-based city browsing domain.

2.2.2 Speech recognizer

The WAMI framework passes on spoken utterances from the client to the speech recognizer, SUMMIT [8], developed by the Spoken Language Systems Group (SLS). Of particular relevance to this project is SUMMIT's ability to dynamically swap out vocabulary sets for proper nouns [15]. For instance, while a user is browsing the city of Boston, the name recognition for local points-of-interest (POIs) is limited to the

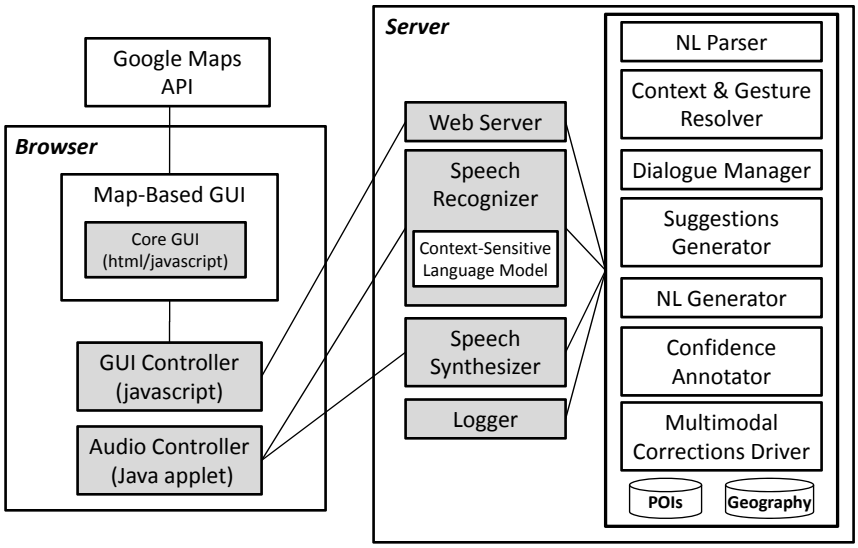

Figure 2-2: *City Browser*'s Architecture (reproduction of Figure 5 in [11]). Shaded boxes represent components provided by the WAMI toolkit.

Boston metro region. The SUMMIT recognizer outputs the text transcription of the utterance, with the proper nouns labeled by class.

2.2.3 Language Understanding Components

The speech recognizer output and GUI manipulations by the user are passed to the language understanding services. The natural language parser converts the text to an intermediate parsed state, and context resolution and dialogue management is performed to resolve semantic meaning. Once the user's intention is parsed, the system performs a database lookup. The relevant results are returned back to the user interface through the WAMI components.

2.2.4 Database Content

The POI database content for *City Browser* is scraped from the online city guide, Citysearch[1]. To add to the database requires updating the web-scraping scripts in order to reflect the Citysearch website updates. Our POI database was last updated in

[1] `http://www.citysearch.com`

2007, highlighting the challenges of keeping the content current. Chapter 4 highlights work on developing dynamically expandable databases, pulling content from online seach APIs.

2.2.5 Previous Evaluation

The *City Browser* desktop and tablet systems were previously evaluted by contacting participants individually and offering gift certificates for online user studies. This process was both challenging and time consuming, requiring reaching out to potential participants and then manually transcribing the interactions. In Chapter 5, we showcase the speed-ups in user evaluation from using Amazon Mechanical Turk.

2.3 Chapter Summary

This chapter highlighted a number of parallel efforts in building speech mobile interfaces, demonstrating the increasing prevalence of speech interaction mechanisms. The remainder of this thesis focuses on the work involved in redesigning and augmenting the *City Browser* system for mobile and vehicle devices, as well as its expansion to interface directly with web-based databases. We will also describe a user study based on Amazon Mechanical Turk, which allows for the widespread and rapid solicitation of user feedback.

Chapter 3

User Interface Design

One focus of this project has been developing a mobile interface for the *City Browser* system. The mobile interface builds off the *City Browser* desktop and tablet interfaces [9], shown in Figure 3-1. In this chapter we first present the creation of a framework for adding user interface designs to the system. Second, we discuss the mobile interface input mechanisms and challenges to designing for small screens. Finally, we give an overview of the mobile *City Browser* interface.

3.1 An Extensible Layout Framework

One step toward achieving wide-spread use of the *City Browser* system is supporting multiple devices: desktop, tablet, vehicle, and mobile. As additional devices with new screen sizes and input mechanisms are released, we hope to continue the effort of developing interfaces customized for these devices. To facilitate development, we created a framework for easily adding interfaces to the *City Browser* system. In this chapter, we will discuss using this framework for creating a mobile interface. The vehicle interface is discussed in Chapter 6.

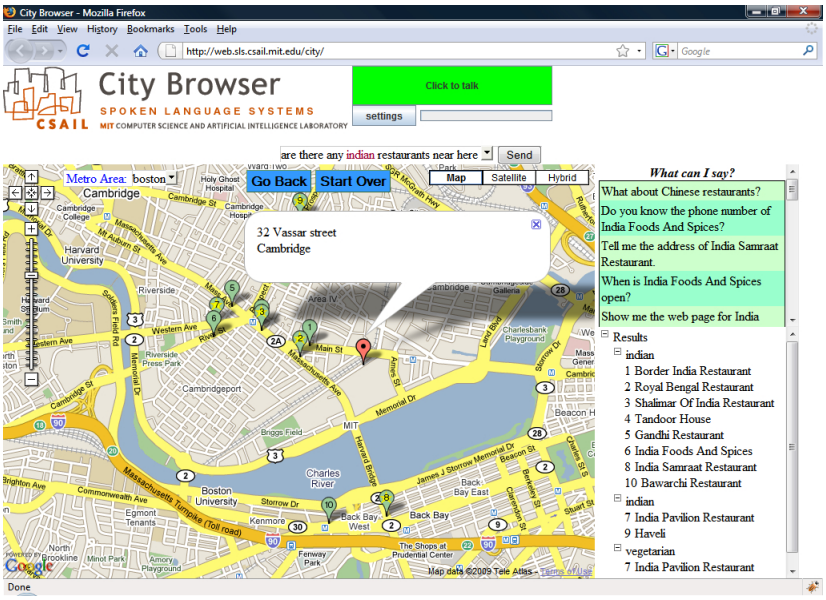

Figure 3-1: A screenshot of the original *City Browser* interface, developed in [9].

3.1.1 Separating Client and Server Code

The *City Browser* user interface implementation in [11] was closely linked with the server-side code. This was primarily due to server-side requirements for embedding audio recording applets in the interface. The latest version of the Web-Accessible Multimodal Interfaces (WAMI) toolkit[1], however, allows for a clean separation between the server-side speech handling and the client-side interface. The interface was re-implemented for this client/server separation, as well as upgraded in look and feel. Specifically, we expanded the screen real-estate dedicated to the map, condensed controls into the left column, and added details for each of the results list entries. The new desktop interface is shown in Figure 3-2. It should be noted that the desktop interface also doubles as the tablet interface. Given the large screen on tablet machines and the fine control of tablet pens, the desktop layout is naturally extendable to these devices.

[1] `http://wami.csail.mit.edu/`

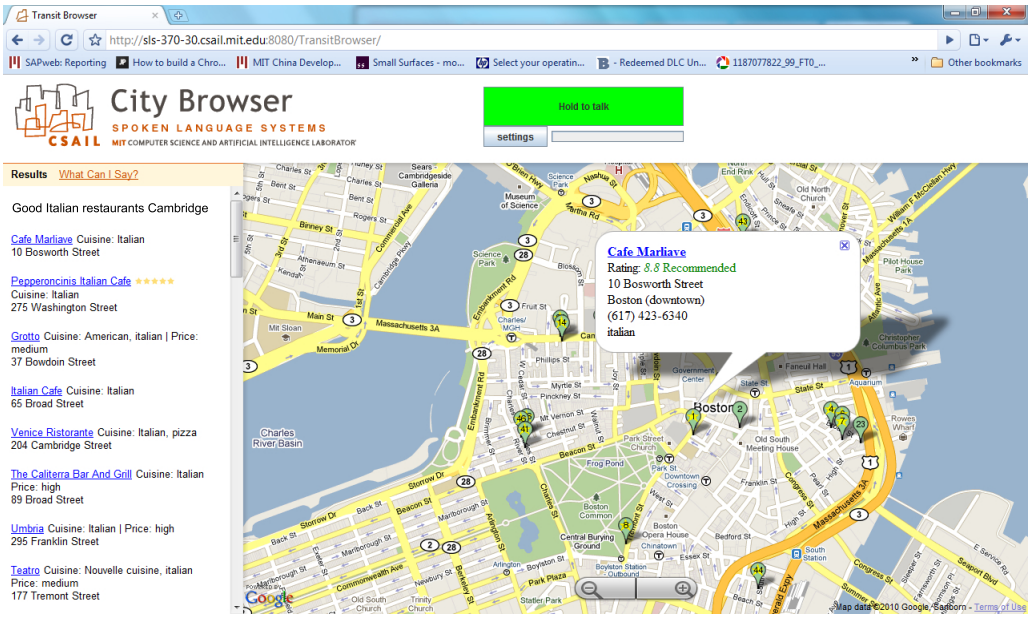

Figure 3-2: A screenshot of the latest *City Browser* interface.

3.1.2 Isolating Device-specific Implementations

We modularize the client-side code into generic methods and device-specific methods. For instance, code needed to parse point-of-interest (POI) results from the server is standard to all the interface implementations, whereas the layout controls are device-specific.

We allow the device-specific views to be interchanged by requiring that the view modules implement an interface. Each implementation is required to have the methods listed in Table 3.1, which provide controls for loading the layout, adding device-specific listeners, and handling server responses. Additionally, a Cascading Style Sheets (CSS) design file can be specified to completely redesign the look of the interface.

Which view to load is specified through a parameter passed in via the URL. The main system then loads the appropriate view's files based on this parameter. A new interface design can be added by extending the loading switch function and specifying new Javascript control and CSS design files.

Method	Purpose
`loadLayout`	Called when the device has finished loading the interface.
`handleDeviceReady`	Called when the WAMI audio applet has finished loading.
`showSuggestions`	Called when the system wants to display the suggestions.
`showResults`	Called when the system wants to display POI results.
`handleDeviceMessage`	Called when the server returns a message.

Table 3.1: Device-Specific View Methods

3.2 Multimodal Input Mechanisms

In mobile devices, the primary input mechanism for manipulating the graphical user interface is touch. Touch imposes several constraints: buttons should be at least 22mm in width [19] and the interface should not require too much panning and zooming to view [14]. To take advantage of the multi-touch functionality of the map, we defer to the Google Maps API[2], which allows multi-touch zoom and panning.

3.2.1 WAMI iPhone Application

For a user to load *City Browser* on their iPhone, they must view the site through the WAMI iPhone application. The WAMI iPhone application[3] handles the speech recording on the mobile device, which is otherwise unsupported in the default iPhone browser. The application is essentially a web browser with added speech recording functionality. When a WAMI web-application is loaded by the WAMI browser, the web-application's reference calls to the WAMI API also load a "Hold to Talk" button at the bottom of the application.

3.3 Graphical Layout for Mobile Devices

Since the *City Browser* system was originally designed for desktop and tablet devices, it had no corresponding mobile interface. Loading the system in a mobile device's web browser simply presented a scaled version of the page, as shown in Figure 3-3.

[2] `http://code.google.com/apis/maps/`
[3] `http://wami.csail.mit.edu/mobile.php`

Figure 3-3: A screenshot of *City Browser* scaled to a mobile device screen.

In designing a mobile user interface, we followed the W3's Mobile Web Best Practices[4]. To fit all the *City Browser* functionality into the screen dimensions (320px by 370px[5]), we divide the conceptual pieces of the interface into widgets: application navigation, map control, recognition results, and help suggestions, as shown in Figure 3-4. Much of this organization was based on our previous work on vehicle-based interfaces [12]. A persistent menu of buttons at the top of the screen serves as the application navigation, allowing users to toggle between the other widgets.

When the user first accesses the application, they are greeted with the map-pane in focus, as shown in the middle screenshot of Figure 3-4. After a query is spoken to the system, the interface automatically toggles to the results list. If the system fails to understand multiple utterances in succession, then the suggestions screen is shown. All other toggling between panes is controlled manually by the user.

As the user speaks, a streaming recognition system returns recognized concepts back to the user, as shown in Figure 3-5. This provides immediate feedback, generating the perception of a faster system and giving the users immediate feedback as to

[4] `http://www.w3.org/TR/mobile-bp/`

[5] Visible display area within the WAMI mobile iPhone app

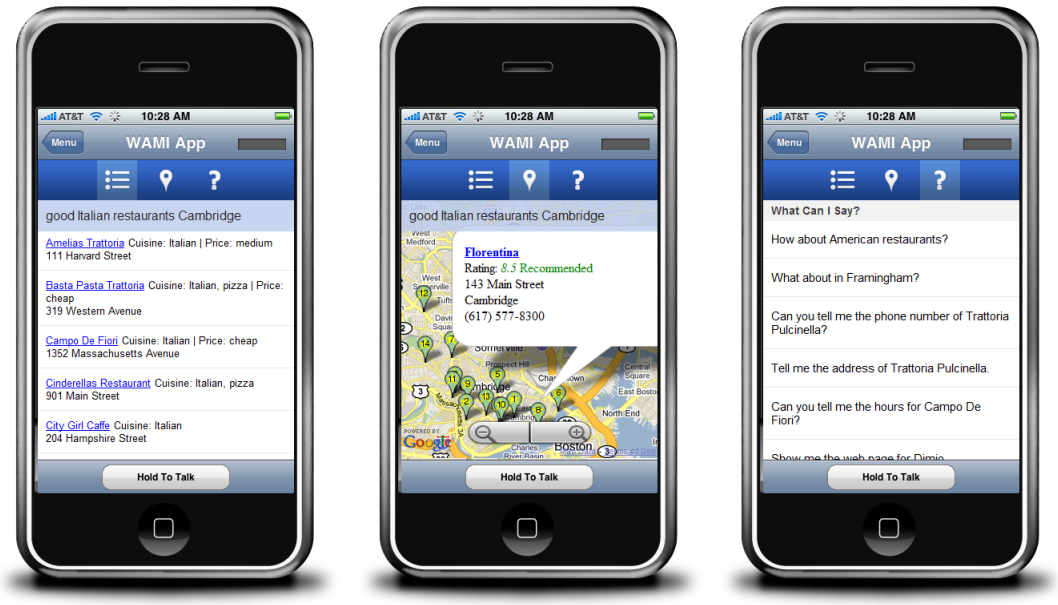

Figure 3-4: Screenshots of the *City Browser* mobile system. The left-most screenshot shows the results list after asking for "Italian restaurants in Cambridge". The middle screenshot shows the map results. The right-most screenshot shows the help suggestions with sample queries.

whether they are being understood. Once the entire utterance is processed, however, we update and reorder the output's key-concepts. This interface is modeled after the success of *FlightBrowser* [21], in which concept slots were displayed to the user. In this project, however, we show only the concept values for space-considerations. Specifically, we show the domain-specific values in this order:

1. Price range
2. Recommendation
3. Cuisine
4. Topic
5. Location
6. Remaining concept values

Figure 3-5: Streaming recognition results. As the user speaks to the system, the recognition results (shown in blue) are streamed immediately to the screen. Here the user is speaking, "Show me cheap Italian restaurants in Cambridge that are high quality."

3.4 Future Work

Future work should focus on two particular areas. The first is exploring better methods of organizing recognition results and allowing concept correction by the user. Promising correction methods have been implemented in [30], and would be applicable to *City Browser*. The second area is expanding the number of device interfaces, such as for the recently released iPad device. Although the desktop interface would likely fit the screen dimensions well, the iPad's focus on touch inputs may require adapting the *City Browser* controls.

Chapter 4

Mechanism for On-Demand Dynamic Content

This chapter presents a mechanism for dynamically generating point-of-interest content, as deployed in the *City Browser* system. Generating content on-demand contrasts with pre-generating read-only databases. In the pre-generated database approach, having fixed data provides the benefit of offline training for language models, but poses the challenge of requiring developers to preemptively identify and create databases needed for responding to all potential user queries. Using a database divided by metro regions and point-of-interest types, the number of data sets needed for coverage of just the United States quickly grows out of hand.

We introduce a method for augmenting the pre-generated static database approach on-demand. As users request content for metro regions unavailable in the database, the system pulls information from online web resources to dynamically grow the database and fill in the missing information. This approach is analogous to other systems which pull information from content providers, such as the MIT *Flight Browser* system which links to ITA [26], or the AT&T Voice Search system which connects to `http://www.yellowpages.com` [5]. Using the third-party content provider and search infrastructure provides the benefits of up-to-date databases and well-addressed scalability issues [5].

The rest of this chapter proceeds as follows: first, we present the format of the on-

demand database entries; second, we discuss augmentations to the system architecture for on-demand database generation; finally, we describe interface adjustments to allow for the wide range of user inputs.

4.1 Dynamic POI Content Generation

A challenge to the scalability of the *City Browser* system is the database content coverage. As a widely-deployed web application, *City Browser* receives queries from users requesting points-of-interest (POIs) from a vast number of metro regions. Anticipating the locations, building the databases, and maintaining up-to-date content is time-consuming for a large-scale system.

Here we describe the method for generating the database content on-demand, with the goal of interchanging the static pre-generated with the dynamic on-demand sources transparently to the system. The primary module of interest is the database lookup module. The database lookup module takes as input the requested metro region and POI type. In the previous version of the system, if no database matching the input parameters was available, the system simply returned an empty set. Our augmented version expands the database on-the-fly with information pulled from other content providers.

4.1.1 Online Content Using the Yelp API

We use Yelp[1] as our content provider for the dynamic database. Yelp, best known for its restaurant information and reviews, has POI content for food, health, and local attractions. Their database is exposed through the Yelp Search API[2]. Developers can provide the API with a set of search terms, and Yelp returns up to 20 POI results.

The *City Browser* database lookup function was modified such that, when no database content is available, the system requests data from Yelp. For the majority of cases, the metro region and POI type are used as the search parameters. In a

[1] `http://www.yelp.com`
[2] `http://www.yelp.com/developers/documentation/search\_api`

```
{"message": {"text": "OK", "code": 0, "version": "1.1.1"}, "businesses":
   [{"name": "Charles Hotel",
     "id": "PmnIrliSIYc_0L5fWUhbKA",
     "address1": "1 Bennett St",
     "city": "Cambridge",
     "state": "MA",
     "zip": "02138",
     "country": "USA",
     "longitude": -71.12182,
     "latitude": 42.372297000000003,
     "phone": "6178641200",
     "state_code": "MA",
     "categories":[{"category_filter": "hotels",
                    "name": "Hotels"}],
     "neighborhoods": [{"name": "Harvard Square"}],
     "url": "http://www.yelp.com/biz/charles-hotel-cambridge",
     "avg_rating": 4.5},
     ...
   ]}
```


Figure 4-1: An example JSON result returned by querying the Yelp API for `hotels in Cambridge`.

few select POI types such as restaurants, in which more than 20 results would be beneficial, the search parameters are further refined (e.g. by cuisine type) such that multiple Yelp queries may be made in succession. All requests are made on-demand, and the full process of querying Yelp and generating the database takes approximately one second. Yelp data are returned in the JSON string format. An example result to a query for `hotels in Cambridge` is shown in Figure 4-1.

Since one goal in augmenting the system with on-demand functionality is to minimize impact to existing methods, the JSON object is adapted to mimic the static pre-generated database files. The JSON object returned from Yelp is converted to a frame with an identical format to the database frames on disk. An example of such a frame is shown in Figure 4-2. Since the database lookup function retrieves results in identical formats independently of the source, the change is transparent to a majority of the system, with the exception of the recognizer vocabulary, as discussed in section 4.2. The search results from Yelp are cached on disk to improve lookup time on future requests, thus dynamically growing the database.

```
{c hotels
   :nfound 20
   :orphan_metro_region "cambridge"
   :values ( {q hotel
                :city "cambridge"
                :citysearch_id 1
                :latitude 42.372297
                :longitude -71.12182
                :name "charles hotel"
                :nicknames ( "charles hotel"
                             "charles" )
                :phone "(617) 864-1200"
                :rating 90
                :recommendation "highly recommended"
                :rest_id 1
                :source "www.yelp.com"
                :state "ma"
                :street "bennett street"
                :streetnum "1" }
   ...
) }
```

Figure 4-2: An example database frame created from the Yelp JSON result for `hotels in Cambridge`.

One important note is that the geographical information data (streets, neighborhoods, and cities) is a special case of querying information from Yelp. The geographical data is needed when user's ask for POIs located in a specific city or on a specific street. Yelp does not expose the geographical data in their Search API. Therefore we infer the city, street, and neighborhood information from the POIs that were generated from querying Yelp. First we generate a list of all unique city names and streets seen in the POI database, also recording the association between streets and their cities. To obtain the latitude and longitude information for each city, we randomly select a POI located in the given city, and borrow the POI's latitude and longitude. A better alternative for future implementations would be averaging the latitudes and longitudes of all POIs in the city, or looking up the values in a separate city database, such as the U.S. Census Bureau's TIGER database[3].

[3] `http://www.census.gov/geo/www/tiger/`

4.2 On-the-fly Recognizer Vocabulary Updates

In the existing system, whenever the user switches to a new metro region, the dialogue control rules dispatch messages to the recognizer to update the dynamic vocabulary in the system. For instance, switching from the Boston to the San Fransisco metro region triggers the system to load finite state transducers (FSTs) from the San Fransisco FST database. These FSTs include the names and pronunciations of restaurants, museums, hotels, and geographical data, such that the recognizer can appropriately identify these spoken words.

We can distinguish two approaches to updating the vocabulary in the recognizer. The first approach involves loading static FST files, which have been pre-generated and stored on disk. The second approach involves passing a dynamically generated list of elements to the recognizer, to be automatically inserted into the dynamic class FST. The first case is well suited for data which has been preprocesssed and generated, such as pre-scraped data from the Internet.

We decide to use the second approach for the Yelp data. We utilize a recognizer capability, which enables us to pass in a list of new POIs to be dynamically added automatically to the recognizer's vocabulary. The recognizer automatically generates pronunciation baseforms through a letter-to-sound system for any words not present in its static lexicon.

The database lookup function flags to the system whether the vocabulary updates should come from the static FST files or the dynamically generated lists. In the latter case, the data are either queried from Yelp or read from a disk cache. This mechanism is implemented by adding several dialogue control rules. The dialogue processing flow is governed by these dialogue control rules, which dictate under which conditions to load the recognizer FSTs. We create twin rules: one handles the loading of the static recognizer FST files, and the other one handles the passing of a dynamic list of POIs. If data is dynamically pulled from Yelp, we trigger the second rule, otherwise triggering the first.

Currently, all the requests to Yelp happen immediately when a user switches to

a new metro region. The advantage of this approach is that we can immediately load the recognizer's vocabulary with the names of all POIs. This corresponds to the creation of a single FST of all proper nouns, incorporated immediately into the general background model. Users can ask for restaurants by name without having to first cue the system. The disadvantage is the several seconds of delay when the system switches to a new metro region that has no cached data. The alternative, "lazy" approach, is to fetch data only when requested. This would preclude a user from immediately asking for a specific POI by name, but would spread the timing delays across multiple interactions with the user. Additionally, this approach to dynamically load vocabularies has been shown to increase recognition accuracy [12]. The database lookup function is able to support both approaches, but the dialogue rules and grammar are currently written for the first approach.

4.3 Dynamic Inputs in the User Interface

The dynamic content generation enables flexibility in metro region selection. In the previous *City Browser* system, users were restricted to a pre-defined set of cities. This has now been upgraded to a text box in which *any* city can now be input to the system, as shown in Figure 4-3. In the ideal case, we would even allow the user to specify a metro region change via speech, as done in the multi-pass, dynamic-vocabulary system built in [15]. For now this functionality is unsupported, but is a potential area of expansion.

One final note is expanding the natural language system to also allow any POI type. Although the database functionality is already written to handle any arbitrary POI type, the language understanding systems have not been trained to understand these POI words (e.g. "pharmacies", "parking lots", etc.). Current additional work is being done to augment the grammars to handle a wider range of POI types.

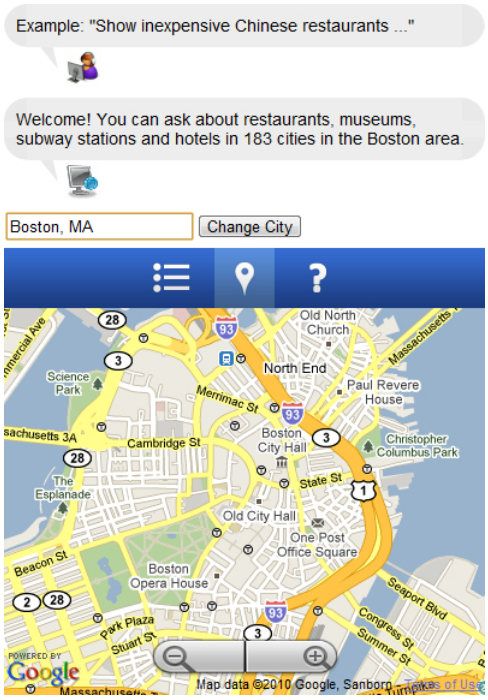

Figure 4-3: The user interface showcasing the ability to arbitrarily enter a metro region to the system.

4.4 Future Work

There are three main areas for future work. The first is implementing a more robust caching scheme. Currently the data pulled from Yelp is cached in text files written to disk. Using a SQL database would be more robust and able to handle potential concurrency challenges.

The second area for future work is delaying POI requests from Yelp to when users specifically request the data, as opposed to the implemented approach of immediately requesting data when the user switches metro regions. This would explore the trade-offs between timing delays and the ability to ask for POIs immediately by name. The optimal approach will likely be a combination of pre-emptive and on-demand querying.

The third area is expanding the input methods to allow for metro region changes using speech. This work would explore incorporating all metro regions into the static vocabulary background model.

Chapter 5

Amazon Mechanical Turk Experiments

In this chapter we discuss two *City Browser* experiments using Amazon Mechanical Turk (AMT). In the first experiment, we collect and analyze user interactions with a live version of our system deployed through AMT. AMT enables us to reach a large audience at a low cost, contrasting the traditional method of seeking out test users individually.

Second, we use AMT to rapidly compile sample text inputs to the system, which are then used as training data for the recognizer's language model. In three days, we are able to collect over 12,000 typed sample queries to our system. We then tag all proper nouns in the sample queries in order for the data to be usable as training sets for our system language models. In this work we resort to manual and pattern-based tagging methods, and therefore use only a small 1,000 sentence subset. Future work would explore auomated tagging methods to incorporate all 12,000 sample queries in the training data. To evaluate the retrained language model, we run the recognizer on the transcribed utterances obtained from our first experiment, which collected live interactions. Even with only the 1,000 additional sentences, we show that the additional training data provides a 9% relative reduction in WER, bringing it to 29.2%.

5.1 Background on Amazon Mechanical Turk

Amazon Mechanical Turk[1] (AMT) provides a potential solution for rapid collection of usability and training data. The service allows programmers to post Human Intelligence Tasks (HITs) online, tasks to be solved by workers for small monetary rewards generally ranging from $0.01 to $0.10 per task. The workers come from all around the world, though we restrict the HIT availability to workers in the United States in order to focus on native American English speakers.

AMT is increasingly used by the research community for its crowdsourcing ability. One such example is the use of AMT for user studies and feedback solicitation [17]. As long as tasks are verifiable and concrete, AMT has been shown to provide high quality user studies at a cheaper and faster rate than traditional in-lab studies. Another example is the speech community's use of AMT to process spoken data. AMT HITs have been crafted to have workers transcribe spoken documents both quickly and accurately at a fraction of the cost of professional transcribers [22, 23].

Similarly relevant is the use of AMT to efficiently construct large corpora, especially important for natural language and speech systems as they rely on the training data for performance [2]. Corpora have been shown to be built at remarkable speeds using AMT. A corpus of over 100,000 read utterances was constructed in under three days in the address recognition domain [22]. Similarly, researchers at Nokia were able to generate enough content from AMT to add new languages to their speech recognizer [18].

5.2 Collecting User Interactions with *City Browser*

Although the ideal usability study for our mobile version of *City Browser* would be in-person evaluations on a mobile device, we made a first approximation study by deploying the system online as a web application. We embedded our online mobile interface into an AMT Human Intelligence Task (HIT), requesting that workers interact with *City Browser* to complete tasks.

[1] `http://www.mturk.com`

You're in New York shopping on Canal Street, when your friend calls and tells you to meet her at the Gershwin Theatre. Find out how to get there using public transportation.

You are helping friends find their way to the Cheer's Bar in the Boston Commons. Help them find their way from State Street to the Bar.

You want to take your little sister to the water park and hotel about 1 hour north of Vancouver, WA but don't know the name. Find out the resort's name and phone number.

You are driving through Northern California to meet a long lost family member at the historic Cascade theater located in Redding, California, but your not sure what freeway exit to take or where the theater is located. Find the theater address and freeway exit information.

Figure 5-1: Four example scenarios generated by AMT workers.

5.2.1 Generating Task Scenarios

The first step in our usability study was generating the task scenarios. One innovation of [22] was the use of AMT not only to evaluate the system, but also to generate the scenarios used in the evaluation. We collected a total of 69 scenarios from users, of which 36 were usable. Four example scenarios are shown in Figure 5.2.1. We modify these scenarios focus on the Boston region, since our data for the Boston-area is more robust and complete. Future work would test the system across additional metro regions.

5.2.2 Embedding Into the Mechanical Turk Interface

We then made a version of *City Browser* customized for AMT. This version mimicked the same dimensions and look and feel of the mobile version. However, we made a small number of changes, such as replacing audio synthesis with text system responses, presenting incremental recognition results, and constantly displaying the task description. These changes were guided by the AMT user testing framework developed in [22]. A screenshot of the AMT Human Intelligence Task (HIT) is shown in Figure 5-2.

The goal of the HIT was to elicit user interaction with the system. In total we gathered 579 utterances from 124 tasks and 35 users. A sample interaction is shown in Figure 5-3. We offered AMT workers $0.10 per completed task.

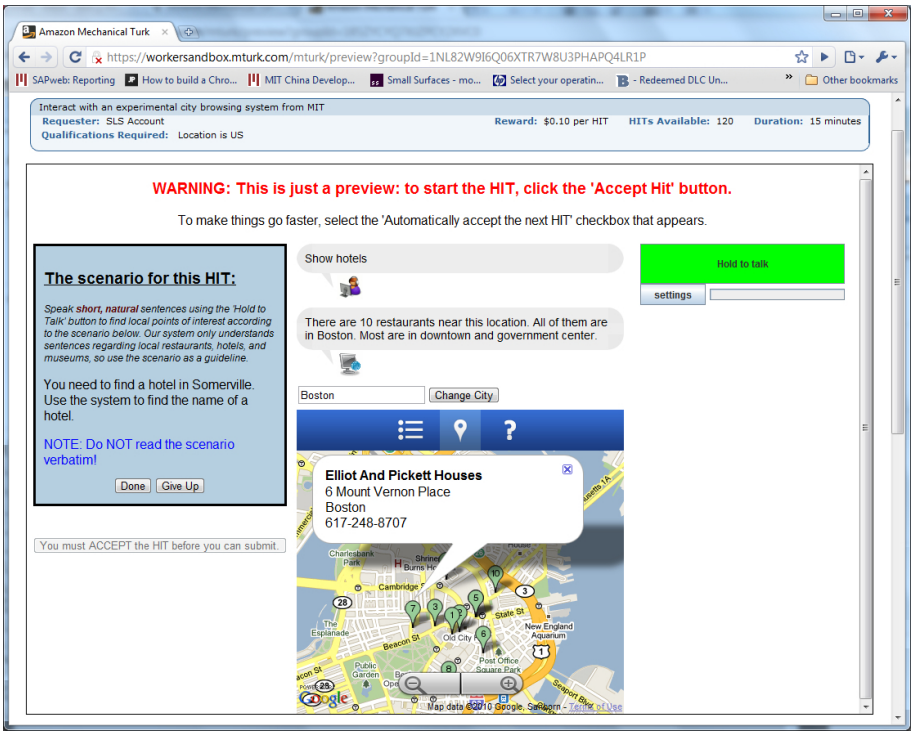

Figure 5-2: A screenshot of the Mechanical Turk interface of *City Browser*.

5.2.3 Transcribing the Audio

In order to run experiments on the data, we transcribed the spoken interactions. Again we utilized AMT, shown as an effective transcription method [20, 22, 23]. We created a transcription HIT which presented workers with audio files and our speech recognizer's top transcription hypotheses. The HIT then asked users to correct the recognition output. A screenshot of this HIT is shown in Figure 5-4. We paid workers \$0.01 per transcription, taking approximately 1 day to transcribe all 579 utterances. The HIT had three workers transcribing each utterance. We then ran an alignment script to merge the transcriptions based on a voting scheme using transcription similarities. Manual examination of 50 of the utterances manually, there is approximately a 90% sentence accuracy rate.

U_1:	Find a hotel near Harvard University.
S_2:	There are 10 hotels near University in Harvard. All of them are in Cambridge. Most are in Harvard Square. [*results shown on map*]
U_3:	[*user clicks results and scans listing*] Find a hotel in Harvard Square.
S_4:	There are 8 hotels near Harvard Square. They are concentrated on Cambridge Street and Bennett Street. [*results shown on map*]
U_5:	Find a hotel on Cambridge Street.
S_6:	There are 2 hotels near Harvard Square on Cambridge Street. The A Friendly Inn and the A Bed and Breakfast in Cambridge. [*results highlighted on map*]
U_7:	What is the address for A Friendly Inn on Cambridge Street?
S_8:	A Friendly Inn is located at 1673 Cambridge Street in Harvard Square in Cambridge. [*A Friendly Inn is focused on map*]

Figure 5-3: A sample interaction between a user and the *City Browser* system. U indicates the user, and S indicates the system.

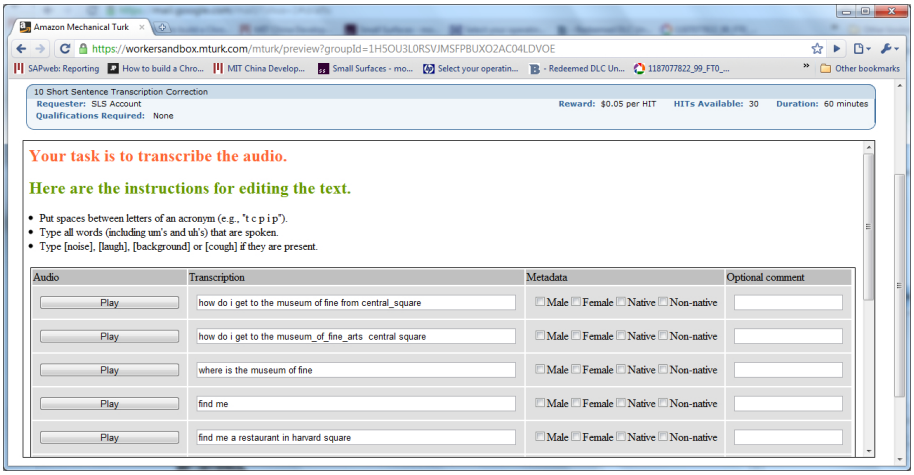

Figure 5-4: The Mechanical Turk HIT for transcribing audio.

5.2.4 Usage Statistics

Here we compute some general statistics on the user data. The average utterance length was 6.92 words per utterance, and there were an average of 35.4 words per session. In particular, we hypothesized that the greater the number of interactions for the task, likely the shorter the utterance length. However, as shown in Figure 5-5, there is no strong correlation (correlation coeffcient -0.129). Likely the differences are simply caused by some users being more verbose than others. Next we also note the effect of WER on the system response. Figure 5-6 shows that, as WER increases, the percentage of correct responses rapidly declines.

The recorded interactions highlighted a number of areas for interface improvement,

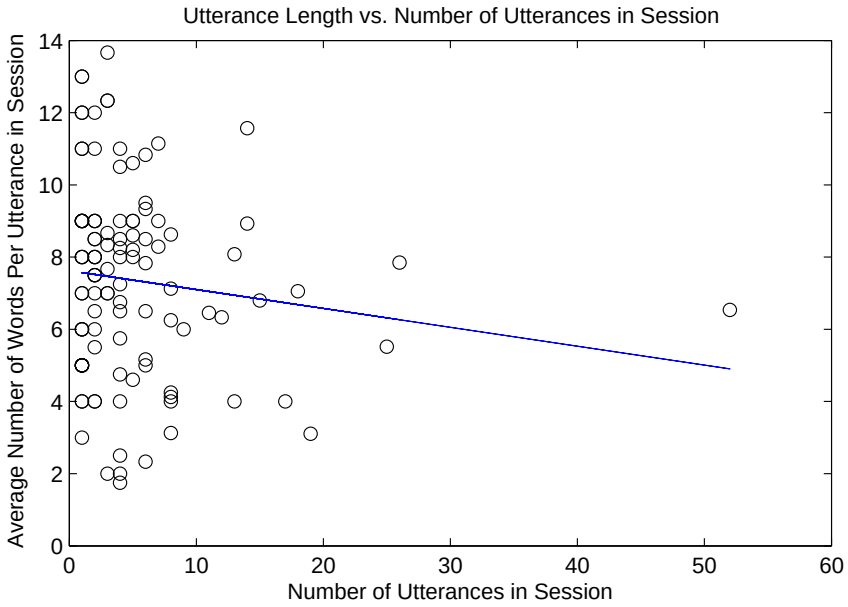

Figure 5-5: The average utterance length in a session versus the number of utterances in the session. Overlayed is the line of best fit.

to be explored in future work:

- Improved recognition of museums. Museums in particular seemed to suffer from the greatest number of recognition errors, likely due to problems in the system training data.

- Typed-input correction mechanisms. Some POIs were impossible to request, since the database simply did not have the information. Supporting typed-inputs appears to be an important needed feature as a fall-back for information entry.

- Single concept correction. Often times utterances would be correct with the exception of a single concept, such as the confusion between *Museum of Fine Arts* and *Museum of Science*. However, to correct the error, users had to restate the entire utterance, since our mobile interface lacked a correction scheme. Future work would explore the best way to integrate this correction feature into the mobile interface.

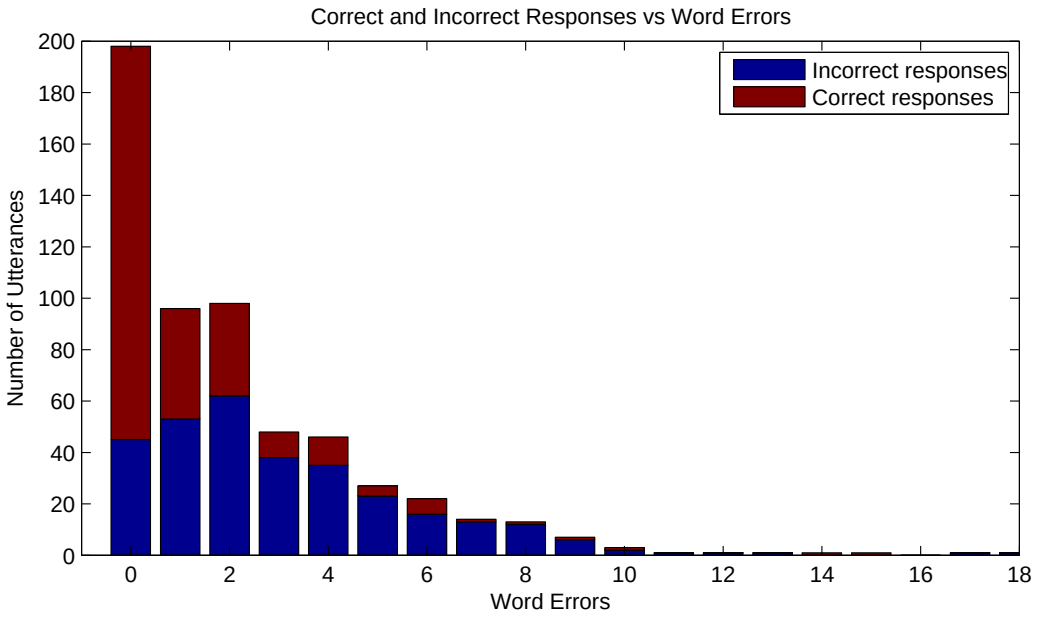

Figure 5-6: The number of correct and incorrect responses by the number of word errors in the utterance.

5.3 Retraining Language Models with Turk Data

This section discusses efforts to improve the speech recognizer language model used by the MIT *City Browser* system. We focus on the collection of training data by eliciting typed inputs from users. The primary difficulty is that high quality data is hard to come by, often requiring both long collection times and intensive annotation labor. Our collection process uses Amazon Mechanical Turk (AMT) to amass typed input queries, which can be obtained faster and cheaper than recording and transcribing interactions. These typed queries are obtained by asking AMT users what they would input to a hypothetical city information system. To evaluate our improved language model, we use a test corpus consisting of spoken utterances from AMT users interacting with a live version of *City Browser*. We show that the improved language model, trained with the addition of our typed sample queries, realized a 9% relative reduction in WER from 32.1% to 29.2%. Finally, by running an oracle experiment training the language models with the transcribed test data, we show that the lower-bound on language model performance is 21%.

Language models for speech recognizers require a substantial amount of data. The

current recognizer used by the MIT *City Browser* system uses more than 20,000 sample sentences. Amassing high quality data is traditionally a laborious task involving creating a live system to record spoken user interactions, transcribing the recorded audio, and finally annotating proper nouns. Amazon Mechanical Turk (AMT) offers a means of collecting data more efficiently than this traditional method. We use the service to obtain human-generated sample text queries by presenting AMT workers with a hypothetical speech system and asking them for example text inputs. This data is then added to our baseline system's data to retrain the language model.

It is important to note that for the example text inputs to be used as training data for the recognizer and natural language components, all proper nouns in the text must be tagged with their POI class. In this work, we use manual and pattern-matching methods, which limit us to a small 1,000 sentence subset of the full 12,000 sentences. Future work would explore automated tagging algorithms to incorporate the full 12,000 sentences.

To test the language model, we use the transcribed utterances from our usability experiments. Our system's baseline recognizer has a recognition word error rate (WER) of 32.1%. A majority of the baseline system's training data consists of sentences automatically generated from template files [13]. Our hypothesis is that human-generated sample inputs would improve the language model accuracy, driving WER toward the oracle experiment's lower bound of 21%. We show that adding in the typed sample inputs from AMT reduced the WER, bringing it down to 29.2%.

5.3.1 Class *n*-gram Language Model

The language model assigns a probability $P(W)$ to an input word sequence $W = w_1, w_2, ..., w_n$. The *City Browser* reocgnizer, SUMMIT, utilizes a class n-gram language model, where all words are mapped to equivalent classes

$$W = \{w_1, ..., w_n\} \rightarrow \{c_1, ..., c_n\} \tag{5.1}$$

in which the classes are each of the POI types, such as `RESTAURANTS`, `HOTELS`, and

MUSEUMS. All proper nouns in the vocabulary are then mapped to one of these classes. The formula for the language model with these equivalent classes then becomes

$$P(W) \approx \prod_{i=1}^{n} P(w_i|c_i)P(c_i|c_{i-(N-1)}, ..., c_{i-1}) \tag{5.2}$$

where the probability of the word within the class, $P(w_i|c_i)$ is the maximum likelihood estimate

$$P(w_i|c_i) = \frac{C(w_i)}{C(c_i)} \tag{5.3}$$

and $C(w_i)$ is the number of times w_i is observed in the training corpus, and $C(c_i)$ is the number of times a class c_i is observed.

The current recognizer language model is trained on a combination of 2,163 transcribed utterances and 20,000 sample sentences automatically generated from template files defined by system developers. The templates specify recursive context-free grammar rules, and word combinations are permuted to fill the grammatical structures and generate sample sentences.

5.3.2 Collecting Sample Queries

To collect our training data, we created an AMT HIT which requested workers to provide us with typed sample queries. Our HIT described a hypothetical system which understands queries about local points-of-interest (POIs), and presented screenshots of the mobile *City Browser* interface. Additionally, we gave the workers five reference sentences, representative of the queries understood by the system. Note that these sentences were the same for all workers, and future work should explore randomizing the sentences presented. We then requested that workers provide us with typed sentences of what they would ask such a system. A screenshot of this HIT is shown in Figure 5-7.

Initially we paid workers $0.03 per sentence provided, for a total of 7800 sample inputs. We then experimented with decreasing the pay rate, collecting an additional 4200 inputs at $0.025 per sentence. We found that the rate of collection was essentially

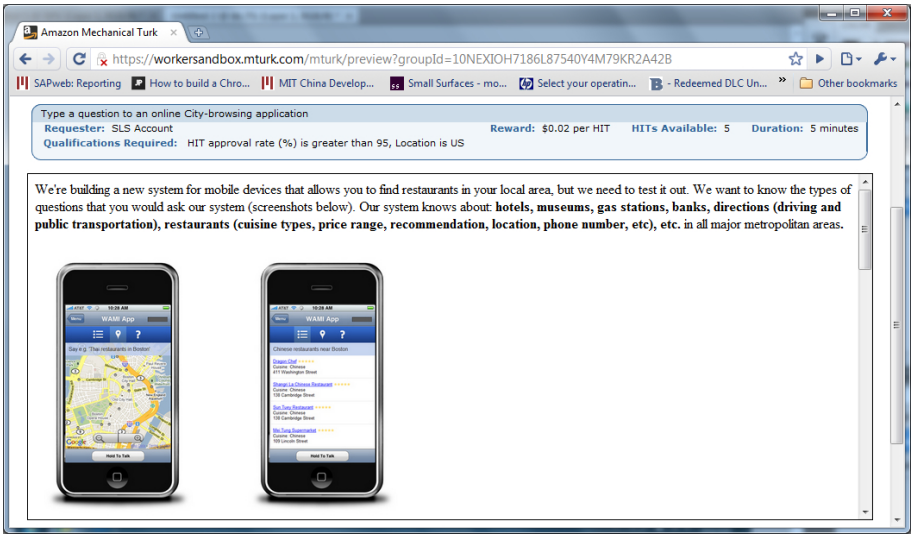

Figure 5-7: The Mechanical Turk HIT used to collect queries.

unchanged. Overall the entire collection process took approximately 3 days.

In the process we encountered several difficulties. We found that the AMT workers were extremely biased by the example sentences we provided. In our initial iteration of the HIT, we emphasized the Boston metro region, only to find that a significant portion of the worker queries also referenced Boston. Therefore we updated the examples to reference cities across the nation. Similarly, the HIT also suffered from worker abuse. In one such example, a single turker submitted over 5,000 sample inputs consisting of a single phrase structure "show me `CUISINE` restaurants in `CITY`", in which each sentence differed from the others only by the cuisine type or city. This data was no better than automatic template-based data, and was therefore thrown out. In our next iteration of the HIT, we made it clear that a variety of sentences were needed, and that simple swapping of proper nouns would result in a rejected submission.

5.3.3 Tagging Point-of-Interest Types

In order to use the sample AMT inputs as training data, we tagged the POIs in the sentences with their word-classes. Several well-known algorithms exist for this task such as the contextual semantic tagger in [3], however, due to the limited number of

sample queries, we chose to tag the data using rule-based methods. We requested that AMT workers provide correctly punctuated and capitalized inputs. Therefore, we leveraged the sentence grammatical structures to infer the POI tags. As an example, our script searched for the pattern "phone number of `OBJECT` restaurant" and inferred that `OBJECT` should be tagged as a restaurant. For instance, this pattern matches the sentence, "show me the phone number of All Asia restaurant" and outputs "show me the phone number of [`restaurant`] All Asia [/`restaurant`] restaurant". The script rules provided only a rudimentary pass at tagging POIs. We therefore manually took a second pass at tagging a 1,000 sentence subset of the queries, with the expectation that more accurate and automated algorithms could be used in the future as the number of samples queries increased. For the rest of this chapter, all mentions of the AMT sample text queries are in reference to the 1,000 manually tagged inputs.

5.3.4 Language Model Experiments

The goal of our experiments was to assess the utility of the typed sample queries to improve language model performance, using the recognizer's word error rate (WER) as the performance metric. First we established the baseline system's performance. We divided the 579 spoken utterances from our usability study into two subsets, reserving 288 utterances as the test set, and 291 utterances as a development set to compare against the typed sample query performance. The baseline recognizer gave a WER of 32.1%. This WER is similar to performance on other corpora from the *City Browser* domain [9], as shown in Table 5.1. The comparable baseline WER suggests that the performance on the turk corpus may be generalizable to the system's overall performance.

	Tablet	**Web**	**Car-Pilot**	**Car**
WER	27.4%	29.2%	31.9%	36.1%

Table 5.1: Word Error Rates of the *City Browser* Corpora [12]

We ran a number of experiments, evaluating language model performance using varying subsets of training data. The complete results are shown in Table 5.2. Here

Language Model Training		WER
0.	baseline (CB corpus + templates)	32.1%
1.	AMT dev set	30.6%
2.	CB corpus	34.3%
3.	AMT text	51.3%
4.	templates	47.4%
5.	templates (partial)	49.2%
6.	AMT dev set + AMT text	29.1%
7.	AMT dev set + CB corpus	27.7%
8.	AMT dev set + CB corpus + templates	30.3%
9.	AMT dev set + CB corpus + AMT text	26.5%
10.	AMT dev set + CB corpus + AMT text + templates	27.9%
11.	CB corpus + templates + AMT text	29.2%
12.	oracle	21.0%

Table 5.2: Language Model Experiments Word Error Rates. The *CB corpus* consists of 2,163 transcribed utterances from recorded user interactions with the previous *City Browser* system. The *templates* are 20,000 sample sentences generated from manually created template files. *Templates*(*partial*) represents 1,000 randomly selected sentences from the complete 20,000 set. The *AMT dev set* consists of the 291 transcribed interactions from AMT, held out from the test set. The *AMT text* represents the 1,000 tagged text input queries.

we describe each of these experiments in turn.

Our first two experiments demonstrate that transcribed interactions are the best training data for the language model, as these utterances are most representative of the test set. Experiment 1 shows that retraining with the 291 transcribed AMT utterances produced a language model with a WER of 30.6%, beating the baseline system. It should be noted that the development set was collected from the same task scenarios as the test set, accounting for the large reduction in WER with only a few training utterances. Similarly experiment 2 gave a WER of 34.3%, which is fairly close to the baseline. The higher WER is due to the fact that the *CB corpus* utterances were collected from past in-lab studies on the desktop interface, and therefore not as representative of AMT workers using the mobile design.

Experiments 3, 4, and 5 showcase the performance of the language model trained on the text data sets. Experiment 3 used the 1,000 tagged utterances obtained from AMT, while experiments 4 and 5 used the full 20,000 template sentences and a ran-

domly selected 1,000 sentence subset, respectively. The increased WER, in comparison with the results of experiments 1 and 2, signifies the differences between text and verbal data.

We then explored permutations of data sets to identify promising combinations for improving language model performance. We started with the *AMT dev set* and first added the *AMT text* sample queries, which gave a WER of 29.1%. Not surprisingly, replacing the *AMT text* data with the transcribed *CB corpus* utterances in experiment 7 improved the WER to 27.7%, and adding back in the *AMT text* dropped our WER to its lowest value of 26.5%. What's interesting is the addition of the templates. In both experiments 8 and 10, the addition of the templates increased WER. This highlights that the templates, which are intended for wide coverage of potential utterances, are not as representative of this particular test set.

In experiment 11, we showed how the *AMT text* data can improve language models in the absence of transcribed interactions. We obtained a 29.2% WER (9% improvement relative to the baseline) on the 288 utterance test set. This result validates our initial hypothesis that the typed text queries can be representative of actual spoken interactions with the system.

Finally, we performed an oracle experiment to see the best possible WER that could be achieved with language model improvements. This was done by using only the test corpus as the language model training data. We found the best performance of our language models to be a 21.0% WER. The reason for such a high WER is largely attributed to low recognition rates of proper nouns and missing vocabulary words. For instance, in the top 50 confused words, 24 were proper nouns, such as "Harvard", "Quincy", and "Cambridge". Similarly, in the top 15 deleted words, we find "museum", "fine", and "arts", corresponding to "Museum of Fine Arts", along with "Blue" and "Fin", corresponding to "Blue Fin restaurant."

5.3.5 Discussion

The experimental results first reinforce the notion that transcriptions of actual spoken interactions with the system are the best quality data for training a language

model. However, the results also highlight that typed text queries can nonetheless provide improvements, given that transcribed utterances can be time-consuming and expensive to collect.

Experiment 7 demonstrates that, given enough available data representative of the test set, the automatically generated template data and sample text queries may no longer be needed. However, this is not always possible, and in such cases supplementing with human-generated data has the potential to increase performance. These sample text queries were shown to outperform the templates in Experiments 8 and 10, a possible indication that it may be valuable to invest in the small costs of obtaining examples from AMT workers or alternatively revisiting the template generation.

The poor results of the typed turk data alone show that they are insufficient as the sole training data for the system, highlighting the differences between spoken and typed text. However, as the other results show, the typed text sentences are still good supplements to actual transcribed data. In particular, the similarity between the *CB corpus + templates + AMT dev set* and *CB corpus + templates + AMT text* results suggests that the typed turk text can be an adequate approximation of the spoken utterances.

5.4 Chapter Summary

In this chapter we documented the process of collecting user interactions with the system and retraining the recognizer's language model. The presence of a large number of incorrect system responses suggests opportunities for improving recognition, vocabulary, and understanding. Toward tackling this problem, we showed a 9% relative reduction in the WER of the *City Browser* recognizer by augmenting the language model training data with typed sample queries collected using Amazon Mechanical Turk (AMT). Moreover, we highlighted how quickly and cost-effectively such data could be collected, serving as a potential replacement for sentences automatically generated from template files.

Chapter 6

Vehicle-based Multimodal Interfaces

As features such as route guidance and media players are added to vehicle interfaces, the non-driving task demands for drivers increase. For route guidance systems in particular, the traditional methods of selecting destinations, scrolling through lists, and manually entering location names letter-by-letter are lengthy tasks with high cognitive load [32]. Speech-enabled interfaces have been shown to be safer alternatives compared with these visual-manual entry methods [6, 28].

There are two common types of vehicle speech-enabled interfaces. The first type focuses on speech-enabling the traversal of hierarchical menus that would be otherwise navigated manually. These systems often employ small vocabularies with fixed grammatical structure. The second type, which we explore in this project, allows unrestricted natural language speech input, in which users are not constrained to pre-defined information hierarchies. This approach has been shown to be more efficient and less distracting for drivers [6].

This chapter presents our work building a vehicle interface for *City Browser*. First we describe the vehicle, hardware, and system configurations. Second we present the vehicle interfaces we developed. Finally we discuss the results of a usability experiment on the interface.

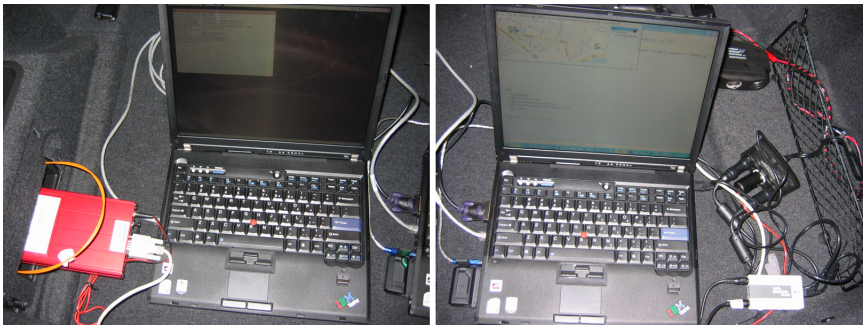

Figure 6-1: Vehicle System Configuration. One laptop acts as the language understanding processor, while the other interfaces with the vehicle.

6.1 System Components

We deployed the *City Browser* vehicle interface in a BMW 530xi sedan. Two laptops are mounted in the trunk of the vehicle. The first laptop performs language processing in addition to acting as the web server. The second laptop handles all vehicle interfacing, such as monitoring the vehicle state and user interactions. Additionally, the laptop display and audio are output to the car's built-in display and sound systems. For speech recording, a microphone speaker array is attached to the driver's sun visor. We use this microphone configuration since the car's built-in microphone had high noise-levels. Figure 6-1 shows the hardware configuration in the trunk of the vehicle.

6.1.1 Input Controls

The vehicle has a large set of input buttons and controls. The *City Browser* system primarily utilizes the iDrive controller. The iDrive is a knob which can be shifted up, down, left, and right. Additionally it can be pressed down for selections and rotated clockwise and counter-clockwise. The additional buttons available – used by early versions of our interface – are show in Table 6.1.1. The locations referenced are shown in Figure 6-2.

Input Button Name	Location	Purpose in BMW's Interface
Speech Recording	steering wheel	Speech recording
Speech Recording	center console	Speech recording
Star	steering wheel	Programmable (no default behavior)
Diamond	steering wheel	Programmable (no default behavior)
Telephone	center console	Telephone dialing
Menu	center console	Main menu button of vehicle navigation system
Up	steering wheel	Scrolling up menu results
Down	steering wheel	Scrolling down menu results

Table 6.1: Vehicle Input Keys

Figure 6-2: The interior control panel of the vehicle. (a) steering wheel buttons, (b) navigation display, and (c) center console iDrive and buttons.

6.2 Evolution of the Interface

The original *City Browser* vehicle interface was a custom-sized version of the desktop interface, as shown in Figure 6-3. In this version the recognition results were shown at the top of the screen, the point-of-interest (POI) results on the right, and the map on the left. The vehicle's star key was mapped to toggle between the sections. Depending on the section in focus, the keys were bound to various functions. A full mapping of the input keys and functionality is shown in Table 6.2. The biggest challenge of this design was the learnability of the key mappings. For instance, it was unclear which

Figure 6-3: Iteration 1 of the *City Browser* vehicle interface.

key was used to switch focus areas and which area was currently in focus.

6.2.1 Second Interface Iteration

To address the learnability of the previous interface, we attempted to more clearly highlight the section in focus. This is shown in Figure 6-4, in which the focus area is accented by graying out the rest of the interface. Additionally, we removed the use of the Star key to switch focus between areas, as most pilot subjects attempted to shift the iDrive instead. To allow for the iDrive up/down/left/right keys to be bound to switching focus areas required alterations to the map-controls. We added a sub-menu when the map was focused, in which users could select a specific-map-mode (such as panning mode), which would toggle the iDrive shifting to panning functionality. This map-menu is shown in Figure 6-4's top screenshot. Lastly we added a suggestions list, which could be reached by shifting down on the iDrive.

6.2.2 Third Interface Iteration

Informally collected feedback on the second interface highlighted two additional problems with the interface: legibility and complex controls. In particular, the font sizes were too small to be read while driving. Drivers could afford only quick glances, leaving insufficient time to read the lists of restaurant results.

In creating another iteration of the design, we followed the BMW look and feel,

Interface State	**Input Key**	**Function**
General	Diamond	Switch state
Map State	Up	Zoom out
	Down	Zoom in
	iDrive Spin Right	Zoom in
	iDrive Spin Left	Zoom out
	iDrive Up	Pan north
	iDrive Down	Pan south
	iDrive Left	Pan west
	iDrive Right	Pan east
Recognition State	iDrive Press	Select correction
	iDrive Spin Right	Scroll correction selection
	iDrive Spin Left	Scroll correction selection
	iDrive Left	Move to previous correctable word
	iDrive Left	Move to next correctable word
POI Results State	iDrive Press	Get details about selected POI
	Up	Scroll up
	Down	Scroll down
	iDrive Spin Right	Scroll down
	iDrive Spin Left	Scroll up

Table 6.2: Iteration 1 Vehicle Input Key Mapping

Figure 6-4: Iteration 2 of the *City Browser* vehicle interface. This iteration includes speech suggestions and map control icons.

shown in Figure 6-5[1]. Here we isolated each of the focus areas of the interface into their specific panes. We then arranged the panes in a 1-D ordering, as opposed to the previous design's 2-D layout. The user could shift the iDrive left and right in order to navigate to the different interface panes. We also found that previous users did little with correcting sentences, and therefore we removed the recognition results to simplify the interface. This design is shown in Figure 6-6.

6.3 Vehicle Interface Usability Study

Working with the MIT Age Lab, we designed a usability experiment for the vehicle-based *City Browser* interface. Our goal was to both assess the system and gauge the potential for speech integration into vehicle navigation systems. We recruited 80 participants from the Cambridge area, balanced across three age groups (25-34, 45-54

[1] `http://www.designingforhumans.com/.a/6a00d8341c870753ef010535c084e3970c-pi`

Figure 6-5: The BMW interface.

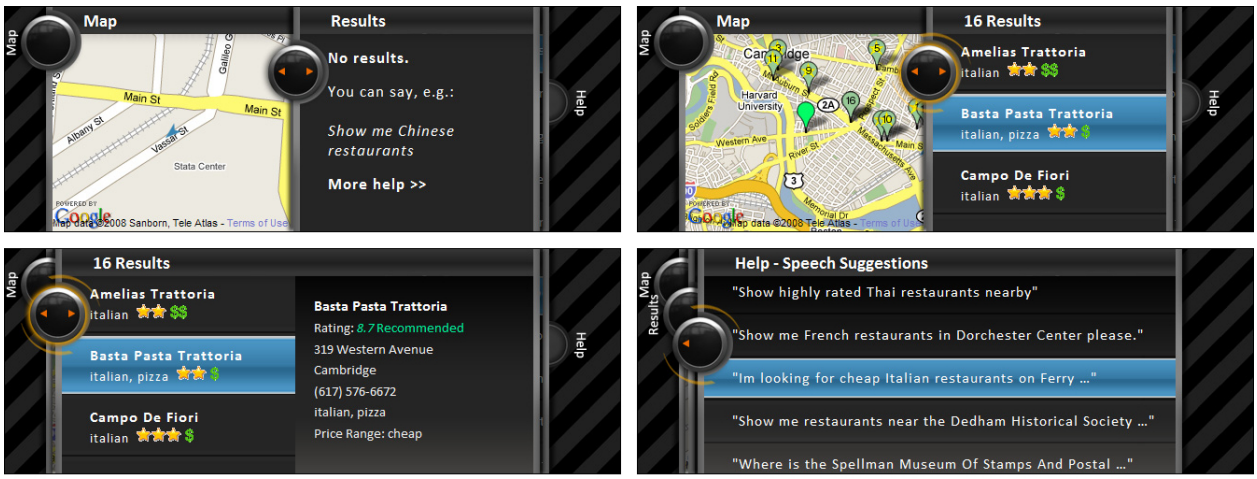

Figure 6-6: Iteration 3 of the *City Browser* vehicle interface. Clockwise from top left: the screen shown when the vehicle starts, the map results, speech suggestions, and detailed result information. This iteration also includes map control icons located on the circular iDrive symbol on the edge of each pane.

and 65-74) [12]. These participants were asked to interact with the vehicle interface by completing tasks after being shown a brief system overview. A photoelectric plethysmograph (PPG) to measure heart rate along with contacts to measure skin conductance level were attached to each participant. All tasks were conducted in a stationary vehicle.

The subjects were given 12 experimental tasks. Two example tasks and user interactions are shown in Figure 6-7, with the complete set of tasks shown in Table A.1. These tasks involved locating POIs using the interface.

Overall the results were positive: nearly 75% of participants found the system to be useful, over 90% found the system to be reasonably accurate, and over 85% found

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. **U**. Show me Chinese restaurants in Cambridge **S**. There are 12 Chinese restaurants in Cambridge. [Shows a results screen similar to figure 1b] **U**. [Uses iDrive controller to browse the list, leaving All Asia Cafe highlighted] **U**. Give me directions. **S**. Here are directions from this location to All Asia Caf.[Show directions on the map]
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. **U**. Where is one eighty Hancock Street in Quincy. **S**. Here is one eighty Hancock Street in Quincy [Shows the address marked on the map] **U**. Are there any cheap restaurants near there? **S**. There are fifteen inexpensive restaurants near one eighty Hancock Street in Quincy. [Shows a results screen similar to figure 1b] **U**. [Uses iDrive controller to browse]

Figure 6-7: Two example tasks given to participants in the vehicle interface usability study.

the system easy to learn. The usability study pointed out several areas for further work:

- Recognition results display. Currently there is no indication of what the system recognized. Although not necessarily required to always be present, there should be a quick method to display what the system heard.

- Improving learnability. The iDrive icon on the edge of the system's panes is intended to serve as an indicator for the available iDrive functionality. This was not immediately apparent to a large number of users and should be further explored.

- Suggestion selectability. Currently the suggestions screen displays unselectable example sentences. An improvement to the interface would allow for users to select the suggestions, automatically inputting them to the system.

6.4 Chapter Summary

In this chapter we presented a vehicle-based version of the *City Browser* system. Through multiple iterations, we developed an interface which maintained the functionality of the desktop version, but adapted controls and display for the automobile environment. Our usability study provided positive feedback for pursuing speech integration and encourages future work to incorporate speech functionality into navigation systems.

Chapter 7

Conclusion

In this thesis, we presented mobile and vehicle interfaces for the *City Browser* system. These interfaces were specifically designed to address the challenges of small screens and limited inputs on mobile devices. Additionally, we augmented the system architecture with a user interface platform to support more efficient reconfiguration for alternative interface designs. We then enhanced the back-end services with the ability to dynamically add data retrieved from third party sources.

Our mobile interface was tested through Amazon Mechanical Turk (AMT). We leveraged the service to rapidly collect a large number of queries, which were then used to retrain and improve the recognizer's language model. Our experiments clearly showed that the best training data is transcribed utterances from recorded interactions with the system. However, when such data are lacking, human-generated sample queries provide a potential alternative. After adding in text sample queries to our training sentences, we were able to see a 9% relative reduction in our recognizer's word error rates, bringing it from 32.1% to 29.2%. Similarly, AMT enabled us to efficiently distribute the system to many users and elicit feedback for improvements in future iterations of the system. Finally, our vehicle usability study on our navigation interface demonstrated the increasing readiness for speech integration into automotive guidance.

7.1 Future System Improvements

There still remain several areas for system improvement. The on-demand database system is currently limited to expanding the database immediately when a user shifts to an unsupported metro region. A more flexible design would allow finer granularity, augmenting the database for an unsupported request at any time. Similarly, the user should be able to shift metro regions using speech, as opposed to being limited to manual metro region controls.

The database system can additionally be improved with a caching scheme supporting concurrency. The current system uses text files written to disk, and could be made more robust with an SQL database.

A natural extension of the system would be developments for the transit-related domain. The current *City Browser* system focuses on points-of-interest, touching on transit briefly through simple driving directions and subway map overlays. In terms of directions, however, users frequently are more interested in getting public transportation schedules and route guidance. We did an initial exploration of adding transit support; however, we faced difficulties with transportation routing services. Public transportation data is rapidly becoming more available through the Google Transit Feed Specification[1], and may be more mature in the coming months. Similarly, an effort to develop open source routing services has been launched by OpenTripPlanner[2].

Another area for improvement is expanding system vocabulary and providing better data for the recognizer language models. Our oracle experiment results highlight both the potential gains and limitations of an improved language model. The rest of the gains could come from supporting additional POI names and categories.

User correction and guidance schemes are other areas for potential research. The work in [30] showcases the wide variety of speech correction mechanisms and their ability to handle out of vocabulary words and increase input rates. The *City Browser* system explored initial methods of showing recognition results, but did not provide

[1] `http://code.google.com/transit/spec/transit_feed_specification.html`
[2] `http://opentripplanner.org/`

a mobile concept correction interface. Furthermore, the system could be enhanced with the ability to guide the user in times of confusion, such as clarifying ambiguous concepts or requesting additional information to further specify the user query. These augmentations would drive the system toward a more-conversational interface.

7.2 Expanding to Additional Device Interfaces

As more mobile devices are brought to market, additional opportunities are created to extend the *City Browser* interface. For instance, the recent launch of the iPad offers a platform to build a touch-based tablet-sized interface. The increasing prevalence of Android-based phones could drive the development of a resizable mobile interface, one which could automatically scale to device screens.

Moreover, the initial success of the vehicle interface highlights the opportunities for applying the *City Browser* system to new domains, such as reviving the kiosk system for subway information booths, building large-scale TV interfaces, or designing intermediate netbook-sized views.

7.3 Final Words

The work in this thesis showcased the growing acceptance and usability of spoken dialogue systems through multimodal interfaces. We used *City Browser*, a map-based application, to highlight the potential of these systems. Our evaluations showed promising direction for creating rich multimodal interaction. With further improvements and developments, *City Browser* and other spoken dialogue systems have the potential to shift the interface paradigm toward natural conversational interaction.

Appendix A

Vehicle Usability Study Tasks

In this appendix we provide the 12 scenarios used in the vehicle-based user study [9].

Task	Description
0	Tutorial task, part of which involves the subject being instructed to say "Show me Chinese Restaurants"
1	You need to find a hotel in Somerville. Use the system to find the name of a hotel.
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	You're showing some friends around Boston and want to go to the Museum of Science. Get directions to the museum.
4	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	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	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	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	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	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	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 A.1: Tasks used in the pilot usability study of the vehicle-based *City Browser* system, reproduction of Table 4.3 in [9].

Bibliography

[1] Q. Brown, F. Lee, D. Salvucci, and V. Aleven. Interface challenges for mobile tutoring systems. *Proceedings of the 9th international conference on Intelligent Tutoring Systems*, 5091:693 – 695, 2008.

[2] C. Callison-Burch and M. Dredze. Creating speech and language data with Amazon's Mechanical Turk. *NAACL-2010 Workshop*, 2010.

[3] A. Cucchiarelli. Semantic tagging of unknown proper nouns. *Natural Language Engineering*, 1999.

[4] M. Dunlop and S. Brewster. The challenge of mobile devices for human computer interaction. *Personal and Ubiquitous Computing*, 6:235 – 236, 2002.

[5] J. Feng and S. Bangalore. Query parsing for voice-enabled mobile local search. *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009.

[6] C. Forlines, B. Schmidt-Nielsen, B. Raj, K. Wittenburg, and P. Wolf. A comparison between spoken queries and menu-based interfaces for in-car digital music selection. *In Proceedings of International Conference on Human-Computer Interaction*, 2005.

[7] A. Franz and B. Milch. Searching the web by voice. *Proceedings of the 19th international conference on Computational linguistics*, 2:1 – 5, 2002.

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

[9] A. Gruenstein. *Toward Widely-Available and Usable Multimodal Conversational Interfaces*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, 2009.

[10] A. Gruenstein, B. Hsu, J. Glass, S. Seneff, I. Hetherington, S. Cyphers, I. Badr, C. Wang, and S. Liu. A multimodal home entertainment interface via a mobile device. *Proc. of ACL Workshop on Mobile Language Processing*, 2008.

[11] A. Gruenstein, I. McGraw, and I. Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. *ICMI*, 2008.

[12] A. Gruenstein, J. Orszulak, S. Liu, S. Roberts, J. Zabel, B. Reimer, B. Mehler, S. Seneff, J. Glass, and J. Coughlin. City Browser: developing a conversational automotive HMI. *CHI*, 2009.

[13] A. Gruenstein and S. Seneff. Context-sensitive language modeling for large sets of proper nouns in multimodal dialogue systems. *Proc. IEEE/ACL 2006 Workshop on Spoken Language Technology*, 2006.

[14] C. Gutwin and C. Fedak. Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques. *ACM International Conference Proceeding Series*, 62, 2004.

[15] I. Hetherington. A multi-pass, dynamic-vocabulary approach to real-time, large-vocabulary speech recognition. *Proc. Interspeech*, pages 545–548, 2005.

[16] M. Johnston, S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. Walker, S. Whittaker, and P. Maloor. MATCH: an architecture for multimodal dialogue systems. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 376 – 383, 2002.

[17] A. Kittur, E. Chi, and B. Suh. Crowdsourcing user studies with Mechanical Turk. *Conference on Human Factors in Computing Systems*, 2008.

[18] J. Ledlie, B. Odero, E. Minkov, I. Kiss, and J. Polifroni. Crowd translator: on building localized speech recognizers through micropayments. *ACM SIGOPS Operating Systems Review*, 2010.

[19] S. Lee and S. Zhai. The performance of touch screen soft buttons. *Human Factors in Computing Systems*, 2009.

[20] M. Marge, S. Banerjee, and A. Rudnicky. Using the Amazon Mechanical Turk for transcription of spoken language. *ICASSP*, 2010.

[21] G. Matthias. Incremental speech understanding in a multimodal web-based spoken dialogue system. Master's thesis, MIT Department of Electrical Engineering and Computer Science, 2009.

[22] I. McGraw, C. Lee, I. Hetherington, S. Seneff, and J. Glass. Collecting voices from the cloud. *Language Resources Evaluation Conference*, 2010.

[23] S. Novotney and C. Callison-Burch. Cheap, fast and good enough: automatic speech recognition with non-expert transcription. *NAACL*, 2010.

[24] S. Oviatt. Multimodal interactive maps: designing for human performance. *Human-Computer Interaction*, 12:93–129, 1997.

[25] L. Reeves, J. Lai, J. Larson, S. Oviatt, T. Balaji, S. Buisine, P. Collings, P. Cohen, B. Kraal, J. Martin, M. McTear, T. Raman, K. Stanney, H. Su, and Q. Wang. Guidelines for multimodal user interface design. *Communications of the ACM*, pages 57 – 59, 2004.

[26] S. Seneff and J. Polifroni. Dialogue management in the Mercury flight reservation system. *Proc. Dialogue Workshop, ANLP-NAACL*, 2000.

[27] P. Tarasewich. Designing mobile commerce applications. *Communications of the ACM*, pages 57 – 60, 2003.

[28] L. Tijerina, E. Parmer, and M. Goodman. Driver workload assessment of route guidance system destination entry while driving: a test track study. *Proceedings of the 5th World Congress on Intelligent Transportation Systems*, 1998.

[29] M. Turunen, J. Hakulinen, A. Kainulainen, A. Melto, and T. Hurtig. Design of a rich multimodal interface for mobile spoken route guidance. *Interspeech*, 2007.

[30] K. Vertanen. Recognition and correction of voice web search queries. *Proceedings of the International Conference on Spoken Language Processing*, pages 1863–1866, 2009.

[31] K. Vertanen and D. MacKay. Speech Dasher: fast writing using speech and gaze. *CHI: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010.

[32] K. Young and M. Regan. Driver distraction: a review of the literature. *Report (Monash University. Accident Research Centre)*, 2003.