Developing Attribute Acquisition Strategies in Spoken Dialogue Systems via User Simulation

by

Edward A. Filisko

B.S., University of Michigan, Ann Arbor (1999)
S.M., Massachusetts Institute of Technology (2002)

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

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author ....
Department of Electrical Engineering and Computer Science
May 17, 2006

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

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

Developing Attribute Acquisition Strategies in Spoken Dialogue Systems via User Simulation

by
Edward A. Filisko

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2006, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

A spoken dialogue system (SDS) is an application that supports conversational interaction with a human to perform some task. SDSs are emerging as an intuitive and efficient means for accessing information. A critical barrier to their widespread deployment remains in the form of communication breakdown at strategic points in the dialogue, often when the user tries to supply a named entity from a large or open vocabulary set. For example, a weather system might know several thousand cities, but there is no easy way to inform the user about what those cities are. The system will likely misrecognize any unknown city as some known city. The inability of a system to acquire an unknown value can lead to unpredictable behavior by the system, as well as by the user.

This thesis presents a framework for developing attribute acquisition strategies with a simulated user. We specifically focus on the acquisition of unknown city names in a flight domain, through a spell-mode subdialogue. Collecting data from real users is costly in both time and resources. In addition, our goal is to focus on situations that tend to occur sporadically in real dialogues, depending on the domain and the user's experience in that domain. Therefore, we chose to employ user simulation, which would allow us to generate a large number of dialogues, and to configure the input as desired in order to exercise specific strategies.

We present a novel method of utterance generation for user simulation, that exploits an existing corpus of real user dialogues, but recombines the utterances using an example-based, template approach. Items of interest not in the corpus, such as foreign or unknown cities, can be included by splicing in synthesized speech. This method allows us to produce realistic utterances by retaining the structural variety of real user utterances, while introducing cities that can only be resolved via spelling. We also developed a model of generic dialogue management, allowing a developer to quickly specify interaction properties on a per-attribute basis. This model was used to assess the effectiveness of various combinations of dialogue strategies and simulated user behavior.

Current approaches to user simulation typically model simulated utterances at the intention level, assuming perfect recognition and understanding. We employ speech to develop our strategies in the context of errors that occur naturally from recognition and understanding.

We use simulation to address two problems: the *conflict* problem requires the system to choose how to act when a new hypothesis for an attribute conflicts with its current belief, while the *compliance* problem requires the system to decide whether a user was compliant with a spelling request. Decision models were learned from simulated data, and were tested with real users, showing that the learned model significantly outperformed a heuristic model in choosing the "ideal" response to the conflict problem, with accuracies of 84.1% and 52.1%, respectively. The learned model to predict compliance achieved a respectable 96.3% accuracy. These results suggest that such models learned from simulated data can attain similar, if not better, performance in dialogues with real users.

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Scientist

Acknowledgments

I would first like to extend my gratitude to my advisor, Dr. Stephanie Seneff, who has guided and supported me through every step of my graduate education. She has truly inspired me over the past six years with her constant flow of creative ideas and her persistent desire to tackle the most challenging of problems.

I would like to thank the members of my thesis committee, Dr. Michael Collins and Dr. Regina Barzilay. I greatly appreciate their comments and helpful suggestions.

The members of the Spoken Language Systems Group have provided a very supportive daily environment over the years. Thanks go to all the research staff and grad students for their willingness to help and share ideas. I also want to thank Joe Polifroni for his especially intriguing and helpful conversations on discourse and dialogue.

Thanks go to my awesome SLS officemates for making the daily work life tolerable through the years: Grace Chung, Xiaolong Mou, Brooke Cowan, Vlad Gabovich, Chia-Hao La, Kate Saenko, Karen Livescu, Han Shu, Mitch Peabody, and Chih-yu Chao. Thanks also go to Marcia Davidson for her humorous interludes and inspirational wit.

I am additionally grateful to those outside school who enabled my learning of a different sort, providing a break from my graduate work. I want to thank Kieran Jordan and Dancers for an amazing and rare opportunity to experience Irish music and dance. I would also like to thank Chef Argyropoulos, Chef Berry, Chef Merlo, Chef Wright, and Kara Nielsen for enabling me to stay in tune with the culinary world. Special thanks go to Craig Gerardi, Wayne Hsiao, Dora Kelle, Bobby Middleton, Alex Park, and Jane Wu for taste testing my food and making every day thought-provoking and fun.

I truly appreciate the support from my family and friends, especially Theresa, Andy, Chris, Carmen, Christian, Johnna, Rrrranjana for her e-mail updates, Maninder for his inspiring wake-up calls, Manpreet for her encouragement, and, of course, the Cheese Club.

Most importantly, I want to thank my parents who, no matter what I do, always give me constant encouragement and their unconditional support.

Ed Filisko
May 17, 2006

This research was supported by an industrial consortium supporting the MIT Oxygen Alliance, by the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan, and by Nokia.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent developments in speech technology have given rise to a growing number of speech-enabled applications in our everyday lives. The variety of such applications is vast, ranging from voice dictation software to voice control of a car stereo. One of the more advanced applications of speech technology is the spoken dialogue system (SDS). An SDS is a computer application that supports conversational interaction with a human to perform some task. Such systems have covered a wide range of domains, including call routing for customer service [34], information kiosk communication [36], battlefield simulations [77], train schedule information [49], off-line task delegation [71], weather information [91], autonomous mobile robot communication [45], flight reservations [81], leisurely activity information [75], conference room reservations [10, 9], e-mail access [82, 79], restaurant information [17], and bus schedule information [60].

An SDS is the product of many different speech technologies, including audio capture, speech recognition, natural language understanding, dialogue management, natural language generation, and text-to-speech synthesis. Figure 1-1 shows the flow between these components in a typical SDS. A user's speech is passed from an audio server to a speech recognizer, which outputs hypotheses of what the user said. The natural language understanding component, or parser, tries to extract meaning from those hypotheses, creating some semantic representation. The dialogue manager assesses that meaning in the context of the dialogue history and outputs a semantic

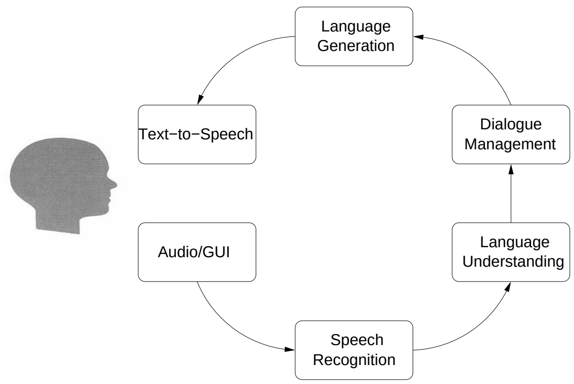

Figure 1-1: Typical flow between the components of an SDS.

representation of its response. A generation module converts the semantic representation into a natural language sentence, which is presented to the user with the help of a text-to-speech synthesizer.

While the effectiveness of SDSs has improved significantly over the past several years, a critical barrier to widespread deployment remains in the form of communication breakdown at strategic points in the dialogue, often when the user is trying to convey a critical piece of information that the system repeatedly misunderstands. This situation is particularly likely to occur whenever the user is attempting to supply a named entity from a large or open vocabulary set. For example, a weather domain might have information for several thousand cities, but there is no easy way for the system to inform the user about which cities it knows. If the user requests information for an unknown city, the system will likely misrecognize it as an altogether inappropriate, known city. If the vocabulary is static and the system lacks an explicit model for unknown words, the user's repeated attempts can only lead to failure.

Figure 1-2 shows an example dialogue between a user and the MERCURY flight reservation system [69], in which the user was attempting to travel to Roseau, Dominica, a city and country that the system did not know. In U1, "the island of Dominica" is misrecognized as "Seattle and Atlanta," while in U3, "Roseau, Dominica" is hypothesized as "Toronto, Connecticut." Despite the user's denial of these city names, and obvious frustration, the system futilely continues to hypothesize city, state, and country names. Clearly, a more intelligent model would be required to

U1:	I would like to fly from Miami, Florida to the island of Dominica [*Seattle and Atlanta*] on December 17th.
S1:	Okay, from Miami to Seattle on Monday, December 17th. Can you provide a departure time or airline preference?
U2:	No, I don't want to go to Seattle.
S2:	What city does the flight arrive in?
U3:	Roseau, Dominica [*Toronto, Connecticut*].
S3:	Okay, from Miami to Toronto on Monday, December 17th. Can you provide a departure time or airline preference?
U4:	No, I'm not going to Toronto. I want to go to Dominica [*Jamaica*].
S4:	Okay, from Miami to Kingston on Monday, December 17th. Can you provide a departure time or airline preference?
U5:	No, I don't want to go to Kingston [*Houston*], I want to go to Dominica [*Connecticut*].

Figure 1-2: Example dialogue from the MERCURY flight domain, illustrating a situation where the user (U) is attempting to fly to a city that the system (S) does not know. Words in brackets indicate what the system recognized.

detect and recover from such situations.

A common approach to help improve the system's behavior in such circumstances is the use of confidence scoring, in which the recognizer provides a score for each word indicating how certain it is that the given word was spoken [42, 86, 38]. A low confidence score could then signal the system to ask the user to confirm the given value. Further improvement could be accomplished by using an explicit unknown word model [7], which could predict that a given word or phrase spoken by the user may be absent from the recognizer's vocabulary and is therefore unrecognizable. In such a case, the system could apologize and suggest that the user propose a nearby larger city; however, a more productive strategy would be to request more information about the city, such as a spelling via speech or the telephone keypad. Armed with this additional information and a large external database of city names, for example, the system could likely identify the intended city name.

A challenging aspect of the above-outlined strategy is the design of the algorithms for error detection and recovery. Confidence scoring and unknown word detection are themselves prone to error; hence the system can never be certain that it does not know the word [44]. A strategy that invokes tedious confirmation subdialogues for every hypothesized city would surely annoy the user; yet an obtuse unwillingness to acknowledge confusion will lead to communication breakdown and user frustration. The user's behavior, in addition to confidence scoring and unknown word detection, can signal a misrecognition [44]. The system thus needs to make use of multiple cues to decide when to invoke an error recovery subdialogue.

The error detection problem is further complicated when a newly hypothesized value for a given attribute conflicts with the value currently believed by the system. We call this situation the *conflict problem*, and the dialogue manager must decide how to respond in such circumstances. Many systems simply overwrite the existing value with the new value and continue the dialogue, placing a burden on the user to correct the error if, in fact, an error was made. Ideally, however, the new value should be ignored if it was *not* intended by the user, implicitly confirmed (e.g., "Okay, to Chicago. Can you provide a date?") if it *was* the user's intended city, or perhaps a spelling should be requested if the intended city was actually *unknown* to the system.

One challenging aspect of developing and evaluating such error handling strategies is the fact that it is extremely difficult to collect appropriate user data. Not only is it costly to collect user data in general, but it is also difficult to influence the user's behavior so as to enhance the occurrence of desirable scenarios. For example, depending on the user and the system's extent of knowledge, unknown words may not be a very common occurrence. However, when an unknown word is uttered by a user, the system should have a strategy to address the situation. It then becomes problematic to define an effective evaluation criterion for conditions that are inherently rare, and, in any case, it requires a period of weeks or months to collect sufficient data from real users both for development and for evaluation.

An attractive solution is to employ a *simulated* user, both to help develop the system and to provide highly controlled evaluation experiments. Increasingly, researchers in spoken dialogue systems have been exploiting user simulation to aid in system development [23, 47, 64, 51, 15]. User simulation is beneficial for the initial development of a dialogue system because it eliminates the need for real users, with whom experiments are costly in both time and resources. It can also be challenging to encourage humans to use such experimental systems. On the other hand, a simulated user can be configured to behave in a predictable manner so that specific dialogue strategies may be tested and further developed. Another notable benefit is that a huge number of simulated dialogues can be generated in the same amount of time required to collect just a few dialogues from real users. Once the strategies of interest have been sufficiently developed with a simulated user, the system, which should be able to more robustly handle problematic user behavior, can then be made available to real users.

The ease with which some dialogue strategies can be developed and incorporated into an SDS depends on the generic nature of the dialogue manager. Traditionally, a separate dialogue manager is designed for each domain, since the complexity and functionality for unique domains often differs. The consequence of this design, however, is the superfluous repetition of functionality shared by various domains. *Generic dialogue management* is a framework in which the functionality provided by dialogue strategies can be utilized across domains. Such an implementation is beneficial for developers, since an array of strategies is immediately available for a new domain, and each strategy will be consistent across the domains that support it.

User simulation and generic dialogue management are invaluable tools for developing and adjusting dialogue strategies. However, it is often the case that rules must be modified by the developer in order to obtain the desired behavior. Manual rule specification can result in systems very robust in handling errors; however, a significant amount of time is also required to devise and tune the rules. In addition, a developer is likely unable to anticipate rules to handle all possible dialogue situations.

An alternative to manual rules is to employ a learning mechanism to automatically determine from data how the system should behave in certain circumstances. Given that user simulation facilitates the generation of a large number of dialogues, while incurring little cost, it has recently been used to provide data from which strategies or decision models can be learned [47, 39]. A system can be developed and tested on such simulated data and then deployed to real users, hopefully resulting in better performance than if the system were not exposed to any pre-deployment testing.

1.1 Thesis Goals and Approach

The intent of this thesis is to provide a framework for developing dialogue strategies by employing user simulation. Specifically, we focus on strategies enabling the dialogue manager to acquire users' intended attribute values that happen to be unknown to the speech recognizer. These unknown values can lead to unpredictable behavior on the part of the system, as well as the user, so it is advantageous for both parties to facilitate the system's acquisition of such values.

Our research is conducted in the flight reservation domain. The acquisition of an unknown city name is accomplished by requesting the spelling of the city from the user, and finding a match in a large off-line database of city names. The development of this strategy required user input, which could have been obtained in three ways: 1) conducting a focused data collection from real users, 2) incorporating the strategy into an existing dialogue system, or 3) utilizing a simulated user. The first way would require significant time and resources, and would likely result in a relatively small amount of data. In addition, the resulting corpus would be static. We did not want to develop the strategy by adding it to an existing system, since unpredictable system behavior could dissuade regular users from accessing the system. Therefore, we chose to employ user simulation, which could provide a large number of unique dialogues, and a dynamic corpus of utterances. The simulated user could be configured to focus on unknown city names, providing opportunities for the system to test its error recovery strategies.

We developed a novel method of utterance generation for user simulation, which exploits an extensive corpus of preexisting user queries, but recombines the utterances using an example-based, template approach. The system incorporates items not in the corpus (e.g., foreign or unknown cities) by splicing in synthetic waveforms. This method allows us to generate realistic utterances by retaining the structural variety in the ways users may pose questions, while introducing in a controlled way cities that would require an error recovery subdialogue to be resolved. Additionally, we can synthesize the spoken spellings necessary to execute the recovery subdialogue. The strategies could then be developed in the context of these simulated dialogues.

Our method of user simulation utilizes speech, which is unlike the approach in the majority of related research [23, 47, 65, 58, 39], which models simulation at the intention level, assuming perfect recognition and natural language understanding. However, some approaches, such as ours, actually desire imperfect recognition. If speech is used, the system is forced to deal with errors that arise from both recognition and understanding. These errors naturally present the system with opportunities to exercise its recovery strategies, as opposed to having the developers introduce simulated errors in a separate step [85].

Another goal was to provide a means by which a newly developed dialogue strategy could immediately be made available for a new domain. Thus, we designed and implemented a model of *attribute-centric* generic dialogue management, in which dialogue interaction properties can be specified for each individual attribute. The properties are specified in an external file, facilitating the rapid modification of dialogue strategies in a domain. This design would allow the developer to specify, for example, a separate spectrum of confidence scores for each attribute. For a city name, a high score could trigger an implicit confirmation, a moderate score could trigger an explicit confirmation, and a low score could elicit a spelling or keypad request. For a date, a high score might trigger an implicit confirmation, whereas any other score might cause an explicit confirmation. Such control is especially useful in a simulation environment where different configurations of the dialogue manager and simulated user must be explored, indicating which strategies might be most successful for certain user behaviors.

The development of dialogue strategies and observation of dialogues often reveals difficult decisions that must be made by the dialogue manager. An additional goal of this thesis was to provide a method by which such decisions could be made more easily. The *conflict problem* occurs when a newly hypothesized city name conflicts with the system's currently believed value for that attribute. This would seem to imply that the system had misunderstood the earlier hypothesis; however, it is often the case that the new hypothesis is the result of an incorrect recognition. Finding it difficult to heuristically specify rules guiding the dialogue manager's response in this situation, we decided to have the system learn an appropriate decision. We defined a set of "ideal" actions that the dialogue manager should take upon encountering the conflict problem. These actions were based on the beliefs of the system and the intentions of the user, which are available during simulated dialogues, but must be hand-labeled in a post-processing step for dialogues with real users. Hence, the simulation system enabled the collection of a large number of features and corresponding ideal actions, without requiring any manual annotation. We applied a learning algorithm to this data in an attempt to predict the appropriate action given a set of feature values. Accomplishing the same task with real users would have required time-consuming real user data collection, as well as hand-labeling of the data. The learned model was later tested in dialogues with real users in order to more realistically determine its utility. Results show that it performed significantly better than a corresponding heuristic-driven model.

We explored one more situation in which the dialogue manager was required to make an important decision. The *compliance problem* arises after the system has asked the user to spell a word. Given the user's response, the dialogue manager must then determine whether the user was *compliant* (i.e., provided a spelling) or *noncompliant* (e.g., repeated the city name without spelling). This decision is significant since both a main domain recognizer and a specialized spelling recognizer will output hypotheses of the user's response to a spelling request; the dialogue manager must decide which hypothesis to trust to make the dialogue continue smoothly. We applied the same simulation/learning technique used for the conflict problem to generate a model to predict a user's compliance. The model was completely developed with a simulated user and subsequently tested in real user dialogues, showing very good performance.

The spelling, simulation, and learning techniques that we present here can be used in the development of dialogue strategies for a new dialogue system, or even to enhance the capabilities of an existing system. Furthermore, we show that dialogue decision models developed on a simulated user can show good performance when deployed for real users.

1.2 Background and Related Work

This section discusses background and previous research in the major areas related to the work conducted in this thesis: unknown words and spelling, learning in dialogue management, user simulation, and the acquisition of attribute values.

1.2.1 Unknown Words and Spelling

A common strategy used in human-human dialogue for handling an out-of-vocabulary (OOV) word is for one participant to ask the other to repeat or spell the unknown word. This provides the inquisitor with more detailed information, which can facilitate an understanding of the word. Several dialogue systems have employed such a strategy in dealing with a confusing or unknown word [6, 66]. We aim to employ such a spelling strategy in our system.

Chung, Seneff, and Wang conducted research, in which they developed the capability to utilize a pronounced version of a word to greatly enhance the accuracy of a letter recognition task. They successfully integrated this speak-and-spell technology into a personal name enrollment task [16, 70]. The interest of the current thesis is in evaluating whether a similar technique would be useful for the error recovery problem.

Place names, such as cities, states, and airports, are prevalent and problematic in any domain where geography plays a dominant role. For example, a weather information system or a flight reservation system must have the city, state, or country names exactly correct in order to be useful. In real user interactions, it is inevitable that some city will be mentioned that is unknown to the system, often being misrecognized as a known city. The dialogue manager must determine whether a poorly scoring hypothesized city name is, in fact, a misrecognized but *known* city, or an entirely unknown word. Such uncertainty must be resolved in a manner that is time-efficient and does not overburden the user.

For example, in the case of a weather information system, very common geographic names would most likely be known to the recognizer (e.g., *New York*, *Moscow*). If a user wants to know the weather in his hometown of Menominee, for example, which is unknown to the recognizer, the system is faced with a problem. If the recognizer is not equipped with an unknown word detector, the closest acoustic match in the recognizer's vocabulary will be chosen as the best hypothesis. The user will then have to proceed through potentially many clarification turns in which the system repeatedly hypothesizes incorrect city names. Since "Menominee" is OOV, the system will never successfully acquire that value.

If, however, a large external database is available (e.g., United States Census data), that the system could consult given a hypothesized spelling of the unknown city, there is hope that the system will find the user's intended city name. Such an approach of access to a much larger vocabulary in spell mode has been successfully applied, for example, by Schramm, Rueber, and Kellner [66]. In the most extreme case, the large external database would be the World Wide Web. One could imagine the system posing a search query on the Web using the hypothesized spelling of the unknown city to obtain, for example, weather information.

While it would be possible to represent several thousand city names explicitly in a recognizer's vocabulary, performance would likely suffer. As the size of the vocabulary increases, so does the perplexity, which is correlated with reduced recognition accuracy [56]. In addition, the acoustic confusability between words will likely increase with a larger vocabulary, which will lead to either reduced recognition accuracy or extensive disambiguation subdialogues. An alternative is to utilize spelling, which can be slightly more taxing on the user, but can also reduce the need for disambiguation subdialogues. Spelling also provides the system with the ability to acquire orthographic information about unknown words, which increases a system's potential capabilities significantly.

It should be mentioned that the flexibility afforded to the user in spelling a word creates a new task for the system. Instead of simply looking up a hypothesized word in a lexicon, the system must now remain open to the possibility that the hypothesized word (i.e., sequence of letters) has been misspelled or misrecognized. The recognition accuracy for spoken letters can be quite low, due to acoustic confusability. For instance, the members of the *E-set* (B, C, D, E, G, P, T, V, Z) are well-known for being confusable to a recognizer, as discussed in previous studies [53]. A technique, such as spell checking, must be utilized to address this issue. Depending on the robustness of the spell checker, however, the system may or may not be able to determine the intended word from an errorful hypothesis. Additionally, with a large database, a potential misspelling could correspond to multiple data items. In such a case, the system would need to initiate a disambiguation subdialogue to resolve the ambiguity.

There are other pragmatic issues to consider in obtaining spelled data from a user whether via keypad or speech. The problem of disambiguating keypad sequences has been addressed using both dictionary-based [20] as well as probabilistic [52] approaches. In both input modes, the user may use abbreviations such as "S T P E T E R S B U R G" for "Saint Petersburg." Another problem is that humans often spell words in creative ways. Some use the military style (e.g., where "alpha" = "A"), while others use example-based spelling (e.g., "B as in 'Boy'"). Some users may include the word "space" to mark the word boundaries of a multi-word sequence, such as "N E W space Y O R K." Some may simply enter a letter sequence containing several meaningful chunks, as in "N E W Y O R K J F K N Y C" for Kennedy Airport in New York City. Many of these issues have been addressed by Schramm, Rueber, and Kellner [66].

Bauer and Junkawitsch [6] describe work which is quite closely related to ours in two respects: 1) they utilize a spoken spelling to disambiguate a large city database, and 2) they employ user simulations to acquire inputs for the spoken spelling. Their strategy is similar to ours in that confidence scoring is used for error detection, and the initial recognizer is configured to support only a small subset of the full city set (1,500 out of 22,000 possible cities in Germany). Their work differs from ours in that the task is isolated city name recognition rather than general language understanding, and their spell-mode recognizer was configured as an isolated letter task, which greatly enhances performance. In a test evaluation of about 1,000 spoken city names, followed optionally by simulated spellings, they obtained an impressive overall recognition accuracy of 96.5%, with spell mode being invoked for about 45% of the city names.

Oria and Vetek [56] recently described two post-processing algorithms to robustly recover from misrecognitions in a continuous spelling task. In their "spell-and-speak" algorithm, a spelling is requested from the user. The hypothesized letter sequences are matched against a reference list of words, from which a finite-state grammar is then dynamically generated. The user is then asked to pronounce the word, which is recognized against the grammar. The system then explicitly confirms the recognized value with the user. In their second approach, a letter-based N-gram model is trained on a large number of words, representative of a specific domain. A similarity coefficient is then calculated against the words in a list based on the number of shared N-grams. Both algorithms result in near perfect accuracy. This system is especially interesting in that it supports a mixture of any of the following three spelling behaviors: standard spelling with letters only, example-based spelling, and military spelling. However, users most often preferred the standard method of spelling.

Our work differs from that of Oria and Vetek in that our spelling algorithm was developed with a simulated user. In addition, the recovery algorithm was ultimately applied in the context of a task-based dialogue with real users, as opposed to an isolated spelling task. These more realistic circumstances gave rise to a variety of interesting and unanticipated user responses to the spelling request.

1.2.2 Learning in Dialogue Management

The process of developing a dialogue manager for a spoken dialogue system is a tedious process requiring collection and analysis of many dialogue sessions, in order to identify and address various unanticipated situations that may arise. Many dialogue strategies are based on rules, meticulously specified by the developer. An initial set of rules is often rather basic and inadequate, but continued use of the system allows the developer to modify and refine the rules, resulting in a system with good performance over time. An additional benefit is that rules can be specified to handle very specific dialogue situations, which may occur rarely in actual use. The major drawback of the manual method is that a significant amount of time is required to devise and hand-tune the rules. Furthermore, a developer is likely unable to anticipate all possible dialogue situations, in which case, additional massaging of the rules may be necessary.

An alternative method is to employ a learning mechanism, in which the system automatically obtains rules indicating how to respond in a given situation. The automatic acquisition of such rules by the dialogue manager can save a significant amount of work, possibly handling phenomena that an expert developer may not anticipate. Learning can take advantage of many diverse features, including those that may be specific to a particular system, in order to learn the best action to be performed in a given situation.

Both reinforcement learning and supervised learning have commonly been used in the area of dialogue management. Reinforcement learning is typically used to learn the optimal dialogue policy for a given task [47, 82, 75, 39], while supervised learning has more often been employed for classification tasks, such as predicting problematic dialogues [49, 80, 83] and identifying a given utterance as a user correction [48, 41].

In their work, Singh et al. [75] were able to optimize the performance of the NJFun system by employing reinforcement learning. NJFun is a telephone-based dialogue system providing information on a variety of activities in New Jersey. They focused on two common problems encountered in dialogue policy design: the type of initiative to employ when requesting an attribute, and whether the system should confirm an attribute value once acquired. It may be helpful at times, when a system fails to understand the user, to enter a system initiative mode, in which very specific questions are asked. This approach restricts what the user may say, but it may also more quickly allow the system to acquire the user's intended attributes. Similarly, the choice of whether to implicitly or explicitly confirm a hypothesized value could make the dialogue unnecessarily lengthy. There is little agreement among researchers in the field as to what the optimal strategies should be, which is one reason why learning the optimal dialogue policy is advantageous. The authors collected 311 dialogues, from real users, for which the dialogue policy was determined via exploration at certain choice points. An optimal policy was learned from this data and was tested, showing improved task completion in the test set as compared to the training set. The optimal policy said that the system should employ user initiative at the outset of a dialogue, and back off to system initiative when repeating a request for an attribute. The policy also generally said to confirm a value with the user at lower confidence values.

Supervised learning approaches to dialogue management are typically applied in the classification of problematic utterances or user corrections. Various features have been used by researchers in such approaches, including discourse features [40], previous system question types [80], user responses to system prompts [44], and prosodic cues [46].

A commonly used tool for classification tasks in dialogue management is the RIPPER [18] rule induction algorithm, which is briefly summarized here. Further details are available, as reported by Cohen [18]. RIPPER takes as input the names and values of a set of features, a set of class names to be predicted from those features, and a set of training examples, each consisting of a vector of feature values and a class name. RIPPER induces rules on a per-class basis, ordered from lowest to highest frequency, leaving the most frequent class as the default rule. Rules are induced, while heuristically maximizing coverage for each rule. The rules are then pruned according to a *minimum description length* metric. A post-processing step optimizes the rule set further with additional pruning. The algorithm outputs an ordered set of rules that can be used to classify future examples. In order to classify a new example, a linear scan of the rules is performed against the feature values of the new example, and the class corresponding to the first matching rule is selected. If no rules match explicitly, the class for the default rule is chosen.

Litman, Walker, and Kearns [50] adopted a machine learning approach to detect poor speech recognition performance. They obtained a corpus of 544 real user dialogues from three dialogue systems: ANNIE [43], a voice dialing and messaging system, ELVIS [82], an e-mail access system, and TOOT [49], a telephone-based system for accessing on-line train schedule information. Each utterance was hand-labeled as having good or bad speech recognition. A variety of features was extracted from this data and was provided to RIPPER in order to learn a set of rules to detect poor speech recognition. While a standard approach to detecting poor speech recognition typically involves only the confidence score based on acoustic measures, this work utilized features at the acoustic level, the lexical level (i.e., the recognized text string), as well as the dialogue level, including elapsed time, number of dialogue turns, number of cancel requests by the user, number of rejected user utterances, and number of help requests initiated by the system. Experiments were performed with models trained on several combinations of the features, each showing an improvement over the baseline (always predict good) of 52% accuracy. The best performance of 77.4% was achieved with a model trained on all features, and the rules contained a mixture of acoustic, dialogue, and lexical features.

Levow [48] used supervised learning to distinguish original user inputs from user corrections. She differentiated between corrections in response to rejections, and those in response to misrecognitions. A corpus of 7,752 user utterances was obtained from a field trial of a speech-only interface to computer desktop applications, such as e-mail and weather reports. Acoustic-prosodic features, including utterance duration, pause length, pitch, and amplitude, were used to train a decision tree. Such features have been shown to indicate when a user is getting frustrated [57]. For example, a user might hyperarticulate a word if the system continues to misrecognize it. The features were obtained mostly automatically with some minor hand-adjustment. The author obtained 30.7% accuracy in distinguishing between corrections in response to rejections and those in response to misrecognitions. However, the best performance was obtained in distinguishing original user inputs from user corrections, with a classification accuracy of 75%.

Hirschberg, Litman, and Swerts [41] also explored the automatic identification of user corrections by using prosodic features (e.g., pitch range, loudness, speaking rate), in addition to features from the speech recognizer (e.g., grammar, best hypothesis, acoustic confidence score), experimental features (e.g., system's current initiative), and features of the dialogue history (e.g., turn number, keyword hypotheses). The features were obtained from a corpus of 152 dialogues between humans and TOOT. RIPPER was used to produce a model predicting whether a given utterance was a correction or a non-correction. Various combinations of the features were explored and it was found that using only the recognizer features or only the prosodic features both resulted in significant improvements over the baseline. The best performance, however, was obtained with a combination of all feature types, reducing the baseline error from 28.99% to 15.72%.

Litman and Pan [49] also used supervised learning to predict poor speech recognition, but they utilized the prediction to potentially adapt the dialogue strategies employed by the system. Their corpus contained 120 dialogues between humans and TOOT. Twenty-three features were computed, including acoustic, dialogue-level, and lexical features. RIPPER was used to induce a model to predict poor speech recognition. The best learned classifier involved a single feature that predicted the number of previously misrecognized user utterances; it correctly classified almost 80% of the dialogues. This predictor was used to signal a communication problem in the dialogue, at which point, the dialogue system could adjust its strategies to be more conservative. Their adaptive system attained 65% accuracy in task success, compared to 23% accuracy for a comparable non-adaptive system.

In their work on detecting communication problems, van den Bosch, Krahmer, and Swerts [80] used the sequence of the six most recent question types (e.g., open question, explicit/implicit verification), and the word graphs corresponding to the associated user responses to learn a model using RIPPER. Training data consisted of 3,739 question-answer pairs from 444 dialogues from a Dutch dialogue system providing train schedule information. The type of each question was hand-labeled. A communication problem was defined as the situation when the system's hypothesis did not match the user's intended value; each intended value was also hand-labeled. The authors defined eight prediction tasks, among them, predicting whether the current utterance will cause a communication problem, based on either the current word graph or the six most recent question types. Another task predicted whether the previous user utterance caused a problem, based on the current and previous word graphs, in addition to the question types. The reasoning behind the latter task is based on the fact that a user's correction can often suggest that the previous utterance was misrecognized by the system. This task attained 91.1% accuracy, a significant improvement over the baseline, in predicting that the previous user utterance caused a communication problem.

Walker et al. [83] used supervised learning in a classification task to predict whether a dialogue was going to be problematic. Their intention was to make this prediction early on in the dialogue, based, in part, on features from the first two dialogue exchanges. Predicting a problematic dialogue, the system could connect the user to a customer care agent or modify its dialogue strategies accordingly. A corpus of 4,692 dialogues was collected during an experimental trial of the *How May I Help You?* [34] spoken dialogue system, and was used both for training and for testing the predictor. Both automatic and hand-labeled features were provided to the RIPPER rule induction algorithm, which then learned a predictor. An interesting point of this work is that one of the features used to train the problematic dialogue predictor, *auto-SLU-success*, was automatically predicted, again using a RIPPER model. This feature indicates, for each exchange, whether or not the system correctly understood the user's utterance. The feature had been hand-labeled in previous work, but was predicted using only automatic features in this work. The *auto-SLU-success* predictor obtained 92.4% accuracy on detecting an errorful user utterance, a 29.4% improvement over their baseline of 63%. The problematic dialogue predictor achieved a 13.2% increase in accuracy over the baseline (always predict successful dialogue) of 67.1%.

One of the problems for which we adopt a learning approach in this thesis is the conflict problem, in which the dialogue manager must decide what to do when a newly hypothesized city name conflicts with its current belief. While a reinforcement learning approach could be applied to this problem, it would not be appropriate for our other problem of interest, the compliance problem, in which the dialogue manager must predict whether the user provided a spelling or not. Since only supervised learning was appropriate for the compliance problem, we chose to explore a single method of learning, which could be applied to both problems.

Supervised learning approaches, such as those described above, typically obtain their training data from static corpora of real user dialogues, as opposed to simulated dialogues. The approach in this thesis differs from the typical work in that user simulation is employed to provide data for training a model to address the conflict problem and the compliance problem. Simulation is performed at the spoken level, meaning that both recognition and natural language understanding errors must be handled by the dialogue manager.

1.2.3 User Simulation

Simulation seems to lend itself well to the problem of learning, since a large number of dialogues can be generated without the need for real users. Of course, this would be most useful during the training phase when a great deal of data is required. In order to accurately assess the performance of a learned model, however, the simulated user employed during training should behave as much like a real user as possible. Testing can also be performed with a simulated user; however, again, the simulated user should behave as a real user if an accurate assessment is to be obtained. Mimicking real user behavior throughout a dialogue is no easy task, however. Properties such as persistence, patience, and even goal consistency need to be considered [61].

An assessment of the state-of-the-art approaches to simulated user modeling has been provided by Schatzmann, Georgila, and Young [61]. In their work, the authors emphasize the utility of training a simulated user via supervised learning from an existing corpus of real user dialogues, and then employing that simulated user in a dialogue system, using reinforcement learning to optimize the dialogue policy, a model which was previously applied by Levin, Pieraccini, and Eckert [47]. A benefit of such a simulated user is that a large number of dialogues can be generated and analyzed relatively quickly. It also allows dialogue strategies to be explored that were unseen in the dialogue corpus. The authors summarize the prominent approaches, that aim to stochastically model the behavior of real users. However, they also note that none of the approaches can realistically represent the variety of real user behavior, and that simple statistical tests are adequate to discern simulated from real dialogues.

This thesis employs user simulation for many of the benefits afforded by the technique. The major goal for using a simulated user here is to focus on a specific user behavior or dialogue situation, that may, in fact, be relatively rare in dialogues with real users. Nevertheless, an attempt is made to create simulated utterances that are as realistic and variable as possible, using a novel method of utterance generation. This research benefits from the ability of a simulated user to generate a large number of dialogues, and from the developer's ability to configure the simulated user to behave in a certain way. For example, the developer can guarantee that the simulated user will spell a city name when requested by the user. Many dialogues containing such forced behavior enables the development of a specific dialogue strategy to handle that particular user behavior.

User simulation had not been widely used in dialogue system research until recently; its popularity is now on the rise [22, 23, 85, 47, 63, 64, 65, 51, 15, 28, 58, 19, 29, 30, 39, 61, 62]. When it has been employed, dialogue modeling has typically occurred at the intention level [23, 47, 58, 30]. Such modeling assumes perfect recognition and natural language understanding in the context of a dialogue system. This abstraction enables a focused effort on creating more realistic simulations; however, imperfect parsing or recognition plays a key role in some simulation tasks [51, 29], in which case modeling at the text or speech level, respectively, is desirable.

One of the very first applications of user simulation to facilitate learning in dialogue systems was performed by Levin, Pieraccini, and Eckert [47]. In their work, the authors generated a simulated user model by estimating parameters of real user behavior from a fixed corpus of Air Travel Information Service (ATIS) dialogues. The parameters modeled, for example, the number of attributes provided by a user in response to the system greeting, as well as the likelihood of the user accepting the proposed relaxation for a given attribute. The simulated user was then utilized in a reinforcement learning framework, in which the system explored a finite set of actions from a small set of states. The system's goal was to minimize an objective cost function over a sequence of actions (i.e., dialogue policy) leading to the successful retrieval of information and closing of the dialogue. Such a method would be difficult to perform with real users, since many of the exploratory actions would seem nonsensical to a human. The authors found that the system was able to learn a relatively complex dialogue policy, not unlike those manually specified in similar systems. In their work, however, the authors assumed perfect recognition and natural language understanding, modeling their simulations at the intention level.

More recently, Henderson, Lemon, and Georgila [39] also explored the utility of combining supervised and reinforcement learning with user simulation in the context of fixed corpora of Communicator [81, 84] data. Their approach differs in that they attempt to learn over a very large state space with a set of 70 possible actions. The intention is to maximize a hybrid reward function, which has a supervised learning component based on a dialogue corpus, a reinforcement learning component to model the expected reward for state-action transitions, as well as a penalty parameter for transitions to states unobserved in the data. Adjusting this parameter modified the influence of each component on the final result. A simulated user was trained via supervised learning and was used to test the hybrid model, which performed 37% better than the best-performing Communicator dialogue system. This performance was obtained when the model was completely based on the supervised learning component. They also modeled user simulation at the intention level.

Recent work by Schatzmann et al. [62] questions the current approach of learning and evaluating dialogue strategies with the same simulated user model. They show that a policy learned with a poor user model may appear to perform well when tested on the same model, but performs significantly worse when tested on different, more realistic user models. The authors note that testing with real users is the best possible scenario; however, the costliness of such a task stresses the importance of research to attain user models that are as realistic as possible.

1.2.4 Attribute Value Acquisition

An important issue throughout an SDS session is the accuracy of the system's belief about a user's intended attribute value. While speech recognition has been steadily improving over time, it is still only able to provide best hypotheses of what a user has said, which can be especially misleading when the user's intended value is unknown. Confidence scores and unknown word models can help; however, it is still possible for a completely incorrect hypothesis to have high confidence and vice versa [44], especially when an unknown word sounds similar to a known word. For example, the unknown city, *Chaffee*, may be misrecognized as the known city, *Chelsea*.

This problem is often realized as an interactive process between system and user, in which the system implicitly or explicitly confirms a hypothesized value with the user, who either accepts the value, or denies it and makes another attempt at conveying his or her intended value [85, 44]. In the case that the user's intended value is unknown, and the system has no mechanism for acquiring such a value, the dialogue will futilely continue, until the user hangs up or reluctantly accepts some unintended value.

One problem with confidence thresholds is that they are often based on simple heuristics and they remain static throughout the dialogue. Confidence may be a better or worse indicator depending on the specific user, the dialogue context, and the noise level. One nice approach to alleviating this problem has been elegantly addressed in work by Bohus and Rudnicky [11], in which state-specific rejection thresholds are optimized over the course of a dialogue.

Most systems, given a high enough confidence for an attribute's value, will simply overwrite any existing value for that attribute. This problem does not typically appear in the intention-based simulated dialogues, unless there is a corresponding error model or the possibility for the simulated user to change its mind. When speech is utilized, however, this issue of conflicting hypothesized values is a significant problem. It is also, of course, a significant problem with real user dialogues.

Higashinaka, Sudoh, and Nakano [40] address this problem of conflicting values over the course of a dialogue, which they call *interactive intention recognition*, by encoding the recognized intentions as discourse features and incorporating them into an attribute confidence scoring mechanism that also uses acoustic and language model features. The majority of discourse features are based on the notion of the *slot value sequence*, which represents the transition of beliefs for a given attribute throughout the dialogue. Some example features are the ratio of the current value (i.e., most recent belief) to all values in the sequence, the slot variety, and the number of times the current value occurs successively in the sequence. The authors deployed a weather information dialogue system to collect real user data, from which the relevant features were extracted. Each slot value was also hand-labeled as correct or incorrect. Two confidence models were trained on the 777 samples that remained after screening the data: the baseline model used only acoustic and language model features, while the proposed model added the discourse features. The proposed model performed better with significantly lower false acceptance and rejection rates than the baseline.

Bohus and Rudnicky [12] approach the problem by integrating information across multiple dialogue turns to update and improve the confidence score for the value of a given attribute. The system's belief in a value is only updated following an implicit or explicit confirmation and the corresponding user's response. A total of 449 dialogues were collected from real users via RoomLine [9], a spoken dialogue system for reserving conference rooms. Explicit and implicit confirmations were initiated based on confidence thresholds. Following each explicit confirmation, the confidence was boosted if there was a perceived acceptance of the value, but the hypothesis was deleted in the case of a perceived rejection. Following an implicit confirmation, the hypothesis was deleted if the system perceived a rejection from the user's response. If a new value was hypothesized from the user's response, the old value was overwritten. The confidence was boosted for all other responses. The data was labeled for the type of response following a confirmation, and correctness of the value being confirmed. The machine learning technique of binary logistic regression was then applied to generate a model predicting whether the value undergoing confirmation was correct or not. Results showed significant reductions in error from the heuristic model used in data collection to the learned model. This indicates the success of the belief updating model to more accurately capture the beliefs of the user.

This thesis also addresses the problem of accurately capturing the user's beliefs, but with the difference that our strategies are developed using a simulated user. An additional complication in our work is that there is a prevalence of words unknown to the recognizer. A utility for obtaining spellings from users is explored as a means of acquiring such unknown values. The relatively uncommon spelling-request prompt gives rise to a whole new variety of user responses, which will be discussed in Chapter 8.

1.3 Overview

This thesis aims to provide the reader with an understanding of the entire process of dialogue strategy development. In this chapter, we explained ways in which misrecognitions can arise in a dialogue causing communication problems between a user and an SDS. In the remainder of the thesis, we explore the development of a method to recover from such problems using a novel method of user simulation, and we explore a mechanism allowing the dialogue manager to learn what decision to make in a given problematic situation. Finally, we observe how well the learned models, trained on a simulated user, can perform in dialogue sessions with real users. The chapters are divided accordingly:

- **Chapter 2: System Background**

  In this chapter, we describe MERCURY, the flight reservation system used for our research. We discuss the GALAXY conversational system framework, which was employed for the implementation of the systems in this thesis. We further describe the specific spoken language technologies utilized as part of the framework.
- **Chapter 3: Error Recovery: From Keypad to Speak-and-Spell**

  This chapter contains an analysis of error recovery via the telephone keypad in MERCURY, as well as preliminary experiments in user simulation, which suggest that a spelling mechanism holds potential for recovering from errors in dialogues with real users.
- **Chapter 4: Creative Utterance Generation for Simulation**

  In this chapter, we describe a novel method of utterance generation for user simulation. The method combines synthesized speech with speech segments drawn from real user utterances obtained from preexisting, in-domain corpora to produce a wide variety of realistic user utterances.
- **Chapter 5: Attribute-Centric Generic Dialogue Management**

  This chapter provides a description of a configurable scheme to control both generic dialogue management and simulated user behavior. The mechanism facilitates the quick modification and exploration of various simulation scenarios.

- **Chapter 6: User Simulation: Developing Dialogue Strategies**

  In this chapter, we combine the work conducted in Chapters 4 and 5 in order to explore various configurations of dialogue strategies and simulated user behavior. In doing so, we are able to develop and test an error recovery strategy involving spelling.

- **Chapter 7: User Simulation: Learning When and How to React**

  In this chapter, we describe how user simulation can be used to generate data for the supervised training of decision models. We explore the conflict problem and the compliance problem, both of which require the dialogue manager to make a decision affecting the flow, and potential success, of a dialogue.

- **Chapter 8: Experiment with Real Users**

  This chapter provides a description of a telephone-based spoken dialogue system, accessed by real users, that utilizes the decision models learned from simulated data. We discuss the results from the experiment and provide an analysis of the user responses to the spelling request.

- **Chapter 9: Contributions and Future Work**

  In this chapter, we summarize the thesis research, discuss its contributions to the field, and suggest possible directions for future work.

Chapter 2

System Background

This chapter provides useful background information on the systems and technologies utilized for this research. A description is given of the domain in which the research was conducted. Information is then provided on the framework utilized in the development of the spoken dialogue systems employed. A short description of the function of each human-language technology component in the framework is provided, along with the specific implementation utilized for each component.

2.1 Research Domain

The majority of this research was performed within the scope of the MERCURY flight reservation system [74, 69]. MERCURY is a telephone-based, mixed-initiative spoken dialogue system, accessible via a toll-free telephone number (1-877-MIT-TALK), that provides information about flights to and from over 500 cities worldwide. This set of cities is comprised of major domestic and international cities, but most are concentrated in the United States. Flight information is obtained from Sabre's real flight information service via the Travelocity webpage [1]. Considerable effort has been invested into making MERCURY intuitive to use and robust in handling a wide variety of ways in which users might express their flight constraints and select the flights of their itinerary. A typical user begins by logging in to the system, providing both a username and password, which allows the system to look up some personalized information, including the e-mail address and the preferred departure city. MERCURY's dialogue plan involves arranging a trip one leg at a time. The mixed-initiative nature of the system means that a user can provide any number of attribute values at any time, including departure and arrival city, date, and time, as well as airline preference. Once the itinerary is fully specified, MERCURY offers to price the trip and, subsequently, to send a detailed record of the itinerary to the user via e-mail, which can then be forwarded to a travel agent for the actual booking.

The MERCURY domain is appropriate for our purposes since the class of city names is potentially very large, and it is extremely unlikely than any given system will explicitly represent all possible city names in the recognizer's vocabulary. As shown in the example dialogue on page 21, situations do arise in which users provide city names that are unknown to the recognizer. Such behavior presents opportunities for error recovery via spelling, which this thesis is intended to address.

A spoken dialogue system for accessing weather information, such as JUPITER [91], would also be a good candidate for our purposes since unknown cities are often uttered, especially by users wanting to know the weather in their small hometowns. While we performed some preliminary experiments in the JUPITER domain, we chose to proceed with MERCURY since the dialogues required the acquisition of multiple attribute values, resulting in more complex interactions with more diverse opportunities for error recovery, such as spuriously hypothesized city names and dates.

2.2 GALAXY Architecture

The spoken dialogue systems utilized in this research were built within the GALAXY Communicator [72] architecture, a multimodal conversational system framework, developed by the Spoken Language Systems (SLS) Group of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology (MIT).

A typical GALAXY configuration is centered around a programmable hub, as shown in Figure 2-1, which mediates communication among various human-language tech-

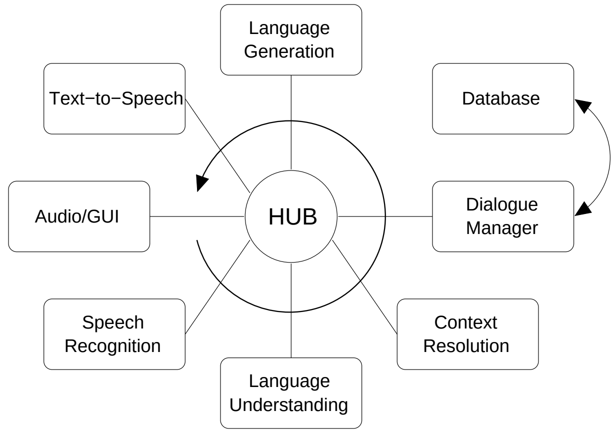

Figure 2-1: The GALAXY architecture.

nology servers. The audio server captures a user's speech and sends it to the speech recognizer [33], which creates a word graph of hypotheses. This information is dispatched to the language understanding server, where the word graph is parsed by TINA [68]. The best hypothesis is encoded in a semantic frame representation and is dispatched to the context resolution server [26, 27], which interprets the user's utterance in the context of the dialogue history and world knowledge. The dialogue manager receives the context-resolved semantic frame and communicates with the database server and language generation server [4, 5] to build an appropriate reply for the user. The system response is then audibly realized by the text-to-speech server [90, 89].

2.2.1 Speech Recognition

A speech recognizer takes human or synthesized speech as input and outputs hypotheses of the spoken words. The output may either be represented as a list of N-best hypotheses or as a word graph of hypotheses. We utilize the SUMMIT speech recognition framework [33, 32], which is a landmark-based recognizer that incorporates four different knowledge sources: acoustic models, phonological rules, lexical constraints, and language models. Further details are provided by Glass [32]. The output of a speech recognizer is then passed on to the natural language understanding component.

```
{c intention
  :topic {q flight
          :pred {p source
                  :topic {q city
                           :*confidence* 4100
                           :name "boston" } }
          :pred {p destination
                  :topic {q city
                           :*confidence* 4500
                           :name "denver" } } } }
```

Figure 2-2: Semantic frame for *I would like a flight from Boston to Denver.*

2.2.2 Natural Language Understanding

The goal of natural language understanding is to extract semantics, or meaning, from text. In our system, the *N*-best list of hypotheses is parsed by TINA [68], which encodes the linguistic knowledge from an utterance as a hierarchical semantic frame, as shown in Figure 2-2. The top-level frame is always labeled as *clause* (c), but subcomponents in the hierarchy can be classified as any of *clause*, *topic* (q), or *predicate* (p), special tags carrying linguistic meaning. Each such entity is also a frame, typically containing additional information represented as keys with associated values. Additional details on the algorithm are available, as reported by Seneff [68]. The resulting semantic frame is then passed to the context resolution server.

2.2.3 Context Resolution

Context resolution, also referred to as *discourse processing*, involves interpreting the meaning of an individual utterance in the context of the dialogue history, physical and temporal context, inference, shared world knowledge, and even common sense. We utilize a context resolution server [27], which performs several specific functions, including resolving anaphora (e.g., "Show me that one."), inheriting implied information from the history, disregarding information that has been overridden, and interpreting elliptical fragments, such as "What about Boston?" An additional responsibility is trying to recover from robust parses by finding relationships between isolated concepts. For example, if the top hypothesis was, "want flight fare," context resolution would establish the relationship, *fare for flight*. The context-resolved semantic frame is then passed to the dialogue manager.

2.2.4 Dialogue Management

The *dialogue manager* (DM) of an SDS is responsible for facilitating the dialogue between the system and a user. Its tasks include preparing a reply frame to encode the system's intentions, maintaining the interaction history of the dialogue, and managing what information is *grounded*, or mutually understood between the system and the user. The *dialogue control* is the part of the DM responsible for using the information gathered by the DM to determine the next step in the dialogue. For example, if a user provides information that is ambiguous, such as *Springfield*, the DM should respond with a clarification question to disambiguate the city name, and/or provide a list of state names from which the user may choose.

The capabilities of the DM determine the potential complexity of the dialogue. For example, one DM may only be able to alert the user that it does not understand a word. Another DM may be able to isolate that unknown word, ask the user for a spelling, and incorporate the word into the recognizer's vocabulary for future use. Maintenance of an explicit user model by the DM could also serve to further increase conversational power.

The strategies to organize dialogue management components in SDSs today are very diverse given that they are often not structured according to any theory of dialogue interaction. A major reason for this is the lack of any theory of spoken human-computer dialogue interaction [8]. There have been guidelines of spoken human-human communication such as Grice's maxims [35]; however, it is not obvious how to apply them in developing design guidelines for human-computer interaction [3]. The maxims assume cooperative conversational partners, but getting the computer and the user to always be cooperative is not an easy task. There are also many plan-recognition based approaches to dialogue management, but they are often criticized in that it is extremely difficult for a machine to recognize the intention of a human's utterance, especially when indirect speech acts come into play [67]. Therefore, most dialogue management components are specifically designed according to the applications they will serve.

There are various ways of implementing dialogue control in an SDS. According to McTear [54], the most common methods include state-based, frame-based, and agent-based. State-based methods, as in the CSLU Toolkit [78], utilize a finite-state automaton to represent valid transitions for every state of a dialogue. Such methods provide a clear representation of the dialogue flow; however, the flow is fixed, and it quickly becomes unmanageable to incorporate all possible transitions into the model. Agent-based methods, such as in the Circuit Fix-It Shop [76], TRAINS [25], and TRIPS [24], are typically logic-based and determine the next dialogue step via proofs. These methods typically involve the identification of user beliefs and intentions, tasks that are often difficult for a computer to perform. Frame-based methods, as in GALAXY [72], fill a frame structure with information gathered from the user, and use rules based on conditions that hold in the frame to trigger various actions. Such methods are often desired for the flexibility in dialogue flow that they support. Each dialogue control method has its own advantages and disadvantages. The type of control that is ultimately used depends on the goal of the application.

The method of dialogue management and control utilized in this thesis is frame-based and builds on the notion of a *dialogue control table* [74, 59]. The control table consists of an ordered list of domain-independent rules which are traversed at run-time. The conditions for a rule are checked against the dialogue state, and a rule is triggered if the conditions hold, thereby determining the next step in the dialogue. Figure 2-3 shows an excerpt from a dialogue control table. The first line, for example, indicates that if there is no `:selection`, `:date`, or `:city` key present in the frame, then the action, `query_for_constraint` will be triggered, and the system will request either a *date* or *city* attribute from the user. More details on the dialogue management for this thesis are provided in Chapter 5.

```
!:selection & !:date & !:city --> query_for_constraint
!:selection & !:num_found | :num_found < 0 --> evaluate_db_query
!:selection & :num_found > 0 & :filter name --> filter_db_tlist
```

Figure 2-3: Excerpt from a dialogue control table.

2.2.5 Natural Language Generation

Once the dialogue manager has determined how to respond to the user, it sends its intentions to the natural language generation server, which will convert those intentions into a natural language sentence, suitable for presentation to the user. We utilize GENESIS-II [4] for language generation, which provides a set of tools allowing a template-based specification of rules. Its flexibility overcomes challenges such as word-sense disambiguation and *wh*-movement. GENESIS-II has been used for the generation of natural languages (e.g., English, Mandarin), as well as formal languages (e.g., SQL, HTML). Once an appropriate natural language sentence has been obtained, it is then passed to a speech synthesizer.

2.2.6 Speech Synthesis

Speech synthesis is utilized in an SDS to audibly realize the response from the dialogue manager. In other contexts, such as the user simulation explored in this thesis, speech synthesis can be used to simulate a user's input for development of dialogue strategies. In this work, we utilize ENVOICE [90], a concatenative speech synthesizer. ENVOICE synthesizes a given utterance by generalizing from existing speech corpora, optimizing for concatenation of the fewest number of available speech segments. Words not present in the corpora can be synthesized at the phone level.

2.3 Specialized Components

As mentioned in Chapter 1, one of the goals of this thesis is to explore the effectiveness of utilizing a spelling strategy to acquire words from the user that are unknown to the recognizer. In order to implement this strategy, a specialized spelling recognizer was activated for the user's response to the system's request for a spelling; the main domain recognizer was not active for the user's response. In preliminary experiments with real users, we discovered that users did not always provide a spelling as requested; rather, they simply repeated the city name or provided some other unanticipated, but in-domain, utterance.

In order to address this issue of *noncompliance*, we implemented a dual-recognition configuration, in which both the main domain recognizer and the specialized spelling recognizer are activated when the system requests a spelling from the user. The dialogue manager will then receive hypotheses from both recognizers and must decide which recognizer to trust. If the user provided a spelling, the hypotheses from the specialized recognizer will be processed. If the dialogue manager predicts that a spelling was not provided, the top hypothesis from the main recognizer will be processed. Further details of this issue will be discussed in Chapters 6 and 7.

The idea of using two recognizers has been proposed in the context of hyperarticulated user speech [41, 44]. It has been found that users tend to hyperarticulate their corrections to a system following misrecognitions. Recognizers are not typically trained on such speech; thus, they perform poorly. Given that a system could predict an utterance as a user correction, the idea is that a recognizer, specially trained on hyperarticulated speech, could then be activated to achieve better accuracy. While this idea has not been implemented, our work differs from the proposed system in that a decision about user compliance is made based on the outputs of both recognizers, rather than first making a decision, then choosing which recognizer to activate.

While it is possible to explicitly represent the alphabet in the main recognizer, this would increase perplexity. Since our research also involves unknown words, the letter hypotheses would have to compete with the unknown word model. In addition, there would likely be a high incidence of spuriously hypothesized letters. Since spelling seems to be a rather isolated occurrence (i.e., in an error recovery subdialogue), and in order to maintain the domain-independent nature of the spelling technique, we chose to implement the dual-recognition strategy.

Figure 2-4 shows the modified GALAXY architecture, which enables the dual-

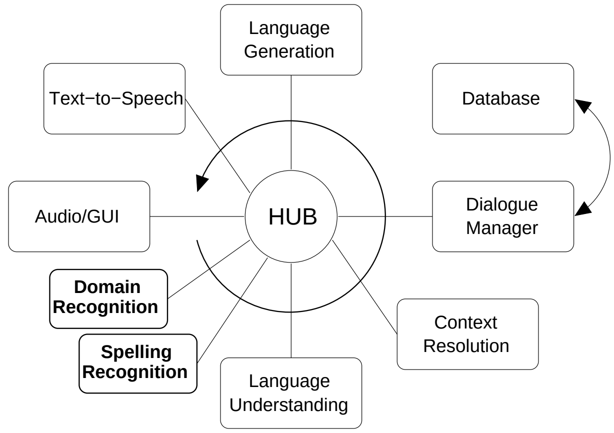

Figure 2-4: Modified GALAXY architecture with dual recognition for error recovery with spelling.

recognizer-based error recovery strategy utilized in this thesis. This architecture is activated following a spelling request from the system. Once the user's response is captured, it is sent to both the main domain recognizer, as well as the specialized spelling recognizer. While a speak-and-spell recognizer was utilized in preliminary experiments and some simulation experiments, a more intuitive spelling-only recognizer was ultimately used in experiments with real users.

2.3.1 Speak-and-Spell Recognizer

The speak-and-spell recognizer was developed by Chung, Seneff, and Wang [16, 70] to enable users to enter their names in an enrollment task. A speak-and-spell utterance consists of a spoken word followed by a corresponding spelling, such as "Chicago C H I C A G O." A two-stage recognition process then ensues. The first pass captures only the spelled portion of the utterance, while the pronounced word is marked as "<unknown>". The second pass then utilizes the pronunciation to constrain the hypothesized spellings. The process relies on ANGIE [73], a hierarchical framework that encodes the subword structure of an utterance using a probability model trained on nearly 100,000 first and last names. This approach enables ANGIE to generalize to unseen words. Further details are available, as reported by Chung, Seneff, and Wang [16].

2.3.2 Spelling Recognizer

Preliminary experiments with real users showed that some users, instead of providing a speak-and-spell utterance, would simply spell their intended city name. Since it seemed that a simple spelling was perhaps more intuitive than speaking and spelling a word, a simplified spell-mode recognizer was utilized in later experiments that only hypothesized letter sequences. The recognizer's language model is based on statistics on a sequence of so-called letter-phoneme units that were determined by parsing a large collection of words into a syllabic substructure organization.

2.4 Summary

This chapter provided information about MERCURY, the spoken dialogue system used in the development of this thesis. The GALAXY conversational system framework was then described. This framework is utilized by the SLS Group in the development of its dialogue systems, and it was used for the systems explored in our research. The general components of a spoken dialogue system were explained, and the specific implementations utilized for this work were described. Finally, a brief description of the specialized spelling recognition servers specifically used for this research was provided.

Chapters 1 and 2 are intended to provide sufficient information to familiarize the reader with the architectures and technologies used in the further development of this thesis. More detailed descriptions of the components relevant to this research will be given in the subsequent chapters.

In the next chapter, an analysis of the original error recovery mechanism used by MERCURY will be explained. It is this analysis that prompted the further exploration of spelling as a means for the system to acquire a user's intended value for an attribute, when that value was unknown to the system's recognizer.

Chapter 3

Error Recovery: From Keypad to Speak-and-Spell

This research focuses, in part, on the two-stage problem of error detection and subsequent recovery, when the user is trying to provide a named entity (e.g., city name) which the system fails to understand. It is not a straightforward process, however, for the system even to notice that it has made a mistake. Tedious confirmation subdialogues for every attribute provided would lead to annoyance and widespread unwillingness to use the system at all. Hence, the system should only invoke a confirmation subdialogue when it perceives there to be a communication breakdown.

Error recovery is also quite a challenging task. The system may persist in misunderstanding repeated spoken renditions of the same named entity, unless a substantially different tactic can be adopted to assure higher communicative success.

This chapter begins with a description of the MERCURY flight reservation system's error detection strategy, as well as its requests for city name entry via telephone keypad for error recovery. This keypad technique was available in the original system design, but it only handled known city names. An analysis of users' reactions to the keypad requests is then presented, indicating the degree to which this strategy was successful. Finally, the exploration of an alternate recovery strategy, speak-and-spell, is described, demonstrating its potential utility for error recovery in an attribute acquisition dialogue.

3.1 Mercury Flight Reservation System

Mercury [69] is a spoken dialogue system providing a user with the ability to search for flights and obtain an itinerary over the telephone. The system provides information about flights for over 500 cities worldwide. More information on the system was provided earlier in Section 2.1.

3.1.1 Error Detection and Recovery

A critical aspect of flight dialogues is the successful communication of the source, destination, and date, all of which are susceptible to recognition error. Mercury's default policy is to use implicit confirmation to communicate its interpretation of a user's utterance to that user. In the meantime, it monitors the evolution of these three critical attributes throughout the dialogue. When it detects odd behavior, it switches into a mode where keypad entry is solicited. The keypad entry is compared to existing hypotheses for the attribute's value and, if a match is found, the corresponding value is assumed to be correct. Otherwise, an explicit confirmation subdialogue, soliciting a "yes/no" answer, is invoked.

For source and destination, the system tabulates at each turn whether the attribute was inherited, repeated, or changed. If a change is detected after flights have already been retrieved, the system prompts for spoken confirmation of the surprising move, anticipating possible recognition error. After two consecutive turns where the user has either apparently repeated or replaced the departure or arrival city, the system requests that the user enter the city by spelling it using the telephone keypad. This strategy is also used if a substitution or repetition of the city is followed by an utterance that is not understood, or whenever the user explicitly requests to change the departure or arrival city. Mercury's 500 cities happen to be uniquely identifiable by their keypad codes; however, if this were not the case, a follow-up disambiguation subdialogue could be arranged. This keypad mechanism also provides the opportunity to confirm whether the desired city is known or unknown.

A similar process takes place for dates. If the user appears to repeat the date,

U1:	I would like to fly to Austin [*Boston*] tomorrow morning.
S1:	Sorry, I'm not sure what you said. There are no flights from Boston to Boston.
U2:	Austin, Texas.
S2:	Okay, flights from Austin to Boston departing between 5:00 am and noon on Monday, March 10th. One moment please... If I have misunderstood you, you should say, "Scratch that." There are no nonstop flights. I have flights on American, Continental, and Delta. There are at least 6 flights departing between 8:00 am and 9:00 am. Can you be more specific?
U3:	I would like to fly from Boston, Massachusetts to Austin, Texas.
S3:	Please enter the departure city using the telephone keypad, ending with the pound key.
U4:	267866# [user enters "Boston" via keypad]
S4:	Okay, from Boston to Austin. What date will you be traveling?
U5:	I would like to leave tomorrow morning.
S5:	Okay, flights from Boston to Austin departing between 5:00 am and noon on Monday, March 10th. One moment please...

Figure 3-1: Example of a dialogue involving a successful entry of a departure city using the telephone keypad, initiated by the system based on perceived confusion. Words in brackets indicate what the system recognized.

without providing any other information, there is the suspicion that a misrecognized date has again been misrecognized the same way. In this case, the system tries to find an alternative hypothesis for the date by reexamining the N-best list of recognizer hypotheses, and, in any case, also asks for user confirmation. As in the case of cities, the system invokes the keypad upon repeated date corrections.

Figures 3-1 and 3-2 provide two example dialogues involving keypad entry of a city. Figure 3-1 illustrates a dialogue where there is clear confusion, and the system eventually takes the initiative to invite a keypad entry of the departure city. The user wanted to go to "Austin", which the system misunderstood as "Boston". This particular user had a default departure city of "Boston", which caused the system to suppose that the user had requested a pragmatically unreasonable flight from "Boston" to "Boston". The user's follow-up fragment, "Austin, Texas", was correctly understood, but misinterpreted as the departure city instead of the arrival city, leading to further confusion. It was only after the user had cleared up the confusion, with the complete utterance, "I would like to fly from Boston, Massachusetts to Austin, Texas," that the system was finally on the right track, but by this point, it had identified difficulty with the source, reacting by launching a keypad entry request, with subsequent resolution.

U1:	I would like to <uh> [*I would like to Atlanta*]	
S1:	Okay, from Hartford to Atlanta. What date will you be traveling?	
U2:	October 31st.	
S2:	Okay, from Hartford to Atlanta on Wednesday, October 31st. Can you provide the approximate departure time or airline?	
U3:	Late morning.	
S3:	Okay, flights from Hartford to Atlanta departing between 10:00 am and noon on Wednesday, October 31st. One moment please... I have a nonstop Delta flight leaving at 11:20 am and arriving at 1:45 pm. Would that work?	
U4:	No.	
S4:	There are no nonstop flights. I have a U.S. Airways flight connecting in Charlotte leaving at 10:45 am, a United flight connecting in Washington Dulles leaving at 11:20 am, and a U.S. Airways flight leaving at noon. Please select one of these flights or change any constraint you have already specified.	
U5:	I would like to change the arrival city.	
S5:	Please enter the destination city using the telephone keypad, ending with the pound key.	
U6:	6752636# [user enters "Orlando" via keypad]	
S6:	I heard you say you want to arrive in Orlando. Is that correct? If you need help, you can say, for example, "What cities do you know in Oregon?".	
U7:	That is correct.	

Figure 3-2: Example of a dialogue involving a successful entry of an arrival city using the telephone keypad, provoked by a specific request by the user to change the arrival city. Words in brackets indicate what the system recognized.

Figure 3-2 shows an example subdialogue where the destination city was successfully entered using the telephone keypad, based on an explicit request on the part of the user to change the arrival city. Interestingly, the user delayed the correction until the system invited him to change any constraint that was already specified. This particular user probably believed that responding to the prompts was required, although it is conceivable that the user's delayed response was due to inattentiveness. This dialogue thus reveals some of the potential difficulties encountered due to users' false assumptions about the system's behavior.

3.1.2 Analysis of Telephone Keypad Responses

Data from MERCURY interactions have been collected for the past several years [74]. In examining these dialogues, we have come to the realization that, while entering the data via telephone keypad (as a four digit numeric code for month and day) seems to be intuitive to users and therefore an effective mechanism for correcting misunderstandings, the situation is far less effective in the case of city names.

A detailed analysis was performed of all instances where the system requested a source or destination entry via the keypad; the users' reactions to the requests were observed and quantified. We found that this strategy, when users were compliant, was generally successful for determining the desired source or destination. For example, if the user were to enter "3387648#", with '#' signaling the final letter, the system would understand "DETROIT", and the dialogue would smoothly continue.

In addition to many successful responses, however, several errorful responses were also observed, including misspelled words (e.g., "TEIPEI" for "TAIPEI"), out-of-vocabulary words (e.g., "DOMINICA"), or a string of valid references that could not be resolved as a single place name (e.g., "NEWYORKJFKNYC" for "New York's Kennedy Airport"). A user time-out or hang-up was also common, and constituted a significant number of responses.

A total of 172 instances were observed in which the system prompted users to enter a source or destination via the keypad. The number of occurrences is rather low since this solicitation was only activated as a last resort. The system then entered a state where speech was not an option. The users' responses to these prompts are summarized in Table 3.1. It was most surprising that nearly half of the time, the user did not even attempt to use the keypad, while a sequence was actively entered in only 88 cases. The user let a time-out occur in 50 cases, and hung up the telephone in an additional 34 cases.

This attempt rate of 51.1% is significantly lower than originally hoped. Even within the 88 compliant cases, the results are disappointing, as shown in Table 3.2. In 61 cases, the keypad sequence entered by the user corresponded to a valid city or airport name. Most of these were known to the system and were processed successfully. The remaining 30.7% of attempts consisted of misspellings (such as a double tap on a key, substituting the number '0' for the letter 'o', terminating with '*' instead of '#'), or apparent garbage.

These results suggest that the strategy of prompting for keypad entry of questionable parameters shows some potential for recovering from situations in which the system is confused about what the user has said. We believe that such recovery

Description	*Count*	*Percentage*
user attempts at keypad entry	88/172	51.1%
time-outs	50/172	29.1%
hang-ups	34/172	19.8%

Table 3.1: Summary of users' responses to 172 system prompts to enter a source or destination using the telephone keypad.

Description	*Count*	*Percentage*
valid city/airport entered	61/88	69.3%
misspelled city/airport entered	19/88	21.6%
garbage entered (e.g, "***#")	8/88	9.1%

Table 3.2: Summary of a total of 88 user attempts at entering a source or destination city or airport name using the telephone keypad after being prompted by the system.

can contribute to successful dialogue completion, as well as to elevating the user's tolerance level. Nevertheless, our results also pose two questions that need to be addressed: 1) why do some users' attempts at keypad entry contain errors, and, more importantly, 2) why do some users not even attempt keypad entry?

It is not possible to know why an individual user failed to enter a valid keypad sequence; no mechanism was available to interview users about their behavior. It can, however, be speculated that the errorful sequences were due to the non-intuitive nature of spelling with a telephone keypad, a user's unfamiliarity with the spelling of a given word, typos, or a user's confusion as to what qualified as an acceptable entry (e.g., Would an abbreviation or nickname be understood?).

It must be noted that what qualifies as a valid keypad sequence depends on the spelling correction capabilities of the system. Even a simple spell checker (not utilized during the MERCURY data collection) could potentially allow the system to make sense of an errorful keypad sequence.

In the case of a time-out, it is difficult to know what each user was thinking while waiting. It is likely that the user was hoping for a return to speech mode after the time-out. The user may have hesitated for fear of sending the system down an even more divergent path. It is also possible that users were inattentive when the system instructed them to terminate with the pound key, and that they therefore entered the entire city, but without a termination code. Clearly a strategic modification to automatically generate a '#' after a significant pause might help reduce this type of error.

The reason for a hang-up is more obvious, given the dialogue context. For example, if the user had repeatedly said that he wanted to fly to Anchorage and the system had already hypothesized three other cities, it is understandable that he would have hung up in frustration.

The telephone keypad would seem to be a very practical mode of information entry given its physical accessibility and limited ambiguity per key. This small set of data in the flight domain, however, suggests that it is confusing, annoying, or simply intimidating to many users. The next challenge, then, was to utilize a similar error recovery strategy, but to adopt a different mode of information entry, one that was more intuitive and less intimidating. Such an option is discussed in the next section.

3.2 Spoken Spelling

Allowing a user to spell a word has several benefits, including maintaining a single mode of communication (i.e., speech), as well as being less taxing, more efficient, and more intuitive. The goal is to make users feel confident that spelling a city name is a plausible request and that it *can* be the most effective path to task completion.

Undeniably, spelling recognition comes with its own set of problems, especially misrecognition of the spoken letters. One way to minimize such errors is to incorporate limited spell checking, such as allowing a single insertion, deletion, or substitution per word. For example, a spelling sequence recognized as "T E N V E R" could be mapped to "D E N V E R" as the closest match in the database. Obviously, a trade-off exists where overgenerous spelling correction could lead to a false hypothesis.

A great challenge in developing conversational systems is that dialogue strategies can only evolve through extensive experimentation, which requires a large amount of data, particularly for situations that occur rarely in actual dialogues. To expedite development and evaluation of the recovery strategy, we decided to make use of

Original user utterance:	No, I will be going from Tokyo to Korea.
N-best list:	<uh> no i_will be going from tokyo to korea
	<um> no i_will be going from tokyo to korea
	<uh> no <uh> i_will be going from tokyo to korea
Hypothesized	<uh> no i_will be going from tokyo to miami
destination cities:	<uh> no i i_will be going from tokyo to korea
Miami	<uh> no i_will be going from tokyo to bermuda
Bermuda	<uh> nope i_will be going from tokyo to korea
	<uh> no will be going from tokyo to korea
	<uh> no no i_will be going from tokyo to korea
	<uh> no i_will be going from tokyo to korea
Original prompt:	Please type the full name of the destination city using the telephone keypad, ending with the pound key.
Original user reply:	73685# (Seoul)
Simulated prompt:	Please speak and spell the destination city.
Simulated user reply:	Seoul, S E O U L.

Figure 3-3: Preliminary user simulation setup, in which a simulated user continues a real user dialogue from the point of error detection, providing a spoken spelling instead of a keypad entry.

simulated user data to artificially continue MERCURY dialogues beyond the point where the system had originally asked for a keypad entry, as described in the next section.

3.3 Preliminary User Simulation

In order to test speak-and-spell as an alternative strategy for error recovery, the first step was to note all instances in which a keypad entry was requested in the original MERCURY corpus. Figure 3-3 contains an example, demonstrating how the context for each speak-and-spell request was obtained. The first row contains the utterance spoken by the original user. The second row contains the *N*-best list generated by the recognizer for that utterance. The third row contains the system's original keypad prompt following the original user's utterance, as well as the original user's response to that prompt. For these experiments, the system continued the dialogue just before this point, by requesting a speak-and-spell utterance instead of a keypad entry. The simulated user then generated a speak-and-spell utterance using DECtalk [37]. The

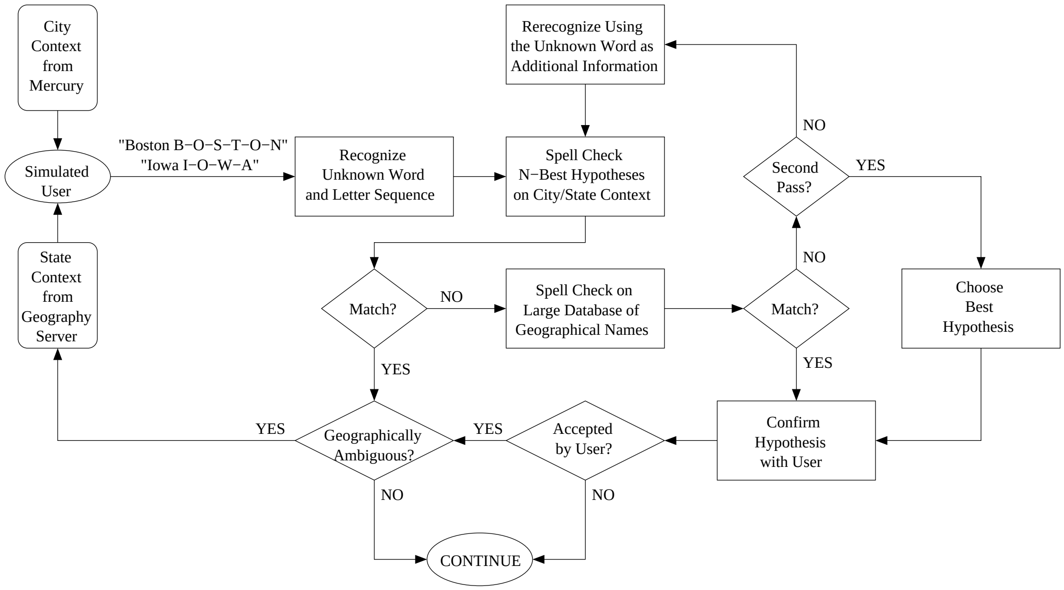

Figure 3-4: Flow chart detailing the two-pass dialogue strategy for recovering from a problematic source or destination.

simulated prompt and reply are shown in the bottom row of Figure 3-3. The letters spoken by DECtalk were not separated by pauses, that is, they were intended to represent a continuous spelling, as is typically produced by a human.

A block diagram of the recovery algorithm is shown in Figure 3-4. Each synthesized waveform[1] contained a pronunciation of the city name that the user was trying to communicate in the original dialogue, immediately followed by a spoken spelling of that city name (e.g., "Boston B O S T O N"). The waveform was passed to a first-stage speech recognizer, which treated the spoken word as an unknown word and proposed an N-best list of hypothesized spellings for the synthesized letter sequence. For speech recognition, we used the SUMMIT framework [33], and the unknown word was modeled according to techniques described by Bazzi and Glass [7].

Following the first-stage recognition, a two-stage matching process first consulted a list of "cities in focus" that were extracted as hypotheses from the original user's final utterance prior to the keypad turn. In Figure 3-3, the cities in focus are *Miami* and *Bermuda*, as shown in the second row. If a match or conservative partial match was not found in this focus list, as in this example, a large database of nearly 20,000

[1]While DECTalk speech is artificial, we have not explicitly trained our recognizer on it, and thus we argue that it can serve as an effective stand-in for real human speech.

city and state names was consulted for a full or partial match. If either type of match was found, a confirmation subdialogue was initiated.

If a match was found, a geography server determined whether the name was ambiguous. If so, a disambiguating item (e.g., state name) was requested by the dialogue manager. The simulated user then randomly chose from a list of candidate state names provided by the geography server. This utterance was also processed as a speak-and-spell utterance, mainly to obtain more data to assess the performance of the speak-and-spell algorithm.

If no match was found in either the focus list or the database of city names, another recognition cycle was initiated, in which the phonetic content of the spoken word was used to enhance the performance of the spelling recognizer, following procedures described by Chung, Seneff, and Wang [16]. A letter-to-sound model was used to map from a graph of letter hypotheses proposed by the first-stage recognizer to their corresponding plausible pronunciations, using techniques described by Seneff, Lau, and Meng [73]. The final set of hypotheses was obtained by merging hypotheses produced from both halves of the user utterance. Once again, both the focus list and the city name database were then searched for a match.

The idea was that the second stage should only be invoked upon failure, in order to reduce the amount of computation time required. An alternative strategy would have been for the system to unconditionally execute a second recognition to obtain a potentially more correct hypothesis. Such a strategy, however, would have increased the system's overall processing time.

3.3.1 Results and Discussion

The simulation was performed on a total of 97 user utterances, all of which MERCURY had designated as trouble situations in the original dialogues. The utterances utilized are those for which the system's *hypotheses* contained city names, whether or not the user had actually mentioned a city name in the utterance.

The simulation results are shown in Table 3.3. Out of 97 problematic sources and destinations generated by the simulated user, 58 required disambiguation with

Description	*Count*	*Percentage*
correct city after pass 1	68/68	100.0%
correct city after pass 2:		
short list match	2/2	100.0%
database match	11/14	78.6%
no match (last resort)	5/13	38.5%
total cities correct	86/97	88.7%

Table 3.3: Simulation results for 97 speak-and-spell city names showing the number of correct cities hypothesized by the system, after each of two recovery passes. For pass 2, a match was found in the short list or in the geographic database. No match resulted in resorting to the best recognition hypothesis.

a state name (e.g., "Boston in Georgia"). Therefore, 155 speak-and-spell utterances were ultimately passed through the synthesize-and-recognize simulation cycle. All but one of the state names were correctly recognized. This high performance was likely due to the correct state's guaranteed existence in the focus list consulted during the algorithm.

Our algorithm dictated that a second pass, integrating the spoken name portion of the waveform with letter-to-sound hypotheses derived from the spoken spelling portion, be omitted if a match was found in the first pass. One question to ask is whether the system was being overconfident in this strategy. The results in the table support the notion of using the second pass sparingly. In 68 cases, the system was sufficiently confident with its hypothesized city after the first recognition pass to omit the second pass; it made no errors in these decisions.

About a third of the time (29 cases), the system, finding no match, initiated a second pass to incorporate pronunciation information. There were two instances where the second-pass hypothesized city was found on the short list of focus cities from the original user utterance; both were correct. For the remainder, the large database was consulted. The system proposed the correct city in nearly 79% of these cases. After failing to find any match, the system attempted its last resort of proposing the best hypothesis from the second-stage recognizer. Not surprisingly, the system determined the correct city name in only 39% of these cases. Nevertheless, this percentage suggests that it may be better to perform the confirmation rather than to simply tell the user that the city is unknown, given that the recognizer may be correct without the aid of any database.

The majority of incorrect city hypotheses were due to limitations in the spell checker and the absence of international names in the geographic database. The spell checker, while quite powerful, allowed only a single insertion, deletion, or substitution of a letter, or a swap of two letters. A more robust spell checker would likely have eliminated many of these errors.

The system's performance in hypothesizing the correct candidate for nearly 89% of the problematic city names was encouraging. These results show that this error recovery strategy was largely successful in the synthesize-and-recognize user simulation cycle. The results were, of course, biased in that the simulated user was cooperative with all system requests. The results of the MERCURY analysis in Section 3.1.2 show that an errorful or nonexistent response from a user is a very likely possibility. The installation of this strategy in a real system would require that user behavior be carefully monitored.

Although the prospects for the speak-and-spell input mode are promising, we would not want to entirely abandon the use of the telephone keypad. It has been and remains a convenient and effective means by which to spell words. A more appropriate use of the keypad could be as a back-off strategy after the spoken spelling has failed, or in very noisy environments, where speech would be nearly impossible. One advantage of the keypad is that, barring mistakes, the system can be confident that when '3' is pushed, one of the letters, 'D', 'E', or 'F', is intended. When combined with the spoken word being spelled, such keypad ambiguity can be reduced even further [16].

3.4 A Live Test of Speak-and-Spell

It should be noted that the simulations just described assumed a completely compliant user who never hesitated or made spelling mistakes. Real dialogues are never so simple, however. In order to test how well the strategy would stand against real users, we incorporated the speak-and-spell strategy into the Jupiter weather information system [91] for a period of four days. There were several instances of successful recovery; however, many types of noncompliance were also revealed. One seemingly natural action was to simply spell a city name without first speaking it. Another very common action was for the user to ignore the speak-and-spell request altogether, and to simply repeat the previous utterance. This noncompliant behavior was likely influenced by prior exposure to a version of the system that did not adopt a spell-mode strategy. When this behavior occurred, only the specialized recognizer was active, resulting in the system's complete failure to understand the query. A resolution to this problem is presented in Chapter 6.

3.5 Summary

Following an analysis of the telephone keypad recovery technique in the MERCURY flight reservation system, it was discovered that users often had difficulty entering their intended city, or they chose not to use the keypad at all. A series of basic user simulation experiments were performed to assess the potential of a speak-and-spell strategy as a more intuitive recovery technique. The results indicated that speak-and-spell could, in fact, serve as a successful recovery technique; however, a brief experiment with real users revealed unanticipated user reactions, suggesting the strategy required further development.

The user simulation technique described in this chapter is very simple; nonetheless, it demonstrated the potential power that user simulation can provide in developing and debugging dialogue strategies. In the following chapter, a much more powerful, and novel, technique for user simulation is described and its application to further develop the speak-and-spell error recovery strategy is explained.

Chapter 4

Creative Utterance Generation for Simulation

Incorporating the speak-and-spell recovery strategy into the JUPITER weather information system revealed some unanticipated user behaviors, as discussed in Section 3.5. The most common such response was when the user did not provide a speak-and-spell response when requested, but rather repeated his or her previous utterance. It was clear that further development of the recovery strategy was necessary.

Although the complete speak-and-spell strategy could conceivably have been permanently incorporated into the JUPITER system, there was a concern that such a move might alienate the pool of devoted users due to potentially annoying system behavior in unanticipated circumstances. It was believed that an extensive stress-testing stage was necessary to reveal many of the possible dialogue situations that could arise due to recognition error or noncompliant user behavior. This testing could arguably be achieved by soliciting paid subjects to participate in various experiments involving carefully defined scenarios. However, this process is very time-consuming and costly, and hence did not seem like a particularly attractive solution.

Since the simulation experiments in Chapter 3 were proven to be useful, it was instead decided to explore an approach involving a simulated user. Such an approach would provide a means to develop and test a specific dialogue strategy, without requiring any input from real users. The MERCURY flight domain is inherently more complex than JUPITER since the former involves the acquisition of source, destination, date, time, and possibly airline name and seating preference. JUPITER only needs to acquire a city name. Therefore, we decided to develop the simulation idea within MERCURY, since it supports more interesting and challenging dialogue scenarios.

An important issue associated with user simulation is determining the medium (e.g., text, speech) and format (e.g., semantic frame, text string) of the simulated user's output. In previous research, the output of simulated users has ranged from strings of text to spoken but read utterances obtained via data collection. Since this research focuses on recovering from recognition errors, the simulated output would have to be a speech waveform. The method of output generation would also have to facilitate testing of specific word classes (such as words unknown to the recognizer), which would allow troublesome dialogue situations to be encountered and analyzed. Ideally, such a method would also not require a data collection effort to obtain real user utterances for simulation.

The remainder of this chapter begins with a brief survey of previous methods utilized for generating output from a simulated user. A novel strategy will then be described for generating a rich set of realistic simulated user utterances through creative recycling of an existing in-domain speech corpus.

4.1 Previous Generation Methods

Some of the earliest work on user simulation was conducted by Eckert, Levin, and Pieraccini [23]. The main goal in this work was to provide a means to automatically evaluate a dialogue system, and to compare the performance among multiple dialogue systems, in an "unbiased, accurate, and inexpensive" way. In these simulations, the simulated user's intentions were communicated to the system on a semantic level, as attribute-value pairs; perfect recognition and understanding were assumed. They also explored the communication of intentions on the syntactic level, as the simulated user generated template-based word sequences, passing them to an understanding component, which then sent hypotheses to the dialogue manager. The authors explained that simulation at the spoken level would be beneficial, but not until other questions relevant to their research were answered.

Watanabe, Araki, and Doshita [85] used the notion of user simulation to explore the effectiveness of various strategies of explicitly and implicitly confirming values in a meeting scheduling task. The mode of communication is natural language in the form of Japanese text. Recognition errors are simulated by deleting words from the original utterance or swapping the intended value with an incorrect value from the same semantic class. The authors do note that further examination is necessary due to the simulated nature of the dialogues, which do not accurately model properties of real user behavior, including the introduction of words unknown to the system.

User simulation was again used by Levin, Pieraccini, and Eckert [47], but for the purpose of learning optimal dialogue strategies via reinforcement learning. The simulated user produced a semantic representation of its intentions in template form which, again, assumed perfect recognition and understanding.

Scheffler and Young [63, 64] explored user simulation in two domains, an on-line banking system and a movie guide. They also represented the simulated user's queries only at the intention level. Before each dialogue began, the simulated user was assigned a set of goals (i.e., a scenario to solve), and its decisions about what to say next were conditioned on a joint consideration of the user's goals and the system's prior responses. Furthermore, a real user corpus was used both to define the scenarios and to provide a probability model of simulated recognition errors applied to the specified attributes. In the movie domain, they compared time to completion for each goal with those of the real user dialogues, showing a strong correlation, although the real user dialogues took consistently longer on average.

In subsequent work on user simulation, Lopez et al. employed speech for the simulated user's utterances [51] in order to compare the performance of two different recognizers and two different strategies for handling confirmation (implicit and explicit). The authors obtained their speech for simulation by recording customers while they were ordering in a fast food restaurant. They then selected a subset of those utterances to obtain wide phonetic and semantic coverage. The transcriptions of these utterances were subsequently read aloud and recorded in a quiet environment by nine subjects to produce the corpus of spoken utterances ultimately used by the simulated user. While the syntax of the simulated utterances was based on real user speech, the utterances were reproductions and, therefore, did not represent naturally spoken intentions. They did, nevertheless, exercise both the understanding and recognition components throughout the dialogue.

Another method of utterance generation was explored by Chung in the context of running simulations to obtain data for training both the recognizer and understanding components of a restaurant information domain [15]. In this work, the simulated user produced a semantic frame of its intentions. This frame was sent to a natural language generator, which output a text string based on a set of templates devised by the author. One set of simulations was performed with text strings; another was performed with spoken versions of those strings, synthesized from segments of real user speech, enabling an assessment of the system's robustness to recognition errors in the simulated dialogues.

4.2 Novel Generation from Existing Corpora

In the novel method of utterance generation introduced here, spoken utterances are utilized in the simulations. In order to produce a wide variety of utterances, speech segments of real users from an existing corpus of in-domain utterances are combined with segments of speech synthesized by DECtalk. This approach allows the utilization of existing dialogue corpora, without the restriction of having to use the utterances in their original contexts. This method is beneficial in three ways:

1. it models the variety of ways that real users with similar goals pose questions and statements to convey their intentions,
2. it enables the simulated user to respond to the dialogue manager with utterances absent from the corpus (e.g., speak-and-spell utterances), and
3. it facilitates the controlled exploration of specific classes of words via splicing (e.g., foreign or unknown city names).

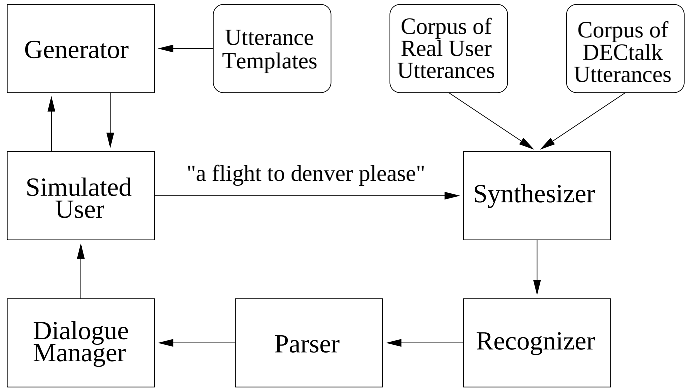

Figure 4-1: Flow within the dialogue system during a typical dialogue turn with the simulated user.

The generated utterances will then be sent through the entire SDS cycle. This step naturally introduces errors that may come about through both recognition and natural language understanding, so the simulated utterances of the dialogue are, in fact, close approximations to real user utterances.

A diagram of this creative utterance generation process is shown in Figure 4-1. The process begins with the simulated user, which randomly selects a value for each required attribute. These attribute/value pairs, or beliefs, will be maintained for the entire dialogue. The developer specifies whether each of these attributes *must* be spoken, *may* be spoken, or must *not* be spoken in the user's initial utterance. For example, in one set of experiments, the user's initial utterance contained only *source* and *destination*.

The simulated user sends this constrained set of beliefs via an e-form (electronic form) frame to a generator, which consults a database of utterance templates, selects a template according to a set of developer-specified rules, and splices in the values provided in the e-form. The generator outputs a string, just one possible surface representation of the beliefs held by the simulated user. This string is then passed to the synthesizer to produce a synthetic speech waveform, which is then passed to the speech recognizer as if it were a real spoken utterance.

"on January the twenty second"
"Wednesday twenty two November"
"a week from tomorrow"
"four days from now"
"on the day before Thanksgiving"
"on the last Saturday in July"

Figure 4-2: Examples of date patterns extracted from a corpus of MERCURY utterances.

4.2.1 Database of Utterance Templates

The database of utterance templates was created by parsing nearly 16,800 unique transcriptions of previously recorded MERCURY utterances. The result of the parse is the same set of sentences; however, the members of a given class are replaced by the corresponding class name. For example, the utterance:

```
flights to boston on may third
```

would be translated to:

```
flights to <us_city> <month_date>
```

This process produced a corpus of nearly 10,400 unique utterance templates. Simultaneously, all members of a given class were extracted, producing a set of diverse patterns to instantiate that class name in the utterance templates. This is of particular importance in modeling the variety of ways a user can express a date. Representative examples of extracted date patterns are shown in Figure 4-2.

An example database entry is shown in Figure 4-3. Each entry is associated with a list of utterance templates (":sentences"). The generator locates the set of database entries that match the attributes in the e-form provided by the simulated user. A template is then randomly chosen, according to a uniform distribution over the set.

A simple example should help to explain the generation of a simulated user utterance. Assume the simulator produces the following e-form:

{c eform
    :source "detroit"
    :destination "denver" }

```
{c eform
   :clause "statement"
   :source "<us_city0>"
   :destination "<us_city1>"
   :sentences (
               "from <us_city0> i would like to go to <us_city1>"
               "hi i would like to go from <us_city0> to <us_city1>"
               "i am flying from <us_city0> to <us_city1>"
               "i am interested in flights from <us_city0> to <us_city1>"
               "i am leaving from <us_city0> to go to <us_city1>"
               ...)
}
```

Figure 4-3: An entry from the database of utterance templates. Each unique e-form is associated with a set of corresponding sentences. This e-form contains a total of 58 unique sentence templates.

The system consults the corpus of e-forms, like the one in Figure 4-3, gathering all that match the specified source/destination constraints. It then chooses a template using a uniform distribution over the full set of *templates* contained in all matching e-forms. A header file specifies match constraints. For example, we can allow any value for ":clause" (e.g., "exist", "statement", "wh-query") while rejecting any e-forms that contain meaningful additional attributes such as ":departure_time" or ":airline".

If the third template were chosen from the set in Figure 4-3, replacing the class names with the values from the simulated user's e-form would produce this string:

```
i am flying from detroit to denver
```

which would then be passed to the synthesizer.

4.2.2 Concatenative Speech Synthesis

The simulation strategy presented here employs the Envoice [90] concatenative speech synthesizer. Envoice synthesizes a given utterance by generalizing from existing speech corpora, optimizing for concatenation of the fewest number of available speech segments. If a word does not exist in the corpus, Envoice will concatenate subword units, down to the phone level, given that it has a pronunciation of the word.

While the corpus is not likely to contain a requested utterance in its entirety, the

Method	Original	DECtalk	Envoice	Env&DEC
WER	12.0	8.5	13.5	13.6
SER	24.7	22.6	28.8	28.6

Table 4.1: Recognition results for a test set of utterances, realized as speech via four different methods: original user waveform, DECtalk synthesis, Envoice synthesis, and a combination of the two synthesizers (Env&DEC).

synthetic waveforms are nonetheless composed of real user speech segments, and thus model the variety of syntactic forms and phonetic realizations produced by real users. Many waveforms will contain segments from multiple speakers, but this should not disturb recognition since our recognition algorithm has no speaker-dependent aspects.

In order to validate the use of such synthesized utterances in a simulated dialogue system, an experiment was performed to compare the recognition accuracy on a standard test set, realized as speech using four methods:

1. the original waveform as spoken by a real user
2. a DECtalk synthesis of the waveform transcript
3. an Envoice synthesis, in which words not in the Envoice corpus were generated from subword units
4. an Envoice synthesis, in which missing words were generated by DECtalk.

The test set consisted of 1,451 utterances, with a vocabulary of 905 words. The corpus was built from 3,292 real user utterances from the MERCURY domain. A total of 89 test set words were absent from the Envoice corpus and, consequently, had to be generated by concatenating subword units or by employing DECtalk synthesis.

Results are shown in Table 4.1. The set of original waveforms showed a word error rate (WER) of 12%, while the DECtalk set performed significantly better with a WER of only 8.5%. The Envoice sets both showed about a 13% relative degradation in WER from that of the original waveforms.

The intent of this experiment was to demonstrate that the concatenative generation of spoken utterances could lead to recognition performance comparable to that of original single-speaker waveforms. The performance of Envoice is sufficient for the purpose of simulated dialogues, and the additional degradation may even be useful for increasing the need for error recovery mechanisms. Furthermore, because our simulations choose sentence patterns that are evenly distributed over all observations, rather than biased towards the more common ones, the simulated utterances are likely to be further degraded in recognition performance, relative to real user utterances. This may reduce the success rate, but should provide a richer repertoire of failure patterns to stimulate system development.

4.3 Summary

The experiments described in Chapter 3 showed that user simulation could be an appropriate technique for developing dialogue strategies in a spoken dialogue system. Simulation enables batch processing of many dialogues, eliminating the need for real users in the initial stages of development. This chapter introduced a novel method of creative utterance generation via templates for user simulation. Such a method would aim to model the variability of real user utterances, and would also enable the splicing of synthesized words into the templates, facilitating an assessment of the system in handling very specific word classes, such as unknown city names. It was also shown how this method results in recognition performance comparable to that of real users. Any additional errors would only provide further opportunities for the development of our dialogue strategies. In Chapters 6 and 7, this novel method of utterance generation is utilized by a simulated user in a full spoken dialogue system, producing realistic, spoken utterances.

One of the benefits of user simulation is that a variety of different dialogue strategies can be assessed rather quickly. In order to facilitate such evaluation, however, configuration of the dialogue manager's policies must be relatively simple and, ideally, require no changes to the actual code. The next chapter describes the generic dialogue management framework utilized for this research. Properties of a number of dialogue strategies, on a per-attribute basis, are specified in an external file that is loaded by the system on startup. Such an approach allows a variety of different dialogue strategies to be modified and assessed in an efficient manner.

Chapter 5

Attribute-Centric Generic Dialogue Management

In the previous chapter, we presented a novel method of spoken utterance generation for a simulated user. The production of realistic utterances would enable the development of dialogue strategies to address certain types of user behavior. In the application of user simulation to spoken dialogue systems, simple modification of the dialogue manager's strategies is critical so that multiple policies can be quickly configured and assessed. This chapter describes a relatively simple, but powerful method of generic dialogue management, in which a developer can specify which strategies are to be used during a dialogue interaction, as well as for which specific attributes those strategies should be utilized.

As described in Section 2.2.4, the dialogue manager (DM) is the component of a spoken dialogue system (SDS) that determines which strategies to follow to ensure that a user's goal is satisfied. In order to achieve this task, however, the DM typically has to know a great deal about the specific application domain. For example, in a flight reservation domain, the DM must be aware that it needs to obtain source, destination, and date before accessing the database. In a restaurant domain, the DM should know that if no cheap Italian restaurants are available in an area, it would be helpful to inform the user if there are possibly any moderately priced ones.

A single SDS can be very powerful, but it often requires a great deal of time and expertise to build and tune. Due to the heavy reliance of the DM on the details of the application domain, the instantiation of an SDS for a new domain is often delayed by the development of the new DM. The ability to satisfy a user's goal depends on knowing what specific attributes to ask for, and which of many knowledge sources need to be accessed. Therefore, a completely separate DM is often created for each new domain, containing tweaked copies of the cross-domain functionality, as well as any complementary functionality specific to that domain.

Studies focusing on a best practice approach [21] to SDSs indicate that emphasis should be placed on dialogue strategies that are domain-independent, rather than being biased towards a particular system or task. Such strategies may include salutations, apologies, requests for repetition, disambiguation, and context-dependent help. The current trend in dialogue modeling has indeed been to abstract these shared functions out of the DM so that they can be used in a domain-independent manner. Several systems have adhered to this approach in recent years [78, 2, 55, 59, 10, 13, 14]. They all aim to facilitate the portability of an SDS from one domain to the next.

Although the designers of generic dialogue managers use experience and common sense to devise appropriate dialogue strategies, the only way to assess the effectiveness of a strategy is through extensive experimentation [31]. This observation means that the ability to modify the system's dialogue strategies through simplified mechanisms is crucial. Usability of an SDS for a developer (i.e., the one who tunes and deploys a system) is often overlooked because it is the system designers who implement the development interface. Even if a developer is able to easily modify the properties of dialogue strategies for an SDS, the extent to which he or she may do so is minimal.

5.1 Overview of the Model

The goal of this attribute-centric model of generic dialogue management is to facilitate usability for the developers of an SDS, so they may explore and evaluate a wide range of strategic configurations. An *attribute* is defined here as any concept for which a user may provide a value, for example, *destination*, *age*, or *birthplace*. An attribute model exists so that a system's interaction properties may be adjusted for the acquisition and handling of each attribute. A set of generic dialogue strategies serves to provide simple or complex conversational functions across all domains, while a set of expert components, dealing with geography, dates, or time, provides sources of common information for any domain requiring this knowledge.

This generic dialogue management model is intended mainly for database access applications, in which a set of attribute values must be acquired by the system in order to provide some information to a user. Consequently, some relevant types of generic strategies might include implicit and explicit confirmation of a value, selection from a list of values, and spelling the value via the telephone keypad or via speech, as described in Chapter 3.

5.2 Benefits of the Model

According to the framework that was developed as part of this thesis research, the developer specifies properties for the acquisition and handling of each attribute in an external file, which facilitates experimentation among various configurations of dialogue strategies. For example, by simply specifying a property on the interface, a developer could instantly incorporate a technique, such as speak-and-spell error recovery technology, into a domain's dialogue strategies for acquiring the value of a particular attribute. Similarly, if the system is having difficulty acquiring a value via a user's spoken spelling, it could back off to request a spelling via the telephone keypad. The power of such a model lies not only in the specific functions that can be performed at the level of a single attribute, but also in the fact that dialogue control *can* be specified at the detail of each individual attribute.

5.3 Model Components

In this model, exceptional emphasis is placed on the attributes of a domain and the specification of properties to affect the acquisition, processing, and error handling of each individual attribute. The intention is to provide rapid prototyping of a new domain's dialogue strategies, as well as rapid modification and evaluation of strategies for an existing domain. This method also facilitates the exploration of multiple configurations in a user simulation framework.

5.3.1 Attribute Model

Previously reported models have not focused on the properties of *acquiring* individual attributes. Some models do, however, have the potential to specify attribute properties in order to affect the interaction. The Agenda-based architecture [88] from Carnegie Mellon University features a separate *handler* for the acquisition and processing of a single information item (e.g., destination), but each handler only recognizes relevant values in the user's input (e.g., to Miami) and produces an appropriate response. The handler does not affect, for example, *how* the attribute is acquired.

In RavenClaw [10], the successor to the Agent-based architecture above, the Dialog Task Specification is represented by a tree. The terminal nodes provide fundamental functions such as *AskRegistered* or *AskName* in a Login subdialogue, which could potentially be tuned to the specific attribute being requested.

The attribute model proposed here consists of a set of properties for each attribute of the domain. Some example properties include:

- is the attribute obligatory?
- type of confirmation: explicit, implicit, none
- back off to speak-and-spell mode?
- back off to telephone keypad mode?
- include in response/summary?

The attribute model serves to inform the dialogue flow. A domain-independent outline of the dialogue flow can be made domain-*dependent* by consulting the properties in the attribute model. The developer specifies, in a descriptive manner, the properties associated with each attribute and the dialogue strategies that should be applied for that attribute. Such properties designate, for example, what attributes are obligatory and must be requested by a system, as well as what type of error handling to use when a city name cannot be understood.

There is, of course, the argument that a computer should decide what conversational strategies to apply to individual attributes based on what it has learned [49]. The specification of attribute properties, after all, produces more work for the developer. Learning certainly has tremendous benefits; however, in a development scenario, the ability to force specific behavior is essential in debugging or exploring specific dialogue strategies.

Figure 5-1 shows a portion of an example attribute model specified in an external file. The model contains a set of *:handlers*, in which default properties are specified for general semantic concepts. In the example, properties are defined for *city*, *state*, and *date*. The model also defines a set of *:attributes* for the specific domain. In this example for the flight reservation domain, the essential attributes are *source*, *destination*, and *date*. Each attribute is defined by one or more handlers. For example, the *source* attribute is defined by a *city* handler and a *state* handler. The property values specified in the top-level *:handlers* section apply to all attributes supporting that specific handler. A default property value may be overridden by defining that property within the handler under a specific attribute. For example, the *spell_reply* property under the default *city* handler is the prompt that the system should present to the user when requesting the spelling of a city name. The default value is *"Please spell the city name."* In the flight domain, both *source* and *destination* support the *city* handler, so the prompt should differ for a *source* city and a *destination* city. More specific values may be provided for *spell_reply* under the handler for each of these attributes. For a source city, the prompt is thus, *"Please spell your departure city name."*, while the prompt for a destination city is, *"Please spell your arrival city name."*

A variety of dialogue interaction properties can be specified on a per-attribute basis in the attribute model. For example, *sas_list_confirm* under the default *city* handler specifies that if a user's spelling matches a city name in an off-line database of city names, the system will explicitly request confirmation of that city. The *oov_confirm*

```
:handlers (
  {c city
    :sas_hyps_confirm "implicit"
    :sas_list_confirm "explicit"
    :oov_name "$dyncityoov"
    :oov_confirm "spell"
    :confidence "thresholds_A"
    :spell_reply "Please spell the city name."
    :confirmation_reply "Okay, did you say @?" }
  {c state
    :need_reply "Can you provide the state name?" }
  {c date
    :key ":date"
    :confidence "thresholds_B"
    :need_reply ( "Can you provide a date?"
                  "Can you try saying the date again?" ) }
)
:attributes (
  {c source
    :handlers ( {c city
                  :key ":source"
                  :need_reply "What city will you be departing from?"
                  :spell_reply "Please spell your departure city name." }
                {c state
                  :key ":source_state" } )
  }
  {c destination
    :handlers ( {c city
                  :key ":destination"
                  :need_reply "What city will you be traveling to?"
                  :spell_reply "Please spell your arrival city name." }
                {c state
                  :key ":dest_state" } )
  }
  {c date
    :handlers ( {c date} ) }
)
:confidence (
  {c thresholds_A
    :values ( "_    1500 spell"
              "1501 3999 explicit"
              "4000 _    implicit" )
  }
  {c thresholds_B
    :values ( "none explicit" )
  }
)
```

Figure 5-1: Excerpt from an example attribute model specified in an external file.

property specifies that if an unknown word is hypothesized for a city, the system should request a spelling for the user's intended value. The value of the *confidence* property links to a list of threshold spectra at the bottom of Figure 5-1. For a city, its confidence is compared against *thresholds_A* and the associated action is performed. For example, a city confidence of 4000 or greater will cause the system to implicitly confirm the hypothesized value. A date, on the other hand, does not have a confidence associated with it, so the system acts according to *thresholds_B*, which says to explicitly confirm a value if no confidence is present. It can be seen that the *date* attribute does not override any default properties of the *date* handler, meaning that all of its property values come from the default handler.

Other useful tools include the inclusion of '@' in a string value, which is replaced by the value of the attribute in focus. For example, the value for *confirmation_reply* under the default *city* handler is *"Okay, did you say @?"*. The '@' will be replaced dynamically with the value of the relevant attribute, producing, for example, "Okay, did you say *Detroit*?".

Another useful feature is the specification of a list of prompts, through which the system may cycle so as to avoid constantly repeating itself. For example, the *need_reply* property under the default *date* handler contains a list of prompts. The system will cycle through this list each time a value for the date attribute is requested.

Figure 5-2 shows a dialogue that could result if the system were to act according to the example attribute model. In S(1), the system explicitly asks the user for confirmation of the date, *July 10th*, which is triggered due to the lack of a confidence associated with the date. In S(2), the system implicitly confirms the source and destination cities (due to high confidence scores), as well as the date provided by the user. Having obtained all required attributes, the system checks the database for items matching these three constraints, and suppose 20 items are returned. Another property might indicate that an additional attribute should be solicited if greater than ten items are returned from the database query. Therefore, in the second part of S(2), the system solicits airline, departure time, or arrival time to narrow down the results. In U(3), the user corrects the system's hypothesized destination. In order to recover from its error, the system requests that the user spell the arrival city, due to a low confidence score. In S(4), the system explicitly confirms the destination as part of the error recovery process. In S(5), the system is back on track and implicitly confirms the source, destination, and date, as well as solicits further information to narrow down the database results.

The attribute model enables a system developer to modify a variety of interaction properties affecting, for example, attribute acquisition and system prompts on a per-attribute basis. The properties may be specified in a domain-independent manner, separate from the general dialogue management functionality. In addition, the abstraction of properties into an external file facilitates quick modification of a dialogue manager's strategies, which can then be assessed by employing a user simulation framework.

The utility of this externally specified attribute model prompted us to use it to control our simulated user. A variety of properties could easily be specified in an external file, again, facilitating the quick modification of the user's behavior. The ability to quickly modify the dialogue manager's strategies and the user's behavior would allow us to more easily experiment with various configurations of the user simulation framework. This application will be discussed further in Chapter 6.

This approach enables a system to stress the importance of certain attributes. For example, in the weather domain, it is crucial for the system to obtain the correct value for the city attribute. Likewise, in the flight reservation domain, the source and destination are attributes that the system must obtain confidently. One very powerful tool would be a dialogue strategy that focused on obtaining the value for a given attribute (e.g., city name) until the system was very confident of that value. For example, if the system were not confident on the first pass, it could ask the user to repeat. If still unconfident, the system could request the user to speak and spell the word, followed by a request for a telephone keypad entry of the word's spelling. Such a strategy would be warranted for an important domain attribute, whereas it would be overkill to use this strategy to determine whether the user preferred an aisle or a window seat.

U(1):	I want to fly from Boston to Dulles on July 10th.
S(1):	*Did you say July 10th?*
U(2):	Yes.
S(2):	*Okay, from Boston to Dallas on July 10th. Can you provide an airline or a departure or arrival time?*
U(3):	No, not Dallas—*Dulles.*
S(3):	*Please spell your arrival city name.*
U(4):	Dulles D U L L E S.
S(4):	*Do you mean Dulles Airport?*
U(5):	Yes.
S(5):	*Okay, from Boston to Dulles Airport on July 10th. Can you provide an airline or a departure or arrival time?*

Figure 5-2: Dialogue between a user (U) and a flight reservation SDS (S).

5.3.2 Dialogue Strategies

A set of generic dialogue strategies, or conversational skills, is what facilitates portability from one SDS domain to another. The purpose of the generic dialogue strategies is to prevent the need to repeat functionality across domains, as well as to maintain consistency in conversational skills that are common from one domain to the next.

The dialogue strategies supported by different generic dialogue systems are various. RavenClaw [10] supports confirmation, disambiguation, requests for help and repetition, suspending and resuming the dialogue, starting over, and reestablishing the context of an interrupted dialogue. The Rapid Dialogue Prototyping Methodology [14] supports confirmation, requests for help and repetition, *dead-end* management, in which the system recovers when no solution is available to a user's query, timely dialogue termination to avoid extraneous interaction with the user, and incoherence handling, in which the user has provided incompatible values for a single attribute.

In this work, more technology-dependent strategies such as requesting speak-and-spell entry or telephone keypad entry of a problematic word have been explored.

The immediate availability of domain-independent dialogue strategies, no matter how simple, is very powerful, and facilitates the rapid prototyping of the DM for a new domain.

More complex generic strategies, however, can enrich the dialogue even further. The current implementation supports the most basic of generic dialogue strategies, including the following:

- attribute clarification
- implicit attribute confirmation
- explicit attribute confirmation
- request keypad entry of attribute

The current implementation also supports the more complex strategies of requesting a spelling or a speak-and-spell utterance of an attribute's value when the system is not confident about what the user said [28]. The ability to simply "push a button" and immediately have an error recovery subdialogue in place is extremely powerful, and shields the developer from the underlying system complexity. Removing that subdialogue would be as easy as pushing another button. Such an ability can be very useful for a system developer in the context of user simulation, when the performance of a spelling or keypad strategy must be assessed. Enabling such exploration of various dialogue strategies for individual attributes is a major goal of this thesis.

5.3.3 Expert Components

A set of expert components, each responsible for maintaining a specific body of common knowledge, complements the generic dialogue management model. A single component may contain the knowledge for areas such as geography, dates, or time, and any functions associated with such processing. Previous work in the SLS Group has provided a date and time server for GALAXY which converts relative dates such as *the following Friday* and *the day before Thanksgiving* into absolute dates [59]. The server's functionality is domain-independent and provides a service that is useful across many domains.

A useful ancillary contribution from this thesis research is the development of a domain-independent geography server, which acts as an interface between a DM and several geographic databases, containing information such as city, state, latitude, longitude, and population size. The server provides functionality such as checking the validity of certain city/state combinations, finding the nearest city to some given city, translating between U.S. state names and abbreviations, and checking for place name ambiguity. This functionality previously allowed the increase in the number of cities for which JUPITER can provide weather from 500 to 38,000. The geographic server can be used as a plug-in module and is available to any domain.

5.4 Summary

This chapter presented an attribute-centric model of generic dialogue management, which focuses on the acquisition properties of individual attributes. Central to the framework is an attribute model, in which various properties relevant to a specific attribute are specified. These properties include the system prompts produced when requesting an attribute value, the type of confirmation to initiate (e.g., explicit, implicit) when an attribute value is suspicious, and the type of error recovery (e.g., spelling request, keypad request) that should be employed when the system has trouble recognizing an attribute value. The remainder of the model is comprised of a set of generic dialogue strategies and a collection of expert components to handle specific bodies of knowledge, useful across multiple domains.

The ability of a developer to rapidly configure the dialogue manager's strategies is very important for user simulations, since a variety of dialogue policies can be quickly configured and assessed. The next chapter describes how the generic dialogue management framework was utilized in user simulations, along with the creative utterance generation technique from Chapter 4, to develop and assess the speak-and-spell error recovery strategy. A variety of configurations of both the dialogue manager and the simulated user are explored.

Chapter 6

User Simulation: Developing Dialogue Strategies

In Chapter 3, we described how the deployment of the speak-and-spell (SAS) strategy in the JUPITER system revealed unanticipated user responses, the most common being when the user did not provide an SAS response when requested. It was clear that further development of the strategy was essential.

In an effort to improve upon the initial, simplistic implementation of the SAS strategy, which did not account for such user behavior, we decided to configure a system implementation that would run the two recognizers in parallel when the system requested an SAS utterance. The N-best lists from both the main recognizer and the SAS recognizer would be passed to the dialogue manager, which would then be responsible for deciding which recognizer to trust.

In order to further develop the strategy, we decided to explore an approach that involved simulating user behavior. In Chapter 4, we introduced a novel method of utterance generation for user simulation, that would allow us to test a variety of dialogue strategies against more realistic, and potentially problematic, user input.

As mentioned previously, since MERCURY is inherently more complex than JUPITER, supporting more interesting and challenging dialogue scenarios, the simulation idea was developed within MERCURY. In order to accommodate the simulated user, the GALAXY configuration was modified, as shown in Figure 6-1. The simulated

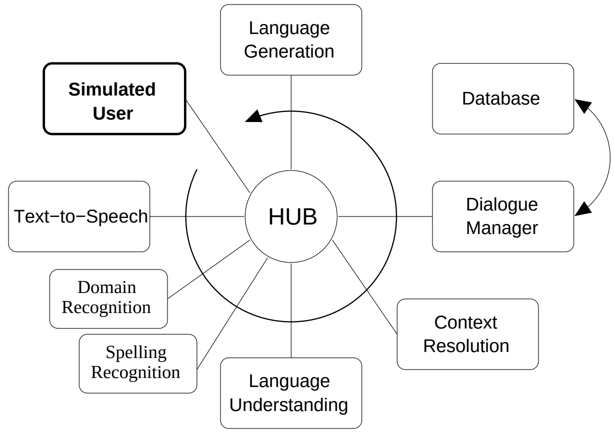

Figure 6-1: The GALAXY user simulation architecture.

user replaces the real user in an otherwise identical GALAXY configuration. The audio/telephony server is replaced by the simulated user server, which acts based on the system's response and calls upon the language generation and speech synthesis servers to convert its intentions into a speech waveform.

As mentioned in Chapter 5, the simulated user is configurable by the developer, so that various behaviors may be adjusted and evaluated. For example, the user may initially speak any number of required attributes (e.g., *source*, *destination*, *date*) for the domain, or the user may be noncompliant with an SAS request; that is, it will not respond with an SAS utterance when prompted to do so. Additionally, a configurable dialogue manager is employed, which can be set up to vary the type of error recovery request initiated based on various triggers, such as a low confidence score, or the number of SAS recovery attempts delivered before giving up on a given attribute.

The complete simulation framework would provide a means to develop and test a specific dialogue strategy, in which the behavior of the simulated user could be controlled as desired, and no input from real users would be required.

In the remainder of this chapter, a set of experiments is presented that involves various configurations, in which a simulated user attempted to convey its intentions to the system. Results are provided as to how well the system performed in acquiring the correct values for the specific attributes. The results also reveal how well certain dialogue strategies were able to handle specific user behaviors.

6.1 Dialogues with a Simulated User

The replacement of a real with a simulated user enables the batch processing of many dialogues. The generation and analysis of a large number of dialogues while using a specific dialogue strategy allows us to debug the system within a tightly controlled framework, as well as measure the efficiency or success of the given strategy.

As mentioned earlier, the dialogue manager (DM) of a dialogue system is responsible for determining which action the system should take next, depending on the history of the dialogue and any requisite information that must be obtained from the user. In the simulations performed here, the DM has a goal—to acquire values from the user for the following five attributes in the flight domain: source city and state, destination city and state, and date. As described earlier, the simulated user randomly selects a value for each attribute, and generates a creative utterance containing a subset of these attributes. After the recognition, parsing, and interpretation in context of the generated waveform, the corresponding semantic representation is passed to the DM.

6.1.1 Overall Dialogue Strategy

Figure 6-2 shows our strategy for acquiring a set of attributes. The DM searches the user's input for any required attributes and, finding one, checks to see if the attribute/value pair has already been confirmed by the user. If so, the DM continues to look for other attributes in the input. If acquiring an unconfirmed attribute has not already been aborted (due to reaching a maximum number of failed attempts), the DM makes the following moves. For a hypothesized date it simply requests explicit confirmation of the value. A source or destination will either be marked as an *unknown* city, or be accompanied by a confidence score for the hypothesized city name. A high confidence score causes the DM to accept the city as the user's intention. Given a low confidence score or an unknown city, the DM initiates a series of recovery attempts, including SAS requests and explicit confirmations, until the attribute's value has been confidently acquired, or its acquisition has been aborted. The DM then continues to

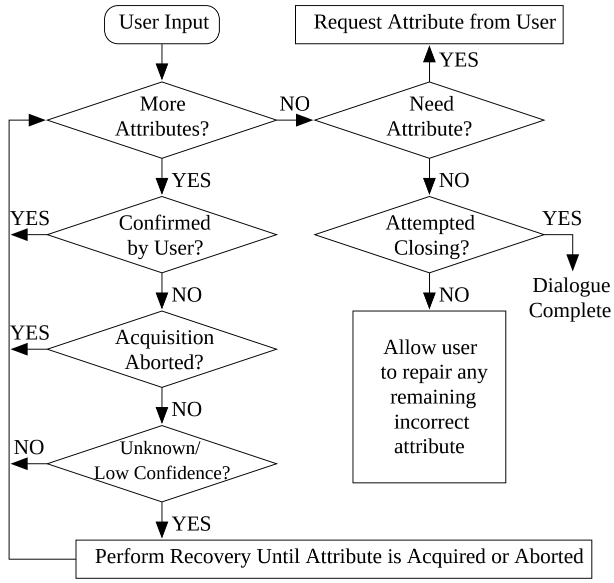

Figure 6-2: Flow chart of the dialogue manager's strategy for acquiring values for a set of attributes.

look for more attributes in the user's input.

After the DM has processed all attributes in the input, it cycles through the required attributes to see if any has not yet been provided by the user. In such a case, it asks the user to provide a value for the attribute. The simulated user provides exactly the information requested by the DM. If a city or state is specifically requested, the user speaks only the city or state name. If a date is requested, the date is incorporated into a templated utterance, as described in Chapter 4, and is then spoken by the user. However, if the DM implicitly confirms a city name with the user that is not correct (e.g., "Okay, to Dallas. Can you provide a date?"), the user provides the correct city name, as well as the requested date.

Once the DM has confidently acquired all attributes, it attempts to close the dialogue. The user then has this last chance to correct any values that the DM may have incorrectly acquired.

6.1.2 Experimental Configurations

The above strategy was utilized in five simulation experiments. For all experiments, a dictionary of about 20,000 entries specified the set of cities considered for a match in an SAS subdialogue, as previously described in Section 3.3. In all configurations, the system ran a main recognizer (capable of recognizing nearly 500 city names) alongside the specialized SAS recognizer whenever it prompted for an SAS utterance, and both hypotheses were given to the DM for consideration. The system attempted only one error correction cycle per attribute, although it allowed the user one last chance to initiate an additional correction at the end. In each configuration, 50 dialogues were simulated, in which the initial user utterance contained both *source* and *destination*. The *date* was subsequently requested by the system. A known city (i.e., recognizable by MERCURY) was randomly selected from a uniform distribution over nearly 360 unique American city names, while an unknown city was similarly selected over 1,700 unique American city names. A date was randomly selected from a uniform distribution over nearly 1,800 unique date phrases.

Obtaining the full *source* and *destination* often meant first obtaining the city name, followed by the state name, since a large number of city names were ambiguous in our database. Therefore, the measure of attribute accuracy in our results refers to the system correctly identifying the source city, source state, destination city, destination state, and date. If a city name was unique, its corresponding state name was implicitly confirmed, without necessarily ever having been mentioned by the user.

Table 6.1, on page 95, describes the configuration for each of the five simulation experiments. *Source* and *Destination* indicate whether a city name spoken by the user was known (kno) or unknown (unk) to the recognizer. *Initial* indicates whether the user's first utterance of each dialogue contained just the city name or both the city and state names. *SAS* indicates whether the system could ever initiate SAS mode as an error recovery mechanism. *Compliant* indicates whether the user would provide an SAS utterance upon request. Noncompliant users would respond by rephrasing their previous utterance.

The figures on pages 99–101 provide several examples of conversations between the simulated user and the MERCURY system. Figure 6-3 shows a successful dialogue, illustrating the simplest possible dialogue flow. The user's first utterance provides the city and state names for both source and destination. The system only explicitly confirms the date, since the confidence scores for the city and state names were above threshold.

In Figure 6-4, the user provides city names that are unknown to the system. The system correctly obtains the source and destination cities through SAS mode. Since these city names are unique in the database, their associated state names are implicitly confirmed. After the date is confidently acquired, the dialogue is complete.

Figure 6-5 shows how the simulated user can rephrase an utterance to help the system understand. The cities and states are implicitly confirmed by high confidence, except for *Vail*, which is acquired via SAS mode. The system misunderstands the implicit month (April) in U3. It is likely the system misrecognized "Make" as "May", thereby hypothesizing the incorrect month. The same date phrase is plugged into a new template (U5), and the system is able to correctly acquire the intended date.

Figure 6-6 shows a dialogue with a noncompliant simulated user. The system prompts for an SAS response (S1). Instead of providing the requested format, however, the user just provides a new utterance containing the source city (U2). The system hypothesizes *Reading* (S2), but that is rejected by the user (U3). The system again requests an SAS response (S3) and the user is again noncompliant (U4). This time, however, the system implicitly confirms *Utica* and *Oakland* due to high confidence. The system proceeds to request the state names and date, and the dialogue is successfully completed.

Figure 6-7 provides an example of a failed dialogue. Despite the noncompliant user behavior in U3 and U7, the system is able to correctly acquire all city and state names. The system is unable to understand *February nineteenth* as the user's intended date, however. The system misrecognizes the day as *ninth* (S9) and prompts for the date once again (S10), following the user's rejection (U10). The user rephrases the utterance (U11), but the system hypothesizes the same value, which has already been rejected. Having reached the maximum number of explicit requests for the date, the system apologizes and ends the dialogue. This example demonstrates the need for a more aggressive approach to obtaining dates. For example, the *N*-best list of recognizer hypotheses could be searched for alternate dates.

Configuration	A	B	C	D	E
Source	kno	kno	kno	kno	unk
Destination	kno	kno	kno	kno	unk
Initial	city	city&state	city	city	city
SAS	yes	yes	yes	no	yes
Compliant	yes	yes	no	n/a	yes

Table 6.1: Configurations for user simulation experiments. The user spoke a known (*kno*) or unknown (*unk*) *Source* or *Destination*. Either *city* or *city&state* was provided in the user's *Initial* utterance. The system chose whether or not to initiate speak-and-spell (*SAS*) mode recovery. The user may or may not have been *Compliant* with a request for a speak-and-spell utterance.

Configuration	A	B	C	D	E
SuccDialogue	31/50	36/50	32/50	33/50	30/50
AvgSuccTurns	7.5	6.6	7.5	7.7	9.0
AvgFailTurns	9.0	8.4	10.2	9.1	10.3
AvgAccuracy (%)	87.2	88.4	88.8	91.4	85.9

Table 6.2: Results for the simulation configurations in Table 6.1. *SuccDialogue* is the number of successful dialogues, that is, each dialogue had 100% attribute accuracy. *AvgSuccTurns* and *AvgFailTurns* are the average turns per successful and failed dialogue, respectively. *AvgAccuracy* is the overall average attribute accuracy per dialogue.

6.1.3 Simulation Experiments and Results

Table 6.2 shows the results of the five simulation experiments. A single dialogue is considered *successful* if all acquired attributes were correct. We also report average number of turns per both successful and failed dialogue, and mean attribute accuracy over all dialogues.

Recognizing that the number of successful dialogues varied among different runs of a given configuration, mainly due to the variety in the synthesized speech, we ran 10 simulations of 50 dialogues each, for configuration B. A mean of 33.9 ± 2.5 on the number of successful dialogues was obtained.

Configuration A serves as a baseline, in which the simulated user provided known city names, and the user was compliant to SAS requests. Since only the city name was provided in the user's initial utterance, the state name was frequently requested, likely elevating the average number of turns per successful dialogue. This configuration resulted in 31 successful dialogues out of 50.

The best dialogue success rate is realized in configuration B, identical to configuration A, except that state names were also provided in the user's initial utterance. While only 36 of 50 dialogues were successful, more than 88% of the attribute values were correctly obtained. As expected, configuration B also yielded the lowest average of 6.6 turns per successful dialogue, beating out configuration A, since the system did not have to request state names. This result reflects the efficiency of packing more information into a single initial utterance.

Configuration C models a noncompliant user, which responded to an SAS request by rephrasing the previously spoken utterance. The simultaneous use of two recognizers enabled the system to handle this situation appropriately, as confirmed by the results. Interestingly, the overall success rate was higher than in A, suggesting that this strategy is actually a productive one. One would hope that real users might be inclined to ignore an SAS request precisely in those situations where the system plausibly knows the intended city.

In configuration D, the system never adopted a strategy of soliciting an SAS utterance from the user. It could only request explicit confirmation of its own hypothesized city names. The results were surprisingly good, with 33 successful dialogues and an average attribute accuracy of 91.4% – the best performance achieved for any configuration. This result makes it clear that SAS subdialogues are not very effective for in-vocabulary city names.

Configuration E is the one situation where SAS subdialogues are clearly useful. The city names provided by the simulated user were all *unknown* to the system. This means that no city names could be understood without a mechanism such as speak-and-spell. Because the city names were out-of-vocabulary, the recognizer would be expected to propose the unknown hypothesis more often, or it should assign low confidence scores to any known cities that were erroneously hypothesized. As expected, this strategy resulted in an increased average number of turns per successful dialogue at 9.0. Although this configuration gives the worst performance, the fact that nearly 86% of the acquired attribute values were correct is very encouraging.

Comparing configuration A with configurations C and D might lead one to believe that SAS mode is not particularly effective. However, its main goal is to allow cities unknown to the original recognizer to be handled. Without SAS mode, configuration E would have yielded a 0% dialogue success rate. For the other configurations, it is more a matter of hoping that SAS mode will not degrade performance, rather than expecting it to improve.

For each experiment, the average number of turns per *failed* dialogue was greater than that per successful dialogue. Limits on acquisition methods must be enforced in order to avoid run-on dialogues. The system performed each of the following at most twice per attribute: explicit request, explicit confirmation, and SAS mode initiation. The tradeoff lies in loosening the limits to potentially acquire the correct value, at the expense of longer dialogues and possible increased user frustration.

Over all experiments, the average attribute accuracy for failed dialogues was nearly 67%, or almost 3.4 of 5 attributes correct. Interestingly, no single attribute stood out as being more difficult to acquire than another. The *source* value was incorrect in 18% of the acquisitions, the *date* in 16%, and the *destination* in 14%.

Overall, the simulation results illustrate how certain dialogue strategies are more appropriate for specific situations. These simulations can help to make new systems more robust by determining the most effective strategies for user behaviors as yet unobserved in real usage. Simulating various user/system configurations can also allow us to identify which types of user behavior would be (un)successful with a given system. For example, without the dual recognizers, configuration C, with the noncompliant user, would have been disastrous.

6.2 Summary

This chapter described the replacement of a real user with a simulated user in the dialogue system cycle. The work detailed in Chapter 3 suggested that user simulation was a viable option for assisting in the development of the speak-and-spell recovery strategy. Simulation enables batch processing of many dialogues, eliminating the need for real users for initial development. The novel method of creative utterance generation via templates, introduced in Chapter 4, was utilized in order to more realistically model the variability of real user utterances. This method also enabled the splicing of synthesized words into the templates, facilitating the performance evaluation of unknown city names.

The generation of five sets of simulated dialogues revealed how the system might perform with real users in terms of attribute acquisition and error recovery. At the same time, the simulation of user utterances was a tremendous tool for debugging unanticipated situations. Since the simulated inputs are based on patterns derived from real users' utterances, they provide a rich set of linguistic variants. This is an invaluable way to test the dialogue manager's robustness to misrecognitions.

While the user simulation model is useful for dialogue strategy development, quite a bit of manual coding is required for the dialogue manager. Specifically, it is difficult to determine, *a priori*, how the dialogue manager should react when a new value for an attribute is hypothesized, especially when it conflicts with the currently believed value. In the case of running simultaneous recognizers, as described earlier, it can be difficult to tell which recognizer to trust the most.

Automatic learning is currently a popular trend in dialogue system development. Learning rules from data not only minimizes the manual coding of rules for the dialogue manager, but also makes decisions based on data, that would not necessarily be intuitive or anticipatable by a human developer. The next chapter explores how user simulation has been used in the automatic learning of dialogue management rules, and how those rules have been applied to the specific problem of recovering from city name errors.

U1:	Um, looking for flights from Evansville, Wisconsin to Monroe, Arkansas.
S1:	Okay, can you provide a date?
U2:	I will be traveling on December sixteenth.
S2:	Okay, did you say on December sixteenth as your date?
U3:	Yes.
S3:	Okay, from Evansville, Wisconsin to Monroe, Arkansas on December 16th. Thank you.

Figure 6-3: This example demonstrates the simplest dialogue flow for the given system configuration. In this case, only the date is explicitly requested and confirmed, because all city and state names were implicitly confirmed due to high confidence.

U1:	Flights from Vancourt to Humphrey.
S1:	Please speak the departure city, followed by its spelling, like this: Bel Air, B E L A I R.
U2:	Vancourt, V A N C O U R T.
S2:	Okay, did you say Vancourt as your source?
U3:	Yes.
S3:	Please speak the arrival city, followed by its spelling, like this: Bel Air, B E L A I R.
U4:	Humphrey, H U M P H R E Y.
S4:	Okay, did you say Humphrey as your destination?
U5:	Yes.
S5:	Can you provide a date?
U6:	I said December five.
S6:	Okay, did you say on December fifth as your date?
U7:	Yes.
S7:	Okay, from Vancourt, Texas to Humphrey, Nebraska on December 5th. Thank you.

Figure 6-4: In this example, both cities are unknown to the system's recognizer and can only be acquired through speak-and-spell mode. The city names are unique in the database, so the state names are implicitly confirmed, and their explicit acquisition is not required.

U1:	I would like to go to Belgrade, Maine from Vail, Colorado.
S1:	Please speak the departure city, followed by its spelling, like this: Bel Air, B E L A I R.
U2:	Vail, V A I L.
S2:	Okay, can you provide a date?
U3:	Make that the thirtieth.
S3:	Okay, did you say on May thirtieth as your date?
U4:	No.
S4:	Sorry, can you try saying the date again?
U5:	I would like to travel the thirtieth.
S5:	Okay, did you say on April thirtieth as your date?
U6:	Yes.
S6:	Okay, from Belgrade, Maine to Vail, Colorado on April 30th. Thank you.

Figure 6-5: In this example, the system requests a date from the user, but misunderstands the user's response in U3. The user rephrases its utterance in U5 by selecting a new template and plugging in the same date phrase, *the thirtieth*. The system is then able to acquire the user's date.

U1:	Fly from Utica arriving to Oakland.
S1:	Please speak the departure city, followed by its spelling, like this: Bel Air, B E L A I R.
U2:	I am sorry, I am leaving from Utica.
S2:	Okay, did you say Reading as your source?
U3:	No.
S3:	Please speak the departure city, followed by its spelling, like this: Bel Air, B E L A I R.
U4:	Um, what about flights from Utica to Oakland?
S4:	Can you provide the source state name?
U5:	Illinois.
S5:	Can you provide the destination state name?
U6:	Maryland.
S6:	Can you provide a date?
U7:	The, uh, on November sixth.
S7:	Okay, did you say on November sixth as your date?
U8:	Yes.
S8:	Okay, from Utica, Illinois to Oakland, Maryland on November 6th. Thank you.

Figure 6-6: This example demonstrates the behavior of a noncompliant user. Here, noncompliance means that when the system requests a speak-and-spell (SAS) utterance, the user instead rephrases its previous utterance as in U2 and U4. Nevertheless, the system is able to correctly recognize the *source* and *destination* cities, by simultaneously running the main recognizer alongside the specialized SAS recognizer and observing both outputs. The dialogue is successful after the system acquires the state names and the date.

U1:	How about from Austin to Kiel.
S1:	Okay, did you say Boston as your source?
U2:	No.
S2:	Please speak the departure city, followed by its spelling, like this: Bel Air, B E L A I R.
U3:	Um, Austin to Kiel.
S3:	Okay, did you say Austin as your source?
U4:	Yes.
S4:	Can you provide the source state name?
U5:	Pennsylvania.
S5:	What city will you be traveling to?
U6:	Kiel.
S6:	Please speak the arrival city, followed by its spelling, like this: Bel Air, B E L A I R.
U7:	I want to fly out to Kiel.
S7:	Okay, did you say Kiel as your destination?
U8:	Yes.
S8:	Can you provide a date?
U9:	What if I leave on February nineteenth?
S9:	Okay, did you say on February ninth as your date?
U10:	No.
S10:	Sorry, can you try saying the date again?
U11:	Departure on February nineteenth.
S11:	I'm sorry, I don't know all your intentions, but we have to quit this dialogue.

Figure 6-7: In this example of a failed dialogue, the system correctly acquires *source* and *destination*, but it has trouble obtaining the proper date. The system hypothesizes *February ninth* (S9), which the user denies. After the user rephrases the utterance (U11), the system hypothesizes the same date, but does not ask about it again, since the value has already been denied. The system abandons the dialogue after having reached the maximum number of attempts for date acquisition.

Chapter 7

User Simulation: Learning When and How to React

In the previous chapter, it was shown how a user simulation framework could be used to assess the performance of, and further develop, the speak-and-spell recovery strategy, or any dialogue strategy, in general. While the technique was found to be very useful, the dialogue manager required a great deal of manual rule specification to control the system's side of the conversation. Automatically learning such rules would allow a developer's time to be spent much more efficiently.

In this chapter, we consider two problems that can arise in the course of a human-computer dialogue: the *conflict* problem and the *compliance* problem. In the first problem, the DM must decide how to respond when hypothesized values for an attribute conflict. A description of the manually specified rules used to address this problem is provided, followed by an alternative approach, in which user simulation was utilized to generate data for learning a decision model. Simulation experiments are then discussed, the results of which enable an assessment of the performance of the learned decision model versus the manual decision model.

In the second problem, the DM must determine whether a user was compliant with the system's request to spell the intended value. Once again, a decision model is learned from data generated with a simulated user and the performance of the learned model is assessed.

7.1 The Conflict Problem

The specific situation considered here, henceforth known as the *conflict problem*, is when a city name is hypothesized for a given attribute (e.g., flight arrival city), and it conflicts with the system's currently believed value for that attribute. How should the system react to this new city name? Should it be ignored, implicitly or explicitly confirmed, or substantiated by a spelling? Figure 7-1 shows a dialogue between a simulated user (U) and the MERCURY flight reservation system (S), in which this situation occurs. The dialogue proceeds rather smoothly until the simulated user provides a date in U7. The system hypothesizes that the user said, *Vienna*, since that value's confidence was sufficiently high, according to some manually specified threshold. The conflicting and incorrect source city, *Vienna*, is then implicitly confirmed in S7; ideally, it should have been ignored. Incidentally, no date was hypothesized. Consequently, in U8, the user must again provide the date, but must also correct the source city. The user repeats *Asheville* until the system finally correctly recognizes the city name and implicitly confirms it in S11.

Another cause of the conflict problem is when an unknown word is hypothesized as a known word that sounds very similar. For example, the unknown country name, *Dominica*, may be misrecognized as the known state name, *Connecticut*. An unknown word model, which allows the recognizer to tag a suspicious word as potentially unknown, can help alleviate this problem. Sometimes, however, when a truly unknown word is spoken, the recognizer obtains such high confidence for a known word, that the unknown word hypothesis never even appears in the N-best list.

One possible technique to address the conflict problem is for the system developer to manually specify a set of rules, taking various features into consideration. This set of rules will then tell the dialogue manager what to do with the newly hypothesized value. Another possible method is for the system, itself, to learn the decision rules in either a supervised or unsupervised manner. In the system described in Chapter 6, the dialogue manager utilized manually specified decision rules. The following section describes what criteria were used in this rule specification, since these rules would be

U1:	Actually that was a flight from Asheville, North Carolina to Spring Mount, Pennsylvania.
S1:	*From Ashland, North Carolina. I have no knowledge of Ashland, North Carolina. Please make a correction*
U2:	I want to check a flight from Asheville, North Carolina.
S2:	*Did you say from Asheville?*
U3:	Yes.
S3:	*From Asheville. Did you say to Scranton?*
U4:	No.
S4:	*Please speak the arrival city, followed by its spelling, like this: Boston, B O S T O N.*
U5:	Spring Mount. S P R I N G M O U N T.
S5:	*Did you say to Spring Mount?*
U6:	Yes.
S6:	*To Spring Mount, Pennsylvania. Can you provide a date?*
U7:	Well, let's see, on fifteen June.
S7:	*From Vienna. Can you try saying the date again?*
U8:	Okay, I want to leave Asheville on fifteen June.
S8:	*Did you say on June fifteenth?*
U9:	Yes.
S9:	*Okay from Vienna, North Carolina to Spring Mount, Pennsylvania on June fifteenth. Please correct me, or say, clear history, to start over.*
U10:	I am leaving from Asheville.
S10:	*Okay from Vienna, North Carolina to Spring Mount, Pennsylvania on June fifteenth. Please correct me, or say, clear history, to start over.*
U11:	I would like to leave from Asheville.
S11:	*Okay from Asheville, North Carolina to Spring Mount, Pennsylvania on June fifteenth. Please correct me, or say, clear history, to start over.*

Figure 7-1: Example dialogue between a simulated user (U) and the MERCURY system (S). The system hypothesizes *Vienna* from the user's utterance (U7), and proceeds to implicitly confirm the incorrect city name in S7.

used in a baseline system to which the performance of the learned rules could be compared. This alternative approach of learning decision rules is explored later in Section 7.3.

7.2 A Manual Solution

The manual specification of rules for dialogue management can result in very successful performance. The initial set of rules is often very basic and inadequate; however, the developer can modify and add rules as the system is used, producing a very mature set over time. Such a method enables very specific dialogue situations to be handled, even though they may occur rather infrequently.

One drawback of the manual method is the amount of time required to devise and hand-tune the rules. In addition, the system developer is most likely unable to anticipate the necessary rules for handling all possible dialogue situations. Nonetheless, many developers enjoy the benefits of the manual method and are willing to invest the time required to manage the rules.

The system described in Chapter 6 employed a manually specified set of rules to address the conflict problem. The same manual rules would be used for a baseline system to which the system employing the learned rules could be compared. Manual rule specification had been used up to this point, since it was the traditional means of dialogue control in the GALAXY systems.

Figure 7-2 provides pseudocode describing the manual rules. If the currently believed value is grounded (that is, if the value was explicitly confirmed by the user, or a unique database entry was found for the attribute given the current city/state beliefs, or the value matched a spelling from the user), then the confidence score of the newly hypothesized value is assessed. If its value is sufficiently high, according to a manually specified threshold, then the new value will be acknowledged; otherwise, the new value will be ignored. If the currently believed value is not grounded, then the new value will be acknowledged.

Once the decision is made to acknowledge the new value, it is subjected to the

```
manual_rules() {
   if (current belief is grounded) {
      if (new value has high confidence)
         acknowledge_new_value();
      else ignore_new_value();
   }
   else acknowledge_new_value();
}

acknowledge_new_value() {
   if (new value is unknown word hypothesis)
      request_spelling();
   else consult_confidence_threshold_spectrum();
}
```

Figure 7-2: Pseudocode describing the manually specified rules used to select an action in the case of the conflict problem.

Low Threshold	High Threshold	Action
<1500	1500	Request spelling
1501	3999	Explicit confirmation
4000	>4000	Implicit confirmation

Table 7.1: Confidence threshold spectrum used by the manual rule method to determine the next action in response to the conflict problem.

same rules that would apply if the value were being introduced for the first time, in a non-conflicting manner. If the new value is the unknown word hypothesis, a spelling request is initiated. Otherwise, the system consults a spectrum of static, manually specified confidence thresholds, as shown in Table 7.1, to determine the next action.

While manual rule specification can be rather ad-hoc, it is still able to contribute to the successful performance of a dialogue system, as shown by the results in Chapter 6.

7.3 A Learned Solution

An alternative approach to addressing the conflict problem is to employ a learning mechanism. A solution involving learning could help eliminate the need for simple heuristics and reduce the time required to obtain a decision model for a specific dialogue situation. This section will describe the learning approach utilized in this research, as well as how user simulation was employed to accomplish this task.

7.3.1 User Simulation

As mentioned in Chapter 6, the simulated user can replace a real user in the dialogue system cycle, enabling a large number of dialogues to be collected and analyzed. This technique bypasses the high cost of finding real users and encouraging them to use a dialogue system. Another feature of the simulated user is that it can be programmed for certain behaviors, such as noncompliance to a system request, that may only occur rarely in real user data.

An additional benefit of the simulated user is the existence of an oracle, that is, the system can have access to the user's true intentions, which is not possible when real users are involved. Of course, this information is only used for the purpose of assessing the actions taken by the system; it does not influence which action the system actually takes. In addition to producing an utterance, the simulated user sends its intentions to the dialogue manager. This knowledge is essential for gathering data that can be used for training the decision model.

The method used in this research for learning a decision model is as follows: in a simulated dialogue, upon encountering the conflict problem, the system records a set of feature values from the N-best list and the dialogue state. The system also records an "ideal" action; that is, the response that would hopefully lead to the fewest number of turns to successfully complete the dialogue task. These actions, defined in Section 7.3.2, are determined from the system's current state of beliefs, as well as the user's true intentions. After many simulations, the resulting data can then be used to produce a model, mapping a feature vector to an action. In the best case, the feature vector would contain enough information to predict the ideal action when the conflict problem is encountered. This learned model could then be employed in a system with real users.

Scenario	*Intended City*	*Current Belief*	*Hypothesis*	*Ideal Response*
A	Boston	Boston	Austin	Ignore
B	Boston	Austin	Boston	Implicit Confirm
C	Boston	Austin	Houston	Explicit Confirm
D	Parrish	Paris	Harris	Request Spelling

Table 7.2: Example scenarios showing the user's *Intended City*, the system's *Current Belief* and a conflicting *Hypothesis*. Each scenario triggers a corresponding *Ideal Response* from the system.

7.3.2 Ideal Actions

In order to learn a decision model, a set of "ideal" actions was devised, which indicates what the system should do when it encounters the conflict problem. The actions are not absolutely ideal, but they are rules which are designed to result in the system's acquisition of the user's intended city name, without extraneous intervening actions. The actions are defined as follows:

1. **Ignore**: when the system's current belief is *already* the user's intended city,
2. **Implicit Confirm**: when the newly hypothesized city is the user's intended city,
3. **Explicit Confirm**: when the newly hypothesized city is *not* the user's intended city, but the user's intended city is a *known* city,
4. **Request Spelling**: when the user's intended city is an *unknown* city.

The choice of these actions can be explained with the help of Table 7.2. In scenario A, the user's intended city is *Boston* and the system's current belief is the same. If the system were to incorrectly hypothesize *Austin* from the user's next utterance, the system's ideal response would be to ignore the newly hypothesized city name since the system had already acquired the user's intended city.

In scenario B, the user's intended city is *Boston*, but the system currently believes that *Austin* is the intended city. If the system were to then hypothesize *Boston* from the user's next utterance, the system should ideally implicitly confirm *Boston* since it is the user's intention. Any explicit confirmations would simply be superfluous, leading to a longer dialogue.

In scenario C, the user's intended city and the system's current belief are the same as in B, but the new hypothesis is *Houston*. Since this city is not what the user intended, but the intended city, *Boston*, is known to the recognizer, the system should present an explicit confirmation to the user, as in, "Did you say Houston?" The reason for this response is that, if the suggested city is not correct, users have been observed to include the *correct* city name in their reply to the system [44, 12], such as, "No, I said Boston." The hope is that the system would be able to correctly recognize the user's intended city if it were repeated. If the user simply said, "No", then the dialogue would continue as usual.

Finally, in scenario D, the user's intended city, *Parrish*, is unknown to the recognizer. In this case, the ideal response would be to request a spelling for the city name, since a spelling is the only way to acquire a value unknown to the system.

It can now be seen that in order to know the ideal action, the system must know the user's intentions. When real users are involved, this information is only available following manual annotation by experts. User simulation, however, enables access to this knowledge, allowing the automatic assessment of decisions as they are made.

7.3.3 Features

Various features have been used by researchers in learning approaches to dialogue management, including user responses to system prompts [44], previous system question types [80], prosodic cues [46], and discourse features [40].

The 16 features chosen for this work are shown in Table 7.3. They are based on information from the recognizer, as well as on information from the dialogue state. A variety of features are derived from the N-best list returned from the recognizer, including *num_nbest_values*, which indicates the number of unique values hypothesized in the N-best list for the given attribute, and *nbest_dominance*, which provides a distribution of the hypothesized values for a given attribute. A variety of N-best features were chosen since minute details may suggest the recognizer's uncertainty of the user's intended value. For example, if *num_nbest_values* is relatively high, it may suggest that the user's intended value is confusable with other values, and a spelling request might be the appropriate action. As another example, if *num_nbest_values* is 3, but the *nbest_dominance* distribution is 0.90, 0.04, and 0.06, it is likely that the

Confirmation status of currently believed city name (cur_gnd): Possible values are *not confirmed*, *confirmed by user*, *confirmed by rules*, *high confidence*, *high initial confidence*, *found via spelling in hypotheses*, *found via spelling in list*, *found via spelling using edit distance metric*, *last resort via best spelling hypothesis*, *confirmed via choices list*, *confirmed via uniqueness in database*.
Confirmation status of newly hypothesized city name (new_gnd): See values for cur_gnd.
Confidence score of city name (conf): Score obtained by the recognizer for the newly hypothesized value.
Number of *N*-best values (num_nbest_values): Number of unique city names for a given attribute in the *N*-best list.
Unknown word (value_is_oov): Is the top *N*-best hypothesis the unknown word hypothesis?
Number of unknown hypotheses (nbest_oov_count): Number of *N*-best hypotheses containing the unknown word for the given attribute.
***N*-best dominance (nbest_dominance)**: What percentage of the slot values in the *N*-best list are this value?
***N*-best slot absence (nbest_slot_absence)**: Number of *N*-best hypotheses without a hypothesized value for this attribute.
Continuous depth from top (cont_depth_from_top): Continuous occurrence of city name from top of *N*-best list.
Depth of first occurrence (first_occur_depth): At what position in the *N*-best list does this value first occur?
Attribute complete (complete): Has a unique entry in the database been found for this attribute?
Found in spelling hypotheses (found_in_sas_hyps): Was this value found in the set of hypotheses used during spelling recovery?
Found in spelling list (found_in_sas_list): Was this value found in the database of city names used during spelling recovery?
Number of value conflicts (num_value_conflicts): Number of conflicting values for this attribute during this dialogue.
Number of no database results (num_no_db_results): Number of empty results from database query involving this attribute's value.
Number of times slot hypothesized (num_times_slot_hyped): Number of times a value has been hypothesized for this attribute.

Table 7.3: The 16 features chosen to predict the "ideal" system response in the case of the conflict problem.

90% dominant value was intended, and perhaps implicit confirmation is appropriate.

Some features are also based on information maintained in the dialogue state. One example is *num_value_conflicts*, which indicates how many different values for a given attribute the system has believed over the course of the dialogue. If this feature value is high, it is likely the system is having difficulty acquiring the user's intended value and, perhaps, a spelling request would be beneficial.

Another feature based on the dialogue state is *cur_gnd*, which represents the grounding of the value currently believed by the system. As shown in the first entry of Table 7.3, there are many possible values for this feature, including *not confirmed*, *confirmed by user*, *high confidence*, and *found via spelling in hypotheses*. A value of *not confirmed* indicates that the value was recognized with a relatively low confidence and has not been confirmed by the user. If a conflicting value were hypothesized in this case, it would seem more likely for the currently believed value to be overridden than if the value of *cur_gnd* were *confirmed by user* or *high confidence*.

7.3.4 Weka and JRip

The learned models in this research were obtained using Weka (Waikato Environment for Knowledge Analysis) [87], a publicly available, Java-based machine learning package developed at the University of Waikato in New Zealand. The package implements a variety of standard machine learning algorithms. The algorithm chosen for training was JRip, Weka's implementation of RIPPER (Repeated Incremental Pruning to Produce Error Reduction), which was originally conceived by Cohen [18]. A RIPPER implementation was chosen since it is commonly used in the area of supervised learning approaches to dialogue management [50, 41, 80, 46, 83], as discussed in Section 1.2.2.

7.3.5 The Learned Rules

The training data were obtained by generating 596 dialogues with a compliant simulated user; that is, the user always provided a speak-and-spell utterance when requested. The majority of utterances were generated via the creative utterance synthesis, described in Chapter 4, while any spelling utterances were generated by DECtalk. In 298 dialogues, the user asked about flights from a city *known* to the recognizer to a city *unknown* to the recognizer. In the remaining 298 dialogues, the user asked about flights from an unknown city to a known city. There were 298 known cities; each city was used twice. There were 596 unknown city names; each city was used only once. The city names were randomly combined into unique pairs prior to the simulations.

The task of the simulated user was to get the system to correctly understand the source city and state, the destination city and state, and the date. The user's first utterance contained only the source and destination cities. The states and date were provided only when requested by the system.

The simulated dialogues gave rise to 890 instances of the conflict problem, for which the corresponding feature vector/ideal action pairs were recorded. These instances were then organized into the file format required by Weka. A ten-fold cross-validation was then performed on the data using JRip to learn the decision model.

Figure 7-3 shows the learned set of 11 rules, in which the actions are *implicit confirm*, *explicit confirm*, *request spelling*, and *ignore*. To classify a new instance, the rules are scanned linearly and the action corresponding to the first matching rule is taken. If no rules match, the default rule is activated, which, in this case, triggers the *ignore* action.The numbers in parentheses following each rule represent *coverage/errors* in the training data. For example, `(50.0/12.0)` would mean that the corresponding rule was matched for 50 training instances and 12 of those 50 were classification errors. All of the 16 features are represented in the rules, except for *new_gnd* and *found_in_sas_hyps*. The *cur_gnd* feature occurs in eight of the 11 rules, indicating the importance of the grounding of the current belief when considering whether to doubt its correctness.

Some rules demonstrate a degree of complexity not likely to arise from manual specification. Rule 2, for example, says that if at least 72.2% of the relevant slot values in the *N*-best list are the newly hypothesized value (nbest_dominance), and none of the *N*-best hypotheses are missing a value for the relevant slot (nbest_slot_absence),

```
1. (:num_nbest_values <= 6) and (:cur_gnd = hi_conf) and (:conf >= 436)
   and (:conf <= 6326) and (:first_occur_depth <= 0.076923)
   and (:num_no_db_results >= 1)
     => action = implicit (27.0/5.0)
2. (:nbest_dominance >= 0.722222) and (:nbest_slot_absence <= 0)
   and (:num_nbest_values <= 4) and (:first_occur_depth <= 0.0625)
   and (:conf <= 4766)
     => action = implicit (11.0/2.0)
3. (:cur_gnd = hi_conf) and (:cont_depth_from_top >= 0.25)
   and (:complete >= 1) and (:found_in_sas_list >= 2)
   and (:first_occur_depth <= 0.090909)
     => action = implicit (12.0/3.0)
4. (:num_nbest_values <= 6) and (:cur_gnd = not_conf)
   and (:nbest_dominance >= 0.6)
     => action = implicit (17.0/8.0)
5. (:cur_gnd = hi_conf) and (:nbest_oov_count >= 1)
     => action = spelling (104.0/18.0)
6. (:complete <= 0) and (:num_value_conflicts >= 3)
     => action = spelling (21.0/2.0)
7. (:complete <= 0) and (:cur_gnd = hi_conf)
     => action = spelling (38.0/13.0)
8. (:cur_gnd = not_conf)
     => action = spelling (56.0/16.0)
9. (:num_times_slot_hyped >= 4) and (:cur_gnd = hi_conf)
     => action = spelling (36.0/11.0)
10.(:cur_gnd = sas_hyps) and (:nbest_slot_absence <= 0)
     => action = spelling (17.0/6.0)
11.(default)
     => action = ignore (551.0/50.0)
```

Figure 7-3: The rule set learned by JRip to predict the “ideal” response to the conflict problem. The numbers in parentheses represent *coverage/errors* in the training data.

and the N-best list contains at most four unique values for the relevant slot (num_nbest_values), and the newly hypothesized value occurs in the top hypothesized utterance (first_occur_depth), and the confidence of the new value is at most 4766 (conf), then the new value should be implicitly confirmed.

On the other hand, some learned rules are similar to those that would be specified in a manual rule set. For instance, Rule 5 says that, if the current belief was acquired with high confidence (cur_gnd) and the N-best list contains at least one unknown word hypothesis for the relevant slot (nbest_oov_count), then the system should request a spelling from the user.

One observation to note is that *explicit confirm* is absent from the learned rules. Based on the training data, the learning algorithm simply never found it appropriate to perform explicit confirmation. It turns out that the conditions triggering the "ideal" explicit confirmation are relatively rare. Once again, for the explicit confirmation to be taken, the newly hypothesized city must conflict with the system's current belief, it must not be the user's intended city, and the user's intended city must be known to the recognizer. This result suggests that, in the simulated dialogues used to collect training data, when the user's intended city was known to the recognizer, it was most often recognized correctly, with little or no confusion. Whether this behavior would be observed with real users remained to be determined.

7.4 Testing and Results

Testing was performed to determine if the learned decision model would result in the system taking more "ideal" actions in the case of the conflict problem, than it would if it followed the manual decision model. The hope was that, if more ideal actions were taken with the learned model, the dialogues would result in a higher attribute accuracy and, perhaps, a smaller number of dialogue turns.

The test set consisted of 50 dialogues with a compliant simulated user. A set of 100 city names, not used in the training dialogues, was utilized for the test set. Fifty cities were known to the recognizer and 50 were unknown; each city name was

	Manual	Learned
Average No. Successful Dialogues	33.2	35.9
Average No. Turns per Successful Dialogue	7.8	7.7
Average Attribute Accuracy	86.4%	89.0%

Table 7.4: Results of dialogue analysis for two decision models, *Manual* and *Learned*. The numbers represent the averages of 10 runs of 50 dialogues for each model. A *Successful Dialogue* is one in which there is 100% attribute accuracy.

used only once. Fifty test scenarios were created, which included a known source city and state, as well as an unknown destination city and state. The test set of 50 dialogue scenarios was run ten times, and the results were averaged. In the first run, Envoice was used to creatively generate an initial input string (containing the complete source and destination), and a date was randomly chosen when requested by the dialogue manager. The input string and date were then incorporated into the current scenario off-line for use in the subsequent nine test runs. This ensured that the same 50 scenarios and 50 initial input strings were utilized in all ten test runs. Any user responses beyond the initial input were based on randomly chosen utterance templates, thus providing variability in the wording of utterances in corresponding dialogues across the ten sets.

Table 7.4 provides a comparison of the average results for both decision models. Each dialogue involved five attributes: source city and state, destination city and state, and date. Using the learned model resulted in an increase in the average number of successful dialogues from 33.2 to 35.9. A successful dialogue is one with an attribute accuracy of 100%. An increase of 2.6% in average attribute accuracy is seen in dialogues adhering to the learned model. In order to test for significance, we obtained the average attribute accuracy for each dialogue scenario across the ten runs, resulting in 50 samples for each model. We then performed the Wilcoxon signed-rank test, indicating that the learned model resulted in significantly better performance than the manual model ($p \leq .002$).

The results show only a very slight decrease in the number of turns per successful dialogue when the learned model is used. Although significant gains are not obtained for this measure, the improvement seen in the number of successful dialogues and

Reference→ *Prediction*↓	Ignore (64.0%)	Implicit (10.7%)	Explicit (3.4%)	Spelling (21.9%)
Ignore (77.4%)	38.6	3.9	1.8	10.0
Implicit (14.1%)	6.2	1.8	0.2	1.7
Explicit (2.1%)	0.0	0.7	0.0	0.8
Spelling (6.4%)	0.1	1.1	0.4	2.9

Table 7.5: Confusion matrix for decision made using the manual decision model. An overall accuracy of 61.7% was obtained. The parenthesized percentages indicate the distribution of each action within the total set of either predicted or reference actions.

Reference→ *Prediction*↓	Ignore (71.7%)	Implicit (9.5%)	Explicit (2.7%)	Spelling (16.1%)
Ignore (77.3%)	43.6	2.3	0.8	4.6
Implicit (8.6%)	2.7	1.9	0.0	1.1
Explicit (0.0%)	0.0	0.0	0.0	0.0
Spelling (14.1%)	1.3	2.1	1.0	5.0

Table 7.6: Confusion matrix for decision made using the learned decision model. An overall accuracy of 76.1% was obtained. The parenthesized percentages indicate the distribution of each action within the total set of either predicted or reference actions.

the attribute accuracy suggests that utilizing the learned model can better assist the dialogue manager in acquiring the user's intended attribute values.

A detailed analysis revealed how well each model was able to predict the ideal action in response to the conflict problem. Tables 7.5 and 7.6 provide confusion matrices for the action taken when the manual and learned models, respectively, were followed. The numbers in the tables represent the averages taken over all ten runs. Overall, the learned model significantly outperformed the manual model with an accuracy of 76.1% compared to 61.7%.

Both models performed best in predicting the *ignore* action – the manual model was correct just over 71% (38.6/(38.6 + 3.9 + 1.8 + 10.0)) of the time, while the learned model achieved nearly 85% (43.6/(43.6 + 2.3 + 0.8 + 4.6) accuracy. It should be noted that *ignore* is the default action in the learned rules, meaning that if no other rule was triggered, the newly hypothesized value would simply be ignored.

One might think that the learned model's default *ignore* action makes the model appear as if its better performance is artificial. However, it is also the case that, on average, both the manual and learned models predict the ignore action in a little over 77% of their responses to the conflict problem, as shown in the first columns. Yet, the learned model achieves a better accuracy. This observation suggests that the improved performance of the learned model is not a consequence of the default rule, but rather of the greater refinement and complexity of the other rules.

In the case of implicit confirmation, neither manual nor learned model performed very well. When implicit confirmation was the predicted action, the manual model was correct only 18% (1.8/(6.2 + 1.8 + 0.2 + 1.7) of the time, while the learned model performed slightly better, with an accuracy of 33% (1.9/(2.7 + 1.9 + 0.0 + 1.1)). Both models tended to predict implicit confirmation when the ideal action was, in fact, to *ignore* the newly hypothesized value. This confusion can be quite punishing since implicitly confirming a value, when it is *not* correct, places a burden on the user to correct the value [44]. Such a task can be difficult and frustrating for the user, especially if the system is highly confident in the incorrect value.

The *explicit confirm* action did not appear in the learned rules, likely due to the rarity of the triggering conditions, and thus it was never predicted. While it did happen to be the ideal action in a very small number of cases, the manual model also never successfully made that prediction. Therefore, the conservative approach taken by the learned rules was most likely appropriate.

The learned model predicted *spelling request* in a significantly higher percentage of its predictions (14.1%) than the manual model (6.4%). This result suggests that the learned model is more aggressive about requesting spellings from the user. Such aggressiveness could explain why the manual model is correct 64% (2.9/(0.1 + 1.1 + 0.4 + 2.0)) of the time, while the learned model is correct only 53% (5.0/(1.3 + 2.1 + 1.0 + 5.0)) of the time when a spelling request is predicted. Predicting a spelling request when it was not ideal, simply means that the compliant user would need to provide a spelling, perhaps adding an additional superfluous turn to the dialogue. The results in Table 7.4, however, show that such behavior by the learned model did not lead to an increase in the average number of turns per successful dialogue, so the relatively low accuracy of the learned model is not necessarily detrimental.

A common confusion for both models was ignoring the newly hypothesized value when a spelling request was the ideal action. Such a confusion can be quite damaging, since the system *must* initiate a spelling request to acquire an unknown word. The rightmost columns of Tables 7.5 and 7.6 show that the manual model made a greater percentage of such confusions $(10.0/(10.0 + 1.7 + 0.8 + 2.9) = 65\%)$ than the learned model $(4.6/(4.6 + 1.1 + 0.0 + 5.0) = 43\%)$. However, when a spelling request was the ideal action, the lower right corners of the tables show that the learned model predicted it correctly 47% $(5.0/(4.6 + 1.1 + 0.0 + 5.0)$ of the time versus 19% $(2.9/(10.0 + 1.7 + 0.8 + 2.9))$ for the manual model. This result demonstrates the learned model's better ability to predict when a spelling request should be initiated.

All of the previous results suggest that the learned model outperformed the manual model when predicting the "ideal" action in the case of the conflict problem.

7.5 The Compliance Problem

The previous section showed that simulation and learning techniques could be combined to produce a decision model in response to the conflict problem. Since the framework proved promising, it was thought that its application to identifying user compliance could also be useful. This section describes such an application.

As described in Section 3.4, allowing real users to use the speak-and-spell recovery strategy revealed that users often do not provide a speak-and-spell utterance when requested; in other words, they are noncompliant. In fact, it was shown in Section 6.1.3 that noncompliance is a good strategy for the user when the city in question is known by the recognizer. Hence, this option should be available to users. Until this point, a simple heuristic had been used to decide whether or not a user was being compliant. It was discovered that if a noncompliant utterance was provided, the specialized speak-and-spell recognizer would absorb most of the utterance as part of the unknown word, as described in Section 3.3; only three or four letters would typically be hypothesized. This low number of hypothesized letters was then used to indicate that a speak-and-spell utterance was not likely to have been spoken. One major problem with this heuristic was that when a city name of less than four letters was spelled, it was consistently classified as a noncompliant utterance.

After having experimented with user simulation and rule-learning, it was observed that a decision could be made according to a learned model, simply by mapping a vector of features to the decision of compliant or noncompliant.

7.5.1 From Speak-and-Spell to Spelling

The method of obtaining a spelling from the user up to this point had been the speak-and-spell recognizer, which required a user to provide an utterance such as, "Chicago C H I C A G O." From trial experiments with real users, it was observed that users often simply spelled the city name, omitting the pronunciation of the word. In an effort to ease the burden on the user, and because it seemed more intuitive, it was decided that a plain spelling recognizer would be used in place of the speak-and-spell recognizer, allowing the user to merely spell a city name (e.g., "C H I C A G O").

The modification simply involved replacing the speak-and-spell recognizer with a spelling recognizer and the change would be completely transparent to a user. In addition to enabling a more intuitive response from the user, this strategy would facilitate a more straightforward method for detecting a noncompliant user.

7.5.2 The Learned Model

It is intuitively expected that acoustic and language model scores from a spelling recognizer would be poor if the user were to utter a sentence instead of spelling a word. It was, therefore, decided to utilize these scores from the spelling recognizer as features for learning a model to predict user compliance. The following four features were collected for training:

- acoustic_score
- ngram_score
- total_score (acoustic_score + ngram_score)
- word_length

```
1. (:ngram_score <= -34.1935) and (:total_score <= 93.2063)
     => action = noncompliant (240.0/0.0)
2. (:total_score <= 34.5113)
     => action = noncompliant (30.0/0.0)
3. (default)
     => action = compliant (330.0/0.0)
```

Figure 7-4: The rule set learned by JRip to predict user compliance in response to a spelling request from the system. The numbers in parentheses represent *coverage/errors* in the training data–note that no errors were made.

The *acoustic_score*, *ngram_score*, and *total_score* were obtained directly from the recognizer, while the *word_length* was a simple calculation performed by the dialogue manager. The word length feature was included since it was thought that a noncompliant utterance might give rise to a suspiciously long sequence of hypothesized letters.

In order to collect training data, the simulated user generated both a valid spelling and a noncompliant utterance for each of the 298 known cities and each of the 596 unknown cities used to train the model for the conflict problem. This process resulted in 1,788 training samples. Although the recognizer produced a 10-best list of hypotheses, only the features of the top hypothesis comprised a training sample. After each utterance, the dialogue manager recorded the feature vector as well as whether the simulated user was compliant or noncompliant.

The complete set of training data was processed through Weka's JRip learning algorithm to obtain a prediction model consisting of three rules, as shown in Figure 7-4, where the possible predictions were *compliant* and *noncompliant*. As mentioned earlier, the numbers in parentheses represent *coverage/errors* in the training data. Only the *ngram_score* and *total_score* features appear in the final rule set, which has a default action of *compliant*.

7.5.3 Testing and Results

The learned model was tested on the same set of 50 dialogue scenarios used to test the conflict problem models, as described in Section 7.4. The entire set was tested once with a compliant simulated user and once with a noncompliant simulated user. The system correctly classified all 57 of the compliant user's utterances, and all 162 of the noncompliant user's utterances, yielding 100% accuracy for this task. There were understandably more predictions required for the noncompliant user since, as the user persisted in being noncompliant, the system would continue to request more spellings.

These results suggest that the learned model works quite well. It should, of course, be mentioned that the model was trained and tested on DECtalk's voice for the compliant user. Noncompliant utterances were generated via Envoice and creative utterance generation, as described in Chapter 4. Furthermore, the responses provided by the simulated user were formatted exactly as the system required. The hope is that this model could also be used in a live dialogue system accessible by real users. Data from real users would certainly present a larger variety of responses, providing a much more realistic test of the learned model's performance.

7.6 Summary

This chapter began by describing the *conflict problem*, in which a newly hypothesized value for an attribute conflicts with the system's currently believed value for that attribute. While it is possible for a system developer to manually specify rules, indicating how to react in such a situation, it is rather time-consuming, and does not address unanticipated situations. An approach involving supervised learning was presented here, in which a simulated user is employed to gather data, which is then used to train a rule-based model using JRip, Weka's implementation of the RIPPER rule-induction algorithm. Thus, in the case of the conflict problem, the learned model can be used to make a decision, based on a wide variety of features.

The experiments showed that the system employing the learned model resulted in better performance than when the manual model was used. These results highlighted the utility of such a framework, prompting a similar approach to another problem – predicting whether or not a user is being compliant with the system's request to spell a city name. A JRip model was trained and tested, resulting in perfect classification accuracy.

The learned models in this chapter show good performance in tests with a simulated user, which produces relatively clean, well-formed input. A much more realistic test would be to employ these models in a dialogue system accessed by real users, where the variability in user utterances is much more extreme. The next chapter describes the implementation and deployment of such a system.

Chapter 8

Experiment with Real Users

The learned models described in the previous chapter were tested on simulated users. While such tests certainly *suggest* how the models may perform in situations with real users, they do not guarantee good performance. The only way to assess the utility of the models in a system accessed by real users is to deploy such a system. This chapter describes the telephone-based dialogue system developed and utilized for collecting real user data, thereby providing a more realistic test of the learned models.

8.1 Flight Attribute Data Collection System

In order to collect real user data, a system similar to the one used for simulation was configured. The domain was the MERCURY flight reservation system, and the user's goal was to get the system to understand the attributes in a given scenario, which included a source city and state, a destination city and state, and a date.

Each user obtained a scenario from a specified webpage, which displayed a unique scenario upon each new visit to (or each reload of) the webpage. An example snapshot of the webpage viewed by each user is shown in Figure 8-1. The webpage explains that the project is a data collection experiment for the purpose of spoken dialogue system development. It then instructs the user to call the system and ask about flights according to the scenario provided. It explains that the user's goal is to get the system to correctly understand the city names and date, which the system would

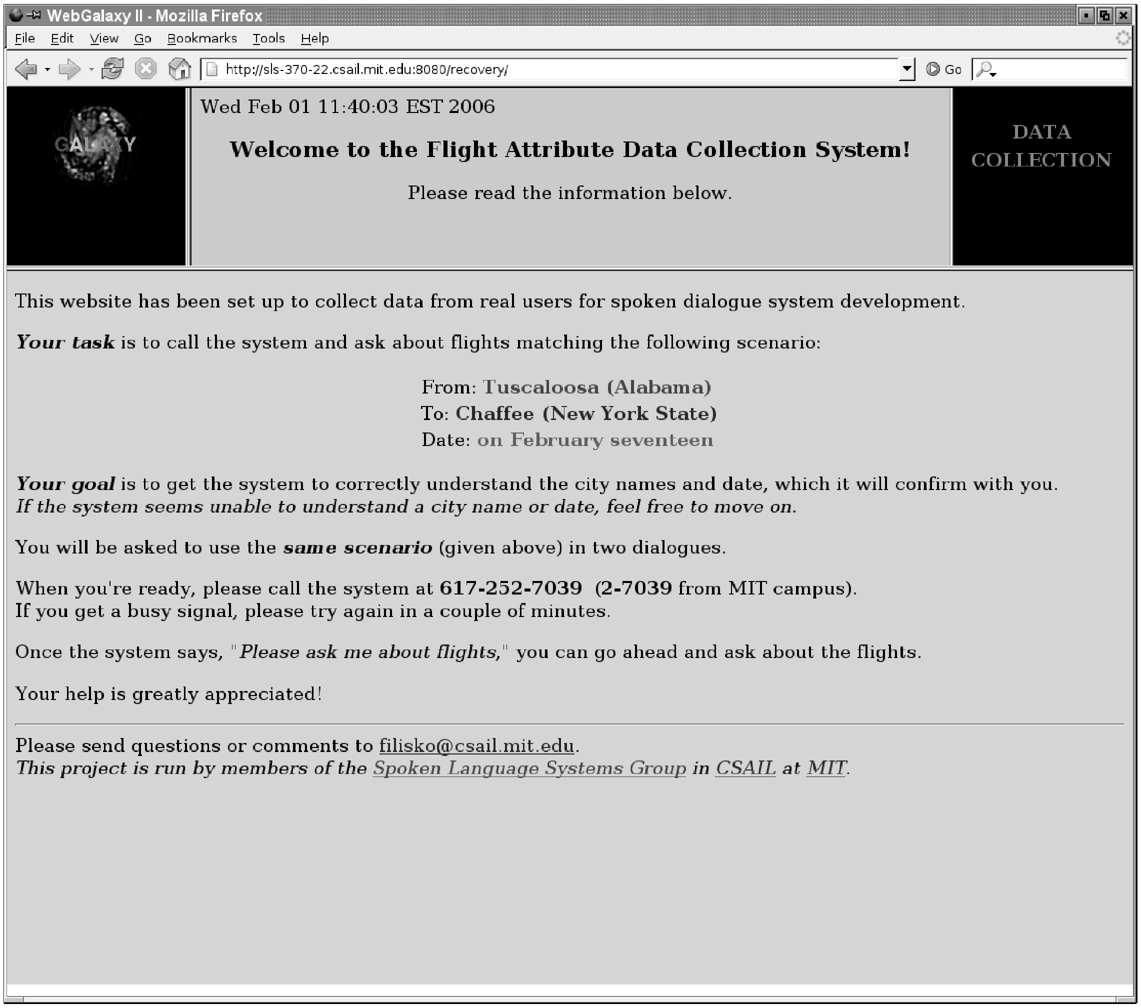

Figure 8-1: Snapshot of the webpage seen by users for the data collection system.

confirm with the user. In order to prevent dialogues from becoming extremely lengthy, the page tells the user to feel free to move on if the system seems unable to understand an attribute's value. The page then provides a telephone number to call to initiate a dialogue. To minimize bias, no example utterances were provided. In addition, the state names are specified in parentheses next to the city name, to suggest that the state name need only be provided for purposes of disambiguation.

8.1.1 Changes from the Simulation System

The most important difference between the data collection system and the simulation system is the replacement of the simulated user by a real user. This greatly increased the potential for a wider variety of user utterances, as well as for unanticipated responses and situations.

Another difference is that instead of requesting a speak-and-spell utterance from the user as an error recovery strategy, only a spelling was requested. This decision was largely based on observations of user behavior from the live test of speak-and-spell, described in Section 3.4. Spelling the word seemed like a more intuitive action, and preliminary tests with a spelling recognizer showed good performance. Therefore, the speak-and-spell recognizer was simply swapped with a spelling recognizer for this experiment.

The last notable difference between the data collection system and the simulation system is that the live system would utilize a learned model to decide whether or not a user was being compliant when the system requested a spelling (see Section 7.5). The results obtained from tests of a simulated user suggested that this model could also be useful when real users were involved. Results of the model's performance are presented later in this chapter.

8.1.2 The Scenarios

The set of scenarios was generated prior to the beginning of the data collection. The city names were taken from the same set of 50 scenarios used to test the decision models for the simulation experiments, as described in Chapter 7. In order to create a large number of scenarios, each of the 50 cities known to the recognizer was paired with each of the 50 cities unknown to the recognizer, for a total of 2,500 possible unique scenarios.

When the webpage was accessed or reloaded, the next scenario in the list was presented to the user. Consequently, every scenario displayed would be unique, unless the entire list were traversed, in which case it would start again from the beginning of the list. The source city in each scenario was always known to the recognizer, and the destination city was always unknown. A date for each scenario was selected at random.

8.2 The Sessions and Users

In order to exactly compare the performance of the system using the manual versus the learned decision model in response to the conflict problem (see Section 7.1), two separate control threads would have to be spawned by the system. Ideally, each user utterance could be sent through both control threads; however, since each decision model could result in a different system request, the user would have to provide different responses. Therefore, this ideal method was not possible.

An alternative method was employed here, in which the system guided each user to perform two sequential dialogues. In the first of the dialogues, the system utilized either the manual or learned decision model upon encountering the conflict problem. Once the first dialogue had ended (i.e., when the system recognized that the user had said, "Clear history," as instructed), the system transparently switched to utilize the other decision model for the second dialogue. The manual model was employed in the first dialogue for half of the sessions, while the learned model was utilized in the remaining half. In order to minimize variability, the webpage instructed the user to utilize the same scenario for both dialogues. This method would enable performance evaluation of a similar set of dialogues for each of the two decision models.

A session from the collected data is shown in Figure 8-2 to help clarify the method. Throughout the first dialogue, the system is following the manual decision model. The system finally summarizes the attribute values in S4 and the user says, "Clear history," in U5 to move on to the second dialogue. After S5, the system switches to follow the learned decision model for the remainder of the dialogue.

Over 33 days of data collection, 48 sessions were recorded by 34 unique users. Users were recruited mainly via e-mail, but also by word-of-mouth. Each session ideally consisted of two dialogues; however, if a user hung up before starting the second dialogue, only one dialogue was available. Five such sessions were excluded from the analysis. Three more sessions were removed since at least one of the component dialogues was outside two standard deviations from the mean number of turns per dialogue. This filtering left a total of 40 valid sessions and 30 unique users.

S0: *Welcome to the flight attribute data collection system. You will be asked to participate in two dialogues. Now, let's begin the first dialogue. Please ask me about flights.*

U1: I need a flight from Riga, Michigan to Yamhill, Oregon on May ninth.
S1: *From Riga, Michigan. Please spell the name of your arrival city.*

U2: Y A M H I L L.
S2: *Did you say to Yamhill?*

U3: Yes.
S3: *To Yamhill, Oregon. Did you say on May ninth?*

U4: Yes.
S4: *Okay, from Riga, Michigan to Yamhill, Oregon on May ninth. Please correct me, or say, "Clear history," to begin the next dialogue.*

U5: Clear history.
S5: *The history has been cleared. Let's begin the second dialogue. Please ask me about flights, using the same scenario as in the first dialogue.*

U6: I need a flight from Riga, Michigan to Yamhill, Oregon on May ninth.
S6: *From Riga, Michigan. Please spell the name of your arrival city.*

U7: I wanna go to Yamhill, Oregon.
S7: *Please spell the name of your arrival city.*

U8: Y A M H I L L.
S8: *Did you say to Yamhill?*

U9: Yes.
S9: *To Yamhill, Oregon. Did you say on May ninth?*

U10: Yes.
S10: *Okay, from Riga, Michigan to Yamhill, Oregon on May ninth. Please correct me, or say, "Goodbye."*

U11: Goodbye.
S11: *Thank you for using the flight attribute data collection system. Have a nice day!*

Figure 8-2: Example data collection session between a real user (U) and the data collection system (S).

	Manual	Learned	Overall
Successful Dialogues	31/40 (77.5%)	34/40 (85.0%)	65/80 (81.3%)
Avg. Turns/Successful Dialogue	8.97	8.50	8.72
Avg. Attribute Accuracy	93.0%	96.0%	94.5%

Table 8.1: Results of dialogue analysis for all dialogues (*Overall*), and dialogues in which the *Manual* and *Learned* decision models were used. A *Successful Dialogue* is one in which there is 100% attribute accuracy.

8.3 Results

As each session was obtained, each component dialogue was transcribed and tagged for ideal actions, in the case of the conflict problem, and user compliance, in the case of the compliance problem. Once all the data had been collected, the dialogues were analyzed and statistics were collected.

8.3.1 Overall Performance

Overall, the results suggest that the system and users worked well together in conveying information from the user to the system. The *Overall* column, shown in Table 8.1, provides statistics for the entire set of dialogues. There were 65 out of 80 dialogues in which 100% of the attribute values (i.e., source city, source state, destination city, destination state, date) were successfully acquired by the system; such dialogues are defined as *successful dialogues*. An average of 94.5% of the attribute values were obtained per dialogue.

8.3.2 Manual vs. Learned Models

Table 8.1 shows that the learned model realized a greater number of successful dialogues, a slightly lower average number of turns per successful dialogue, and a greater average attribute accuracy than the manual model.

As described in Section 8.2, each session was comprised of two dialogues. Since participating in the first dialogue enabled the user to observe the behavior of the system, the user was knowledgeable about the workings of the system upon starting

	Avg. Turns/Dial.	% Dialogues
First Dialogue		
Manual, Success	9.25	72.7
Learned, Success	9.25	88.9
Manual, Failure	11.83	27.3
Learned, Failure	12.00	11.1
Second Dialogue		
Manual, Success	8.67	83.3
Learned, Success	7.83	81.8
Manual, Failure	17.00	16.7
Learned, Failure	13.50	18.2

Table 8.2: Statistics for manual versus learned models in terms of the first or second dialogue in a data collection session. The *% Dialogues* refers to the percentage of first/second manual/learned dialogues.

the second dialogue. The user would know, for example, what kinds of responses were best understood by the system. Since a user was, in effect, no longer an absolute novice to the system after the first dialogue, it was expected that performance would improve for a successful second dialogue.

The results shown in Table 8.2 confirm this expectation. While the average number of turns per successful *first* dialogue is 9.25 for both manual and learned model dialogues, the average per successful *second* dialogue is reduced by 6.3% to 8.67 for the manual model dialogues, but by 15.4% to 7.83 for the learned model dialogues.

It can also be observed that the average number of turns per failed dialogue goes up from the first to the second dialogue for both models. If the user were successful on the first dialogue, this apparently greater user persistence could reflect that the user had faith that the system could eventually acquire all the intended attribute values, and thus the user kept with the dialogue for a few more turns to see if the system would ultimately succeed.

The *% Dialogues* column in Table 8.2 gives the percentage of *First/Second Manual/Learned* dialogues that were either successes or failures. It should be noted that the manual model was utilized in the *first* dialogue for half of the collected sessions, while the learned model was used in the first dialogue for the other half. After filtering invalid sessions, however, there were 22 sessions in which the manual model was used in the first dialogue and 18 sessions in which the learned model was used.

This inequality means that if bias were occurring in the second dialogue, the learned model would be at a slight advantage in assessing the number of successful dialogues, since there were 22 sessions in which the learned model was used in the second dialogue and only 18 in which the manual model was used. While the average number of turns per second dialogue may have been influenced by the fact that the user was no longer an absolute novice, the results show that the percentage of successful dialogues for the learned model actually goes down from 88.9% to 81.8% from the first to the second dialogue, suggesting that bias did not positively affect the *success* of a dialogue in which the learned model was used.

At first glance, the results in Tables 8.1 and 8.2 suggest the utility of the learned decision model. This observation may, in fact, be true; however, the learned decision model is only one part of the dialogue management process. Therefore, these results do not necessarily indicate that the decision model is responsible for the better performance; analysis must be performed at a more detailed level. The next section will provide such an analysis at the level of the decision made in response to the conflict problem.

8.4 Assessment of the Conflict Problem

The conflict problem was defined earlier as the situation in which a newly hypothesized value for an attribute conflicts with the system's currently believed value for that attribute. In order to determine if employing the learned model contributed to the increased performance, as shown in the previous section, the actual decisions made according to the model must be analyzed. Tables 8.3 and 8.4 provide confusion matrices for the actions taken when the manual and learned models, respectively, were utilized. Overall, the learned model substantially outperformed the manual model with an accuracy of 84.1% compared to 52.1%.

The "ideal" reference action in each case was determined by the author, according to the definitions provided in Section 7.3.2. The predicted action in each case was

Reference→ *Prediction*↓	Ignore (52.1%)	Implicit (12.7%)	Explicit (5.6%)	Spelling (29.6%)
Ignore (53.5%)	25	4	2	7
Implicit (19.7%)	7	5	0	2
Explicit (7.1%)	0	0	0	5
Spelling (19.7%)	5	0	2	7

Table 8.3: Confusion matrix for the decision made using the manual decision model. An accuracy of 52.1% was obtained. The parenthesized percentages indicate the distribution of each action within the total set of either predicted or reference actions.

Reference→ *Prediction*↓	Ignore (52.3%)	Implicit (0.0%)	Explicit (0.0%)	Spelling (47.7%)
Ignore (50.0%)	19	0	0	3
Implicit (0.0%)	0	0	0	0
Explicit (0.0%)	0	0	0	0
Spelling (50.0%)	4	0	0	18

Table 8.4: Confusion matrix for the decision made using the learned decision model. An accuracy of 84.1% was obtained. The parenthesized percentages indicate the distribution of each action within the total set of either predicted or reference actions.

determined by either the manual or the learned decision model.

As observed with the simulated dialogues, the best performance for both models was achieved in the case of the *ignore* action. When the predicted action was *ignore*, the manual model was correct almost 66% (25/(25+4+2+7)) of the time, while the learned model showed over 86% (19/(19+0+0+3)) accuracy. The same argument applies in the case of real users as for simulated users: even though *ignore* is the default action for the learned rules, both models predicted this action in roughly 50% of their total responses to the conflict problem. This suggests that the better performance by the learned model is unlikely to be a consequence of the default learned rule.

Implicit and explicit confirmation were never predicted by the learned model. However, the manual model never predicted a truly ideal explicit confirmation, consistently confusing it with a spelling request. It also made many more confusions when *spelling request* was the ideal action. Such confusions can lower performance since, if a spelling request was the ideal action, this means the city in question was unknown to the recognizer and spelling would be the only way to acquire the user's intended value. Any confusions would only lead to superfluous turns until a spelling request was finally initiated. While the manual model misclassified 67% ((7+2+5)/(7+2+5+7)) of the actions that should have been a spelling request, the learned model only misclassified 14% ((3+0+0)/(3+0+0+18)).

The percentage of *spelling request* predictions increased from the simulated dialogues from 6.4% to 19.7% for the manual model, and from 14.1% to 50.0% for the learned model, indicating that the learned model was much more aggressive than the manual model in this respect. Perhaps there is something about the behavior of the real users that was not adequately modeled in the simulated users that contributes to these large differences in results. Admittedly, a rather conservative simulated user was employed, since the goal was not necessarily to accurately model a real user throughout a dialogue; rather, the goal was to provide a means by which very specific user behaviors could be specified and repeated for developing a corresponding dialogue strategy to handle such behavior. A more realistic simulated user might result in improved performance for the simulated dialogues.

A difference was also observed in the models' accuracies when a spelling request was predicted. In the simulated dialogues, the manual model was more accurate than the learned model. However, with the real user dialogues, the learned and manual models were correct 82% (18/(4+0+0+18)) and 50% (7/(5+0+2+7)) of the time, respectively. It was common for both models to predict a spelling request when *ignore* was ideal; however, this type of confusion is much less damaging than others. A spelling request, though the system already believe the correct value, would allow the user to clearly, albeit excessively, convey the intended city to the system. Such a confusion should not affect the acquisition of the user's intended value; however, it may add unnecessary turns to the dialogue.

A final noteworthy confusion is the manual model's tendency to predict *implicit confirm* when *ignore* was the ideal action. This confusion is certainly detrimental since the system consequently believed a spuriously hypothesized value, placing a burden on the user to initiate correction of the error. Users are often uncertain how

Reference → *Prediction* ↓	Compliant	Noncompliant
Compliant	109	3
Noncompliant	2	21

Table 8.5: Confusion matrix for the prediction of user compliance to a spelling request. An accuracy of 96.3% was obtained.

to phrase a correction that will be understood by the system. Ameliorating such an error can be further complicated by the fact that the system believes the incorrect value with relatively high confidence. Continued confusions of this type could lead to frustrated users who are likely to hang up before completing their tasks.

Analysis shows that the learned model performed better than the manual model, leading to better overall performance. A more detailed analysis, at the level of the conflict problem, shows better prediction accuracy and fewer detrimental confusions by the learned model, two reasons which are likely to have led to the increased overall performance.

8.5 Assessment of the Compliance Problem

The compliance problem was described earlier as determining whether or not the user provided a spelling as requested by the system. Section 7.5 described how a model was learned from simulated data, and how it attained perfect accuracy in predicting the compliance of simulated users. That same model was incorporated into the flight attribute data collection system to see how well it could predict the compliance of real users in response to a spelling request. Table 8.5 shows the confusion matrix for the compliance prediction. The results were very positive, with a 96.3% overall prediction accuracy.

The results show that the majority of users were compliant with a system spelling request. There were, however, various instances of noncompliance, which are even more interesting. It is the utterances of noncompliant users that reveal what the system developers may not have anticipated as potential user responses. The next section presents an analysis of the various responses provided by users, both compliant and noncompliant, when a spelling was requested by the system.

8.6 Assessment of Responses to Spelling Requests

One valuable and interesting part of this research was observing the users' reactions to the system's spelling requests. Each user's response depended on several factors, including the nature of the user and the number of previous spelling requests through the course of the dialogue. In the latter case, users tended to provide more than just a simple spelling in their responses to the system.

It must be noted that the way the system requests information from a user can have a great impact on the format of the user's response. For example, in earlier experiments, in which the system requested a speak-and-spell utterance from the user, the system included an example in its request to indicate exactly what was meant by *speak and spell*: "Please speak the arrival city, followed by its spelling, like this: *Boston, B O S T O N*." The user could then mimic the format of the example. In the system described in this chapter, the following wording was utilized by the system for the spelling request: "Please spell the name of your arrival city." No example was provided since simply requesting a spelling should be rather straightforward.

Each user provided some response following a spelling request, and quite a wide variety of responses was discovered. It should be noted that the results presented earlier in this chapter only considered those user sessions that were not determined to be invalid. This analysis of user responses, however, applies to all the dialogues collected, since the interest here is only in the type of an isolated response, not in how that response affects any performance metric. In total, there were 174 spelling requests and the same number of subsequent user responses. The following sections describe an array of user types and the typical responses provided in each case.

	Distribution
Compliant	77.6%
Partially Compliant	5.7%
Noncompliant	16.7%

Table 8.6: Distribution of user response types to a spelling request.

8.6.1 Compliant Users

Table 8.6 shows that, in the majority (77.6%) of requests, the user was compliant and provided exactly the spelling of the city name as requested. Users were successful at spelling city names ranging from as short as *Rush* to as long as *Ski Sundown Ski Area*. Occasionally, the user would insert the word, "space", between words in a city name containing multiple words. Such behavior was anticipated beforehand, however. The word, "space", was explicitly modeled in the spelling recognizer's vocabulary. A post-processing step simply removed any occurrences from the hypotheses, resulting in a sequence of letters only. Thus, the system was able to handle this situation.

8.6.2 Partially Compliant Users

Another response behavior was to include a spelling somewhere in the response, among one or more other words. This was called *partially compliant* and constituted 5.7% of the responses as shown in Table 8.6. Examples of this response include speak-and-spell (e.g., "Stockbridge S T O C K B R I D G E") as well as spell-and-speak (e.g., "W A I M E A Waimea"). When these responses occurred, the recognizer would hypothesize spurious letters for the spoken word, but typically correct hypotheses for the clean spelling. Consequently, the system was usually able to determine the intended city following a spelling-correction step.

8.6.3 Noncompliant Users

A noncompliant user is one who does not provide a spelling of the city name when requested by the system. The use of the word *noncompliant* is not meant to convey a negative connotation. In fact, in dialogue systems, one might say the mantra should be: *The user is always right.* The response provided by a user, whether it be expected by the system or not, is a real user response and, ideally, the system should be able to handle it. Therefore, if a user is noncompliant, as defined here, it is simply a classification and the burden should rest on the system to handle such a response in the future.

The responses observed by such noncompliant users include repeating the city name (e.g., "Trussville", "Potts Camp, Mississippi"), repeating the city name within a phrase (e.g., "I wanna go to Yamhill, Oregon."), repeating the city name as if the user were confused (e.g., "Uh, Redwood Valley, California."), obvious unwillingness to spell the city name (e.g., "I don't wanna spell the name."), or simply providing frustrated responses (e.g., "I just did!", "Why?"). As shown in Table 8.6, nearly 17% of the 174 user responses were noncompliant.

8.6.4 Out-of-Turn Users

An out-of-turn user is one who provides an utterance containing a spelling when a spelling has not just been requested by the system. Similar to the clarification of a noncompliant user, in Section 8.6.3, the use of the phrase, *out-of-turn*, is not meant to suggest a troublesome user. On the contrary, the responses provided by such users are, in fact, some of the most interesting, since their format and timing are frequently unanticipated by the developers. It is the task of the system to handle these responses. Fifteen responses from the data collection were classified as out-of-turn.

Figure 8-3 shows the out-of-turn responses and the system prompt that preceded them. While responses were given after several different system prompts, they most often occurred after the system request to "Please make a correction." This suggestion occurred as part of a longer prompt such as, "From Corpus Christi, Texas to Fargo, Arkansas. I have no knowledge of Fargo, Arkansas. Please make a correction." The hope was that the user would be encouraged to repeat the troublesome city name, after which the system would request its spelling. A valid speculation is that the users also foresaw this sequence, but decided to skip the intermediate step of repeating the city and to just go immediately to spelling the city.

—*Did you say to Pitts?* No, P O T T S C A M P.
—*Did you say to Redwood?* Uh, V A L L E Y.
—*Please make a correction.* To Parkdale P A R K D A L E. The city is W A I M E A. Waimea. To Redwood Valley R E D W O O D V A L L E Y California. Redwood Valley, California R E D W O O D V A L L E Y California. (I) did I just <sigh> R E D W O O D V A L L E Y. C A S S A D A G A. Stockbridge S T O C K B R I D G E. Not Houston, Stockbridge S T O C K B R I D G E. G R O V E C I T Y.
—*Please correct me or say, 'Goodbye.'* To S P R I N G M O U N T Pennsylvania. To Grove City G R O V E...
—*Sorry I'm not sure what you said.* Argonia A R G O N I A. The destination city is Argonia A R G O N I A.

Figure 8-3: Out-of-turn user responses and the corresponding system prompts.

In hindsight, the *Please make a correction* prompt was rather open-ended and almost invited creative responses from the user. The lesson from this observation is that once the system reveals to the user that it *can* accept spellings, the system should be prepared for a user spelling at *any* point in the dialogue. Such an algorithm to handle out-of-turn spelling behavior could be developed with the simulation/learning framework described in this thesis.

8.6.5 Confused Users

Confusion occurred between the system and the user when a partial city spelling by the user (due to a clipped utterance) matched a city name in the database. One example of this is when the user was trying to spell *Myrtle Point* but was cut off after spelling *Myrtle*. The system asked, "Did you say from Myrtle?", and the user replied, "Yes." This situation, in which the user replied, "Yes", occurred two other times with the city pairs, *Gilmanton–Gilmanton Iron Works*, and *Grove–Grove City*.

S1:	Please spell the name of your arrival city.
U1:	R E D W O O D <cutoff>
S2:	Did you say to Redwood?
U2:	uh V A L L E Y.
S3:	Please spell the name of your arrival city.
U3:	R E D W O O D V A L L E Y.
S4:	Did you say to Redwood Valley?
U4:	Yes, I did.

Figure 8-4: Example of a split-utterance spelling.

There was one more instance of the partial city spelling problem, but the user was more creative with his out-of-turn response. Figure 8-4 shows a portion of this dialogue. The system first requested a spelling from the user, who proceeded to spell *Redwood Valley* in U1, but the relatively long pause between *Redwood* and *Valley* caused the system to cut the user off prematurely. The system recognized the city, *Redwood*, from the spelling and explicitly confirmed it with the user in S2. In U2, the user was not sure what to say, since the system was half correct but it did not let him complete his spelling. Therefore, instead of saying, "Yes", the user continued to spell the second word of the city name. The system likely recognized garbage since U2 was an out-of-turn spelling, so the system requested the spelling again in S3. The user spelled the city again in U3, likely taking care not to leave a significant pause between *Redwood* and *Valley*, and was successful.

8.7 Notable Observations

8.7.1 User Compliance

One encouraging result from the real user dialogues is the observed level of user compliance with a spelling request. The destination city in every scenario was unknown to the recognizer, but the user was never told this fact. That nearly 78% of responses to the spelling requests were compliant is a good sign. This result indicates that users seemed comfortable in spelling a city name as requested by the system. In Chapter 3, we presented an analysis of user responses to a spelling request via the telephone keypad. In those cases, a mere 51% of requests were acknowledged with an attempt at a keypad spelling. A big difference between the two analyses is that the spoken-spelling analysis gives us a better idea as to why a user may have chosen not to provide a spelling. The noncompliant user responses indicate that some users simply felt more comfortable repeating the intended city name (perhaps thinking that the system *must* be able to recognize it), while other frustrated users simply refused to spell. In any case, these responses provide us with valuable information that can be used to develop more robust strategies in dealing with such input.

8.7.2 A "Spelled" Date

While the robust acquisition of the *date* attribute was not the major focus of this research, observation of user responses to the *date* request can still provide further information with which to develop the system. When one user was asked for her intended date, she responded by saying, "July zero three," to mean *July third*. Such a "spelling" of the date had not been anticipated and the system was unable to correctly recognize the utterance. Although the current implementation of the MERCURY dialogue system does support entry of the date value via the telephone keypad, the date "spelling" would require further adjustments to the grammar. This user behavior shows that while simulation can be extremely useful in discovering unanticipated situations, only real users can provide such creative ways of providing information. Having been observed, this pattern can be added to the list of ways to express a value for the *date* attribute.

8.7.3 The Learned Model at Work

One example in which the learned model clearly outperformed the manual model involved the user's intended city name, *Spring Mount*, and the hypothesized city name, *Springfield*. In the dialogue following the manual model, the recognizer repeatedly

U1:	A flight from Seattle, Washington, please.	
H1:	a flight from seattle to washington please	
S1:	From Seattle, Washington State. Did you say to Washington?	
U2:	To Cassadaga, Florida.	
H2:	beirut (conf: -11432) to dayton (conf: 6326) florida	
S2:	Please spell the name of your arrival city.	

Figure 8-5: Example of the learned model's ideal response to the conflict problem. U1 is the user's utterance, H1 is the system's hypothesis, and S1 is the system's response.

hypothesized *Springfield* with very high confidence every time the user actually said, *Spring Mount*. The high confidence kept the system from ever requesting a spelling for the city. The frustrated user eventually hung up. Admittedly, the system was deficient in this case in monitoring the number of times that the same value was hypothesized. A consistently hypothesized value, with which the user does not appear to be satisfied, should indicate that the system is continuing to make the same mistake. Nevertheless, in the dialogue that utilized the learned model, the system was able to identify a problem, and successfully acquired the user's intended city name after requesting its spelling.

There were several examples throughout the real user data collection demonstrating the decisions that had to be made by the dialogue manager. Figure 8-5 shows one such example. In U1, the user says she would like a flight from Seattle. In S1, the system implicitly confirms *Seattle, Washington State*, since that city is unique in the database. It also explicitly requests confirmation of *Washington* as the destination. The user provides a correction in U2, in which *Cassadaga* is unknown to the recognizer. In H2, the system hypothesizes *Dayton* as the destination city with high confidence. The manual model, in such a case, would implicitly confirm *Dayton*, due to its high confidence. The learned model, however, requests a spelling from the user, likely due to the high number of destination city hypotheses in the N-best list, despite the high confidence.

8.7.4 Simulated User Behavior

As mentioned previously, the simulated user was not necessarily intended to accurately model a real user's behavior throughout a dialogue; rather, there was a focused effort on configuring the simulated user to display behavior that was likely to give rise to error recovery opportunities. The simulated user could, however, be configured to exhibit behavior based on statistics observed in a corpus of real user dialogues. For example, we have observed in real user dialogues that a user sometimes unintentionally confirms or denies a given city name, and subsequently attempts to correct the mistake. The simulated user utilized in this research did not behave in such a manner; however, it could be configured as such, in which case, strategies could be developed to address this behavior.

8.7.5 Unrestricted Spelling

An important lesson learned from the analysis of the real user dialogues is that once the system reveals to a user that it is able to accept spellings of a word, the user is liable to provide a spelling at *any* point in the remainder of the dialogue. Therefore, the system should be prepared to handle a spelling at anytime. One strategy to address this issue would be for the specialized spelling recognizer to be active for the entire dialogue. Both the main and the specialized recognizer could produce hypotheses for every user utterance, requiring the dialogue manager to decide which recognizer to trust. The learned compliance model performed well at classifying spelled and noncompliant utterances, suggesting it could be beneficial in this proposed solution. However, the model would likely have to be retrained on a wider variety of utterances appearing throughout a dialogue. Such utterances might include partially compliant utterances that contain both a spelling, as well as other sentence fragments. If such mixed-media utterances could be identified, perhaps the spelling could be extracted and a recovery subdialogue could ensue to potentially acquire the intended value. The simulation/learning framework presented here could be used to develop the strategies for such a subdialogue.

8.8 Summary

This chapter described the flight attribute data collection system, a telephone-based dialogue system deployed in order to test the performance of two learned decision models with real users. The user's goal was to get the system to understand a source city and state, destination city and state, and date. The main point of interest was the *conflict problem*, a situation in which a newly hypothesized city name conflicts with the value currently believed by the system. Each user was asked to participate in two dialogues, one in which the system utilized a manually specified rule model to determine which action to take in the case of the conflict problem, and one in which a learned model, based on simulated data, was used to make the decision. The purpose was to assess the benefits, if any, of the learned model over the manual model.

A total of 40 user sessions with 30 unique users was analyzed. The results showed increased overall performance in those dialogues that utilized the learned model, both in terms of attribute accuracy and the number of turns per dialogue. A detailed analysis at the level of the conflict problem showed that the learned model attained much greater accuracy than the manual model in predicting the "ideal" actions, as well as in making fewer detrimental confusions in those predictions.

An additional assessment was performed on a model, learned from simulated data, that predicted the compliance of a user in response to a system spelling request. The model performed with very high accuracy, confirming its utility for this task.

In general, the results obtained in the dialogues with real users are comparable to those obtained with the simulated user in Chapter 7. In some cases, better performance was observed with real users, which is possibly a consequence of an inadequate simulated user model. More realistic behavior on the part of the simulated user could produce results more similar to those obtained with real users.

An analysis of real user responses obtained in this data collection was performed, showing how they can be valuable in enhancing a system to be more robust to a wide variety of user inputs.

Some notable observations arising from the real user dialogues were then discussed, suggesting the utility of the spelling recovery mechanism, as well as the simulation/learning framework for strategy development.

The next chapter will conclude this thesis by summarizing our work, making special note of the contributions of this research to the field of dialogue management. Possible future directions using ideas from this thesis will also be discussed.

Chapter 9

Contributions and Future Work

Spoken dialogue systems possess great potential for providing conversational access to information sources. However, such systems are only useful if they can understand most of the content words in each utterance. Many users have been aggravated by conversational systems that repeatedly hypothesize the same incorrect words, often due to ignorance of critical words the user is speaking. The dialogue manager is often of little or no help, suggesting, "I did not understand. Please rephrase your query.", or hypothesizing completely unintended values. The system must be able to recognize complications such as misrecognitions, repetitive mistakes, and out-of-vocabulary words, and to react appropriately. A more successful interaction can be achieved if the dialogue manager is able to work with the user to resolve an error in the course of the dialogue.

9.1 Contributions

This thesis presents a framework for developing attribute acquisition strategies in a spoken dialogue system. The contributions are concisely stated as follows:

- We present a novel method of utterance generation for user simulation. In this technique, speech segments of real users from an existing corpus of in-domain utterances are combined with synthesized speech segments to create a wide variety of realistic, simulated utterances.

- We present an attribute-centric model of generic dialogue management, which allows the specification of interaction properties on a per-attribute basis. Generic dialogue strategies and expert knowledge components supplement the model. Enabling a developer to access these components facilitates the exploration and development of dialogue strategies, especially for error recovery, via simulation to address unanticipated, and hence potentially problematic, user behaviors.

- We explore a domain-independent spelling mechanism in a task-oriented dialogue system to acquire the names of cities unknown to the system. In doing so, we reduce the degree of unpredictability that may arise when a user presents unknown words to a system.

- We employ a learning mechanism to develop dialogue strategies based on simulated data, and to acquire more robust rules than those resulting from manual specification. We use a simulated user to generate a large number of dialogues, from which we obtain data to learn models to address the *conflict problem* and the *compliance problem*. Though developed and tested on simulated data, the models perform well when applied in dialogues with real users.

The main points of each contribution are discussed in the remainder of this section, which will be followed by a discussion of the potential reusability of various components developed throughout this thesis. This chapter will end with a discussion of potential future work.

9.1.1 Novel Method of Utterance Generation

The majority of current work on user simulation models a simulated user's input at the intention level; however, we do so at the speech level. This approach provides the system with an ample number of opportunities to recover from errors, which are a natural consequence of speech recognition and natural language understanding. In order to control the user input to our specifications, we developed a novel method of utterance generation that produces a wide variety of simulated utterances, while maintaining the realistic nature of real user utterances. They may also be configured to contain specific lexical items, such as foreign or unknown words, that might give rise to errors in a dialogue. These properties are attained by combining speech segments from an existing corpus with synthesized speech for the lexical items that we wish to assess.

This method was beneficial in our development of error handling strategies for unknown words. While the problem of unknown words is significant in spoken dialogue systems, it may only occur infrequently depending on the domain, the user, and the user's level of experience in the given domain. However, when an unknown word or conflicting hypothesis is encountered, it should be handled so as to facilitate a smooth continuation of the dialogue. We configured the simulated utterances to contain city names that were unknown to the system, resulting in several opportunities to test our error recovery strategies. We also used this generation mechanism to create spelling utterances, which were required for successful error recovery. The existing corpus of real user utterances did not contain any spellings, so the spelling utterances were completely synthesized. Thus, this method enabled us to thoroughly exercise our spelling strategy, using only existing corpora and a speech synthesizer, requiring no input from real users.

9.1.2 Attribute-Centric Dialogue Management

Developing the dialogue manager for a new spoken dialogue system requires a significant amount of time and effort if done from scratch. Such development can be facilitated with preexisting components that handle shared functionality across domains. In an information retrieval task, where the system collects a set of attribute values and then provides some information to a user, the accurate acquisition of those attribute values is essential in order for the user to complete his or her goal. The model that we present allows a developer to specify a variety of interaction properties on a per-attribute basis. Such properties can include, for example, whether a value should be implicitly or explicitly confirmed, whether a request for its entry via the telephone keypad should be invoked, or whether a spoken spelling should be requested. Such spelling techniques can be utilized to acquire information about words that the recognizer may not explicitly model in its vocabulary, at least enabling the system to tell the user that it does not know the intended word.

The model provides an array of such generic dialogue strategies for the developer's use, without requiring him or her to know the complexity of the underlying functionality. Such control can be especially useful in a simulated dialogue environment. In Chapter 6, we explored various configurations of the simulated user and the dialogue manager, observing the effectiveness of certain strategies in handling a specific user behavior. Modifying the strategies used for each attribute and adjusting the user's behavior were facilitated by this method of attribute-centric dialogue control. These experiments were also performed without any input from real users, thus providing an efficient debugging and development mechanism for strategies prior to deployment to real users.

9.1.3 Unknown Words and Spelling

The inability of a system to acquire a user's intended value, which is unknown to the system's recognizer, can often lead to unpredictable behavior in the conversation by the dialogue manager, as well as by the user. Providing a method, such as spelling, can allow the user to clearly convey the intended value to the system, which can simply tell the user that it does not know the city, or perhaps do something more complex.

After observing that spelling via the telephone keypad did not seem to be a very successful recovery mechanism in MERCURY, we decided to explore the utility of a speak-and-spell technique. It had been used earlier in a personal name enrollment task [16, 70]. We employed a basic simulated user to continue real, problematic dialogues from the point of a keypad request, by providing a speak-and-spell utterance to the system, in lieu of a keypad entry. Results showed that nearly 89% of the spelled city names were correctly identified by the system, indicating that the strategy may also work well for real users.

In the spelling mechanism, a match to a spelled city name is first sought in the system's hypotheses from the previous user utterance, and then in an off-line database of nearly 20,000 city names. While it would be possible to explicitly represent all possible city names in the recognizer's vocabulary, the consequent increase in confusability among words would likely cause a reduction in accuracy, or an unacceptably high number of disambiguation subdialogues. Therefore, we chose to adopt an approach involving spelling.

The spelling strategy was briefly available to real users via the JUPITER weather information system for a period of four days. We discovered that users sometimes did not provide a spelling as requested. Since the system activated a specialized spelling recognizer upon a spelling request, and deactivated the main recognizer, such *noncompliant* responses caused problems. This issue prompted us to make a slight modification to the system architecture, in which both the main recognizer and the spelling recognizer were active for the user response following a spelling request. The dialogue manager then decided which recognizer to trust based on their hypotheses.

Following further development of the spelling strategy, we incorporated it into the MERCURY flight domain, where the task was to get the system to understand source, destination, and date. Simulated dialogues showed that the spelling strategy was essential to obtain unknown cities in a dialogue scenario. However, they also showed that, when the city names were known to the system, the spelling strategy was not necessary to achieve good attribute accuracy.

Further experiments with real users revealed a variety of interesting user responses, which included providing a spelling when the system had not requested one. This situation caused a problem since the spelling recognizer was only active following a spelling request. We discovered many examples of user behavior, which could be addressed by developing strategies utilizing the simulation/learning framework described in this thesis.

9.1.4 Simulation, Learning, and Real Users

In our development of error recovery strategies, we encountered two situations for which manually specified rules were either inadequate or difficult to devise. In the *conflict problem*, a newly hypothesized value conflicts with the value currently believed by the system for a given attribute. In Chapter 7, we defined a set of "ideal" actions to be taken by the dialogue manager in this situation, based on the system's current beliefs and the user's intentions. A simulated user is beneficial in such a situation since the system can know the user's true intentions during the dialogue, and we can determine what the ideal action *would be*. We thus employed user simulation to generate a large number of dialogues, from which we collected data to learn a model to predict the ideal action given a set of feature values. If such data were obtained from real users, the dialogues would require hand-labeling of the users' true intentions before a model could be learned.

The learned model was tested on both a simulated user, as well as real users, in the flight attribute data collection system, as described in Chapter 8. In both cases, the learned model performed better than a baseline model at both a high level and at the level of the decision made in response to the conflict problem. The learned model attained decision accuracies of 76.1% and 84.1% in the simulated and real user dialogues, respectively, while the manual model achieved accuracies of 61.7% and 52.1%. In addition, the learned model made fewer detrimental confusions, which likely contributed to its better performance.

The reduction in performance for the manual model from the simulated to the real user dialogues is potentially due to an inadequate simulated user model. The model was intended to focus on specific user behaviors, so strategies could be developed to handle them; in other words, the user model was not trained on real user data. More realistic behavior on the part of the simulated user could produce results more similar to those obtained with real users. Such a possibility could be explored in future work.

In the *compliance problem*, the dialogue manager must decide whether a user was compliant with the system's request to spell the value for a given attribute. In previous work, this decision was made heuristically, based on the number of letters returned in the speak-and-spell recognizer's hypothesis. This strategy, however, often led to the misclassification of spellings of short city names. In this thesis, however, we employed user simulation to generate a large number of spelled utterances, as well as noncompliant utterances. We then used the simulated data to learn a prediction model for user compliance. The model attained perfect accuracy when tested on a simulated user, and 96.3% accuracy when tested in real user dialogues, indicating the utility of the learned model for this task.

9.2 Reusability of Components

The strategies and tools developed in this thesis can be applied to other domains; that is, they can be reused. In this section, we briefly discuss the reusability of the novel method of utterance generation, the attribute-centric dialogue control mechanism, the spelling strategy, and the learned models.

9.2.1 Utterance Generation

The novel method of utterance generation for user simulation can be utilized for another domain. If the task in a new domain were to be very different in nature than information retrieval, a new corpus of utterance templates would have to be generated to realistically model the structure of utterances for that domain. If, however, the new domain focused on information retrieval, it is possible that the existing utterance templates could be reused. For example, if the new domain handled train reservations, the following template from the flight domain could simply be reused:

I am leaving from <city0> to go to <city1> on <date>.

Other utterances, however, would need to be modified slightly. For example, if the semantic concept, *flights*, in a) below were replaced by *trains*, as in b), the modified utterance could then be used for simulation in a train reservation domain.

a) I am interested in *flights* from <city0> to <city1>.

b) I am interested in *trains* from <city0> to <city1>.

Additionally, if the word, *trains*, were not present in the vocabulary of the corpus of real user utterances, that lexical item would need to be synthesized.

In general, however, this generation mechanism would be easiest applied to a domain for which a corpus of real user utterances already exists. The technique would then be useful for establishing a user simulation framework, which in turn could be used to develop strategies that would enhance the existing system's capabilities.

9.2.2 Attribute-Centric Dialogue Management

The major goal of the attribute-centric dialogue management framework was that it be portable to other domains concerned with an information retrieval task. The attribute model requires the specification of a domain's attributes and the interaction properties associated with each one. The set of generic dialogue strategies, which includes the implicit or explicit confirmation of values, as well as the spelling mechanism, is immediately available for any domain. In addition, common knowledge, such as geography and dates, is available for any domain by way of the model's expert components.

9.2.3 Spelling Strategy

The spelling error recovery algorithm was designed to be domain-independent. As one of the generic dialogue strategies in the attribute-centric model mentioned above, it is meant to serve as a plug-in module for any domain, whether it be a new domain or an existing one. Spelling would be a beneficial capability in any domain that deals with open vocabulary classes, such as names of places and people. This capability could also be useful in domains in which foreign-language names or words commonly appear, such as a personnel directory or a restaurant information system.

9.2.4 Learned Models

It is possible that the model learned to address the conflict problem could be reused in another domain. The features utilized for learning are all domain-independent, being based on the N-best list from the recognizer, as well as on generic attribute acquisition properties spanning multiple turns. If the incidence of conflicting values in a new domain did not roughly match that of our flight reservation domain, however, a new model would most likely have to be learned for better accuracy.

The compliance model, on the other hand, is based solely on features from the recognition of the user's response to a spelling request. Since spelling is a domain-independent action, this model would be likely to demonstrate good performance in any domain where spelling could be a useful strategy. Since our experiments with real users show that noncompliance occurs rather frequently, such a compliance predictor seems to be an essential component for any system in which user spelling is to be allowed.

9.3 Possible Future Work

This final section describes work that could be performed in the future, building on the work conducted in this thesis.

9.3.1 Simulated User Model

As mentioned earlier, our simulated user model is not based on statistics from a corpus of real user dialogues. It is possible that the discrepancies seen in the assessment of the conflict problem between the simulated user and real users are due to an inadequate simulated user model. While making the user model as realistic as possible was not our intention, future work could entail training a user model based on an existing corpus of in-domain, real user dialogues. Such a model could serve to enhance our ability to develop dialogue strategies, since it could enable us to dynamically model realistic user behavior, such as when a user changes his or her mind and wants to fly to a different city.

9.3.2 Attribute-Centric Dialogue Management

We hope that the model of attribute-centric dialogue management will be useful for other domains, in which large databases of information or open vocabulary sets are involved. Such domains may include a personnel directory system, which could accept a person's first or last name and retrieve a phone number or office location. Spelling would likely be useful for such a task, since the pronunciation of last names and foreign names often varies among different speakers. The model might also be applied to other information retrieval domains, such as movie information and music collection access.

9.3.3 Addressing the Real User Responses

The responses obtained from real users in the flight attribute data collection are valuable, not only for assessing the utility of the learned decision models, but also for providing real examples of user behavior when errors are encountered in an attribute acquisition dialogue. Our approach of simulation and learning for dialogue strategy development can be used to address these unanticipated behaviors. For example, we observed behavior in which a user spelled a multi-word city name across two utterances. In order to develop a strategy to handle such input, we could configure the simulated user to behave in a similar manner, then run simulated dialogues to develop and assess a strategy to handle this issue.

Some users would occasionally provide mixed utterances, such as "speak-and-spell" or "spell-and-speak" responses, when a simple spelling was requested. We could learn a model to distinguish between simple spellings, mixed utterances, and noncompliant utterances, which contained no spelling at all. If a mixed utterance were identified, the letters could be extracted from the utterance and processed.

Another common user response was to spell a city name when the system had not requested a spelling. As discussed in Chapter 8, we could activate both the main domain recognizer and the specialized spelling recognizer for the entire duration of the dialogue. This implementation would require the dialogue manager to decide which recognizer to trust after every user utterance. A new compliance model would likely have to be learned in such a case, since there would be a much wider variety of "noncompliant" utterances to distinguish from valid spellings. We would also need to observe if such an implementation would affect the system's real-time performance.

9.3.4 Improving the Learned Conflict Problem Model

While our experiments showed that the learned model performed significantly better than the manual model upon encountering the conflict problem, there is still room for improvement. The limited amount of data obtained from the data collection was not sufficient to fully explore the abilities of the learned model in predicting *implicit confirm* as the ideal action. Ideally, more data could be collected and the learned model's performance could be assessed. However, the simulated tests revealed a relatively low accuracy of the learned model in predicting *implicit confirm*. It may be more advantageous to first explore additional features that could result in better prediction accuracy.

We may want to reevaluate our definition of *explicit confirm*, in terms of an "ideal" action. The learned model did not even include this action in its rule set, while the manual model rarely predicted it as the ideal action. These results suggest that the conditions under which *explicit confirm* was defined as ideal are simply infrequent. We said that a value should be explicitly confirmed if the newly hypothesized city is *not* the user's intended city, but the user's intended city is *known*. These observations seem to attest to the good performance of the recognizer, since it indicates that, when a user's intended city was, in fact, known by the system, the recognizer did not tend to confuse it with another known city. This could also, however, be due to the possibility that there may not be a high degree of confusability among city names in the MERCURY domain. In any case, further experiments could be performed in a system where the city names in the recognizer's vocabulary are more confusable.

9.3.5 Learning Models for Other Problems

The development framework presented in this thesis could be applied to problems other than the conflict and compliance problems. The simulation aspect is especially useful for problems in which the "ideal" action of the dialogue manager is partially based on information about the user or the user's true intentions. Such information is available in a simulated environment. Therefore, simulation can be used to automatically obtain data to predict an action, whereas hand-labeling would be required following a data collection from real users.

Other problems, such as the compliance problem, can benefit just from the system's ability to generate a large number of examples, hence creating a great deal of data, from which a decision model may be learned. Another problem of this type would occur in a multilingual dialogue system, in which the system would need to distinguish between utterances in multiple languages. A simulated user could be used to generate many example utterances in each language, as well as to automatically tag each utterance with the language in which it is realized. A model could then be learned from this simulated data to predict the source language for a new example utterance.

Bibliography

[1] *Travelocity*. At <http://www.travelocity.com> on 9 May 2006.

[2] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. An Architecture for a Generic Dialogue Shell. *Natural Language Engineering*, 6(3–4):213–228, 2000.

[3] Christopher Baber. Developing Interactive Speech Technology. In Christopher Baber and Janet Noyes, editors, *Interactive Speech Technology: Human Factors Issues in the Application of Speech Input/Output to Computers*, pages 1–18. Taylor & Francis, London, 1993.

[4] Lauren Baptist. GENESIS-II: A Language Generation Module for Conversational Systems. Master's thesis, Massachusetts Institute of Technology, 2000.

[5] Lauren Baptist and Stephanie Seneff. GENESIS-II: A Versatile System for Language Generation in Conversational System Applications. In *Proc. ICSLP*, pages 271–274, Beijing, China, October 2000.

[6] Josef G. Bauer and Jochen Junkawitsch. Accurate Recognition of City Names with Spelling as a Fall Back Strategy. In *Proc. European Conference on Speech Communication and Technology*, volume 1, pages 263–266, Budapest, Hungary, 1999.

[7] Issam Bazzi and James Glass. A Multi-Class Approach for Modelling Out-of-Vocabulary Words. In *Proc. ICSLP*, pages 1613–1616, Denver, 2002.

[8] Niels Ole Bernsen, Hans Dybkjær, and Laila Dybkjær. *Designing Interactive Speech Systems: From First Ideas to User Testing.* Springer-Verlag, London, 1998.

[9] Dan Bohus. *RoomLine: An Automated Conference Room Reservation Assistant*, 2004. At <http://www.cs.cmu.edu/~dbohus/RoomLine/> on 12 April 2006.

[10] Dan Bohus and Alex Rudnicky. RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. In *Proc. European Conference on Speech Communication and Technology*, pages 597–600, Geneva, Switzerland, 2003.

[11] Dan Bohus and Alexander I. Rudnicky. A Principled Approach for Rejection Threshold Optimization in Spoken Dialog Systems. In *Proc. Interspeech*, pages 2781–2784, Lisbon, Portugal, 2005.

[12] Dan Bohus and Alexander I. Rudnicky. Constructing Accurate Beliefs in Spoken Dialog Systems. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 272–277, San Juan, Puerto Rico, 2005.

[13] Trung H. Bui and Martin Rajman. Rapid Dialogue Prototyping Methodology. Technical Report No. 200401 IC/2004/01, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, January 2004.

[14] Trung H. Bui, Martin Rajman, and Miroslav Melichar. Rapid Dialogue Prototyping Methodology. In *Proc. 7th International Conference on Text, Speech, and Dialogue (TSD 2004)*, Brno, Czech Republic, September 2004.

[15] Grace Chung. Developing a Flexible Spoken Dialog System Using Simulation. In *Proc. ACL*, pages 63–70, Barcelona, Spain, 2004.

[16] Grace Chung, Stephanie Seneff, and Chao Wang. Automatic Acquisition of Names Using Speak and Spell Mode in Spoken Dialogue Systems. In *Proc. HLT-NAACL*, pages 32–39, 2003.

[17] Grace Chung, Stephanie Seneff, Chao Wang, and I. Lee Hetherington. A Dynamic Vocabulary Spoken Dialogue Interface. In *Proc. Interspeech*, Jeju Island, Korea, 2004.

[18] William W. Cohen. Fast Effective Rule Induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, California, 1995.

[19] Heriberto Cuayáhuitl, Steve Renals, Oliver Lemon, and Hiroshi Shimodaira. Human-Computer Dialogue Simulation Using Hidden Markov Models. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 290–295, San Juan, Puerto Rico, 2005.

[20] James R. Davis. Let Your Fingers Do the Spelling: Implicit Disambiguation of Words Spelled with the Telephone Keypad. *Avios Journal*, 9:57–66, 1991.

[21] DISC. *Dialogue Management Introduction*, 2000. From the DISC Dialogue Engineering Model. At <http://www.disc2.dk/slds/DMintro.html> on 12 April 2006.

[22] Wieland Eckert, Esther Levin, and Roberto Pieraccini. User Modeling for Spoken Dialogue System Evaluation. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 80–87, Santa Barbara, California, 1997.

[23] Wieland Eckert, Esther Levin, and Roberto Pieraccini. Automatic Evaluation of Spoken Dialogue Systems. In *Proc. Twente Workshop on Language Technology: Formal Semantics and Pragmatics of Dialogue*, pages 99–110, Enschede, The Netherlands, 1998.

[24] George Ferguson and James Allen. TRIPS: An Integrated Intelligent Problem-Solving Assistant. In *Proc. of the National Conference on Artificial Intelligence*, pages 567–572, Madison, Wisconsin, July 1998. American Association for Artificial Intelligence.

[25] George Ferguson, James Allen, Brad Miller, and Eric Ringger. The Design and Implementation of the TRAINS-96 System: A Prototype

Mixed-Initiative Planning Assistant. TRAINS Technical Note 96-5, Computer Science Dept., University of Rochester, October 1996. At <http://www.cs.rochester.edu/research/cisd/pubs> on 12 April 2006.

[26] Ed Filisko. A Context Resolution Server for the GALAXY Conversational Systems. Master's thesis, Massachusetts Institute of Technology, 2002.

[27] Ed Filisko and Stephanie Seneff. A Context Resolution Server for the GALAXY Conversational Systems. In *Proc. European Conference on Speech Communication and Technology*, pages 197–200, Geneva, Switzerland, September 2003.

[28] Ed Filisko and Stephanie Seneff. Error Detection and Recovery in Spoken Dialogue Systems. In *Proc. HLT/NAACL Workshop: Spoken Language Understanding for Conversational Systems*, pages 31–38, Boston, Massachusetts, 2004.

[29] Ed Filisko and Stephanie Seneff. Developing City Name Acquisition Strategies in Spoken Dialogue Systems Via User Simulation. In *Proc. 6th SIGdial Workshop*, pages 144–155, Lisbon, Portugal, 2005.

[30] Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning User Simulations for Information State Update Dialogue Systems. In *Proc. Interspeech*, pages 893–896, Lisbon, Portugal, 2005.

[31] Egidio Giachin. Spoken Language Dialogue. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue, editors, *Survey of the State of the Art in Human Language Technology*, chapter 6.4. Cambridge University Press, 1996. At <http://cslu.cse.ogi.edu/HLTsurvey/ch6node6.html> on 12 April 2006.

[32] James Glass. A Probabilistic Framework for Segment-Based Speech Recognition. *Computer Speech and Language*, pages 137–152, 2003.

[33] James Glass, Jane Chang, and Michael McCandless. A Probabilistic Framework for Feature-based Speech Recognition. In *Proc. ICSLP*, pages 2277–2280, Philadelphia, 1996.

[34] Allen Gorin, Giuseppe Riccardi, and Jerry Wright. How May I Help You? *Speech Communication*, 23:113–127, 1997.

[35] Paul Grice. *Studies in the Way of Words.* Harvard University Press, Cambridge, Massachusetts, 1989.

[36] Joakim Gustafson, Nikolaj Lindberg, and Magnus Lundeberg. The August Spoken Dialogue System. In *Proc. European Conference on Speech Communication and Technology*, 1999.

[37] William I. Hallahan. DECtalk Software: Text-to-Speech Technology and Implementation. *Digital Technical Journal*, 7(4):5–19, 1995. At <http://www.hpl.hp.com/hpjournal/dtj/vol7num4/vol7num4art1.pdf> on 12 April 2006.

[38] 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.

[39] James Henderson, Oliver Lemon, and Kallirroi Georgila. Hybrid Reinforcement/Supervised Learning for Dialogue Policies from COMMUNICATOR Data. In *Proc. 4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 68–75, Edinburgh, Scotland, 2005.

[40] Ryuichiro Higashinaka, Katsuhito Sudoh, and Mikio Nakano. Incorporating Discourse Features into Confidence Scoring of Intention Recognition Results in Spoken Dialogue Systems. In *Proc. ICASSP*, pages 25–28, Philadelphia, 2005.

[41] Julia Hirschberg, Diane Litman, and Marc Swerts. Identifying User Corrections Automatically in Spoken Dialogue Systems. In *Proc. ACL*, pages 1–8, Pittsburgh, 2001.

[42] Daniel Jurafsky and James H. Martin. *Speech and Language Processing.* Prentice Hall, Inc., Englewood, New Jersey, 2000.

[43] Candace Kamm, Diane Litman, and Marilyn A. Walker. From Novice to Expert: The Effect of Tutorials on User Expertise with Spoken Dialogue Systems. In *Proc. ICSLP*, pages 1211–1214, Sydney, Australia, 1998.

[44] Emiel Krahmer, Marc Swerts, Mariet Theune, and Mieke Weegels. Error Detection in Spoken Human-Machine Interaction. *International Journal of Speech Technology*, 4(1):19–30, 2001.

[45] Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. The WITAS Multi-Modal Dialogue System I. In *Proc. European Conference on Speech Communication and Technology*, pages 1559–1562, Aalborg, Denmark, September 2001.

[46] Piroska Lendvai, Antal van den Bosch, Emiel Krahmer, and Marc Swerts. Multi-Feature Error Detection in Spoken Dialogue Systems. In Mariet Theune, Anton Nijholt, and Hendri Hondorp, editors, *Proc. Computational Linguistics in the Netherlands 2001: Selected Papers from the 12th CLIN Meeting*, pages 163–178, Twente, The Netherlands, 2002.

[47] Esther Levin, Roberto Pieraccini, and Wieland Eckert. A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23, January 2000.

[48] Gina-Anne Levow. Characterizing and Recognizing Spoken Corrections in Human-Computer Dialogue. In *COLING-ACL*, pages 736–742, Montréal, Québec, August 1998.

[49] Diane Litman and Shimei Pan. Predicting and Adapting to Poor Speech Recognition in a Spoken Dialogue System. In *Proc. AAAI*, pages 722–728, Austin, Texas, 2000.

[50] Diane J. Litman, Marilyn A. Walker, and Michael S. Kearns. Automatic Detection of Poor Speech Recognition at the Dialogue Level. In *Proc. ACL*, pages 309–316, College Park, Maryland, 1999.

[51] Ramón López-Cózar, Ángel de la Torre, José C. Segura, and Antonio J. Rubio. Assessment of Dialogue Systems by Means of a New Simulation Technique. *Speech Communication*, 40:387–407, 2003.

[52] I. Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. LetterWise: Prefix-based Disambiguation for Mobile Text Input. In *Proc. Symposium on User Interface Software and Technology*, pages 111–120, Orlando, Florida, 2001.

[53] Matt Marx and Chris Schmandt. Putting People First: Specifying Proper Names in Speech Interfaces. In *Proc. Symposium on User Interface Software and Technology*, pages 29–37, Marina del Rey, California, 1994.

[54] Michael McTear. Spoken Dialogue Technology: Enabling the Conversational User Interface. *ACM Computing Surveys (CSUR)*, 34(1):90–169, March 2002.

[55] Ian M. O'Neill and Michael F. McTear. Object-Oriented Modelling of Spoken Language Dialogue Systems. *Natural Language Engineering*, 6(3–4):341–362, September 2000.

[56] Daniela Oria and Akos Vetek. Robust Algorithms and Interaction Strategies for Voice Spelling. In *Proc. Interspeech*, pages 833–836, Lisbon, Portugal, September 2005.

[57] Sharon Oviatt, Jon Bernard, and Gina-Anne Levow. Linguistic Adaptations During Spoken and Multimodal Error Resolution. *Language and Speech*, 41(3-4):419–442, 1998.

[58] Olivier Pietquin. *A Framework for Unsupervised Learning of Dialogue Strategies.* PhD thesis, Faculté Polytechnique de Mons, 2004.

[59] Joseph Polifroni and Grace Chung. Promoting Portability in Dialogue Management. In *Proc. ICSLP*, pages 2721–2724, Denver, September 2002.

[60] Antoine Raux, Brian Langner, Dan Bohus, Alan W. Black, and Maxine Eskenazi. Let's Go Public! Taking a Spoken Dialog System to the Real World. In *Proc. Interspeech*, pages 885–888, Lisbon, Portugal, 2005.

[61] Jost Schatzmann, Kallirroi Georgila, and Steve Young. Quantitative Evaluation of User Simulation Techniques for Spoken Dialogue Systems. In *Proc. 6th SIGdial Workshop*, pages 45–54, Lisbon, Portugal, 2005.

[62] Jost Schatzmann, Matthew N. Stuttle, Karl Weilhammer, and Steve Young. Effects of the User Model on Simulation-Based Learning of Dialogue Strategies. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 220–225, San Juan, Puerto Rico, 2005.

[63] Konrad Scheffler and Steve Young. Probabilistic Simulation of Human-Machine Dialogues. In *Proc. ICASSP*, pages 1217–1220, Istanbul, 2000.

[64] Konrad Scheffler and Steve Young. Corpus-Based Dialogue Simulation for Automatic Strategy Learning and Evaluation. In *Proc. NAACL Workshop on Adaptation in Dialogue Systems*, Pittsburgh, 2001.

[65] Konrad Scheffler and Steve Young. Automatic Learning of Dialogue Strategy Using Dialogue Simulation and Reinforcement Learning. In *Proc. Human Language Technology (HLT) Conference*, San Diego, California, 2002.

[66] Hauke Schramm, Bernd Rueber, and Andreas Kellner. Strategies for Name Recognition in Automatic Directory Assistance Systems. *Speech Communication*, 31(4):329–338, 2000.

[67] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.

[68] Stephanie Seneff. TINA: A Natural Language System for Spoken Language Applications. *Computational Linguistics*, 18(1):61–86, 1992.

[69] Stephanie Seneff. Response Planning and Generation in the MERCURY Flight Reservation System. *Computer Speech and Language*, 16:283–312, 2002.

[70] Stephanie Seneff, Grace Chung, and Chao Wang. Empowering End Users to Personalize Dialogue Systems through Spoken Interaction. In *Proc. European Conference on Speech Communication and Technology*, pages 749–752, 2003.

[71] Stephanie Seneff, Chian Chuu, and D. Scott Cyphers. ORION: From On-line Interaction to Off-line Delegation. In *Proc. ICSLP*, pages 142–145, Beijing, China, October 2000.

[72] Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. GALAXY-II: A Reference Architecture for Conversational System Development. In *Proc. ICSLP*, Sydney, Australia, November 1998.

[73] Stephanie Seneff, Raymond Lau, and Helen Meng. ANGIE: A New Framework for Speech Analysis Based on Morpho-phonological Modelling. In *Proc. ICSLP*, pages 110–113, 1996.

[74] Stephanie Seneff and Joseph Polifroni. Dialogue Management in the MERCURY Flight Reservation System. In *Proc. ANLP-NAACL, Satellite Dialogue Workshop*, pages 1–6, Seattle, May 2000.

[75] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research*, 16:105–133, 2002.

[76] Ronnie W. Smith and D. Richard Hipp. *Spoken Natural Language Dialog Systems: A Practical Approach*. Oxford University Press, New York, 1994.

[77] Amanda Stent, John Dowding, Jean Mark Gawron, Elizabeth O. Bratt, and Robert Moore. The CommandTalk Spoken Dialogue System. In *Proc. ACL*, pages 183–190, College Park, Maryland, June 1999.

[78] Stephen Sutton, Ronald Cole, Jacques de Villiers, Johan Schalkwyk, Pieter Vermeulen, Mike Macon, Yonghong Yan, Ed Kaiser, Brian Rundle, Khaldoun Shobaki, Paul Hosom, Alex Kain, Johan Wouters, Dominic Massaro, and Michael Cohen. Universal Speech Tools: The CSLU Toolkit. In *Proc. ICSLP*, pages 3221–3224, Sydney, Australia, November 1998.

[79] Markku Turunen, Esa-Pekka Salonen, Mikko Hartikainen, Jaakko Hakulinen, William Black, Allan Ramsay, Adam Funk, Andrew Conroy, Paul Thompson, Mark Stairmand, Kristiina Jokinen, Jyrki Rissanen, Kari Kanto, Antti Kerminen, Björn Gambäck, Magnus Sahlgren, Fredrik Olsson, Maria Cheadle, Preben Hansen, and Stina Nylander. AthosMail: a Multilingual Adaptive Spoken Dialogue System for the E-mail Domain. In *Proc. COLING Workshop on Robust and Adaptive Information Processing for Mobile Speech Interfaces*, pages 77–86, Geneva, Switzerland, August 2004.

[80] Antal van den Bosch, Emiel Krahmer, and Marc Swerts. Detecting Problematic Turns in Human-Machine Interactions: Rule-Induction Versus Memory-Based Learning Approaches. In *Proc. ACL*, Toulouse, 2001.

[81] Marilyn Walker, John Aberdeen, Julie Boland, Elizabeth Bratt, John Garofolo, Lynette Hirschman, Audrey Le, Sungbok Lee, Shrikanth Narayanan, Kishore Papineni, Bryan Pellom, Joseph Polifroni, Alex Potamianos, Payal Prabhu, Alex Rudnicky, Gregory Sanders, Stephanie Seneff, David Stallard, and Steve Whittaker. DARPA Communicator Dialog Travel Planning Systems: The June 2000 Data Collection. In *Proc. European Conference on Speech Communication and Technology*, pages 1371–1374, Aalborg, Denmark, 2001.

[82] Marilyn A. Walker. An Application of Reinforcement Learning to Dialogue Strategy Selection in a Spoken Dialogue System for Email. *Journal of Artificial Intelligence Research*, 12:387–416, 2000.

[83] Marilyn A. Walker, Irene Langkilde-Geary, Helen Wright Hastie, Jerry Wright, and Allen Gorin. Automatically Training a Problematic Dialogue Predictor for a

Spoken Dialogue System. *Journal of Artificial Intelligence Research*, 16:293–319, 2002.

[84] Marilyn A. Walker, Alex Rudnicky, Rashmi Prasad, John Aberdeen, Elizabeth O. Bratt, John Garofolo, Helen Hastie, Audrey Le, Bryan Pellom, Alex Potamianos, Rebecca Passonneau, Salim Roukos, Gregory Sanders, Stephanie Seneff, and David Stallard. DARPA Communicator: Cross-System Results for the 2001 Evaluation. In *Proc. ICSLP*, pages 269–272, Denver, Colorado, September 2002.

[85] Taro Watanabe, Masahiro Araki, and Shuji Doshita. Evaluating Dialogue Strategies Under Communication Errors Using Computer-to-Computer Simulation. *IEICE Transactions on Information and Systems*, E81-D(9):1025–1032, September 1998.

[86] Frank Wessel, Ralf Schlüter, Klaus Macherey, and Hermann Ney. Confidence Measures for Large Vocabulary Continuous Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, 9(3):288–298, 2001.

[87] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[88] Wei Xu and Alex Rudnicky. Task-based Dialog Management Using an Agenda. In *Proc. ANLP/NAACL Workshop on Conversational Systems*, pages 42–47, Seattle, Washington, May 2000.

[89] Jon Yi. *Corpus-Based Unit Selection for Natural-Sounding Speech Synthesis.* PhD thesis, Massachusetts Institute of Technology, 2003.

[90] Jon Yi and James Glass. Information-Theoretic Criteria for Unit Selection Synthesis. In *Proc. ICSLP*, pages 2617–2620, Denver, 2002.

[91] Victor Zue, Stephanie Seneff, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and I. Lee Hetherington. JUPITER: A Telephone-based Conversational Interface for Weather Information. *IEEE Transactions on Speech and Audio Processing*, 8(1):85–96, January 2000.