Developing City Name Acquisition Strategies in Spoken Dialogue Systems Via User Simulation

Ed Filisko and Stephanie Seneff *
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge, Massachusetts 02139, USA
{filisko,seneff}@csail.mit.edu

Abstract

This paper describes our recent work on mechanisms for error recovery in spoken dialogue systems. We focus on the acquisition of city names and dates in the flight reservation domain. We are specifically interested in addressing the issue of acquiring out-of-vocabulary city names through a speak-and-spell mode subdialogue. In order to explore various dialogue strategies, we developed a user simulation system, which includes a configurable simulated user and a novel method of utterance generation. The latter utilizes a concatenative speech synthesizer, along with an existing corpus of dialogues, to produce a large variety of simulated inputs. The results from various simulated user configurations are presented, along with a discussion of how the simulated user facilitates the debugging of dialogue strategies and the discovery of situations unanticipated by the system developer.

Keywords: spoken dialogue systems, user simulation, error recovery

1 Introduction

Spoken dialogue systems are emerging as an intuitive interface for providing conversational access to online information sources (Gorin et al., 1997; Pieraccini et al., 1997; Dahlbäck et al., 1999; Gustafson et al., 1999; Zue et al., 2000; Eckert et al., 1997; Walker et al., 2001; Denecke, 2002; Polifroni and Chung, 2002; Seneff, 2002; Glass and Seneff, 2003; Quast et al., 2003). While the effectiveness of such systems 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 weather information for several hundred or even several thousand cities, but there is no easy way for the user to be informed of exactly which cities it knows. When the user asks for the weather 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 will only lead to failure and frustration.

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

Figure 1: 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.

Figure 1 shows an example dialogue in the Mercury flight reservation domain (Seneff, 2002), where the user was attempting to travel to a city the system did not

* This research was supported by an industrial consortium supporting the MIT Oxygen Alliance.

know. The system repeatedly misrecognized the unknown words, *Roseau* and *Dominica*, in a variety of ways, leading to ever increasing user frustration. Clearly, a more intelligent model is needed to detect and recover from such situations.

An improvement in system behavior can be accomplished through confidence scoring (Jurafsky and Martin, 2000; Wessel et al., 2001; Hazen et al., 2002) and/or an explicit unknown word model (Bazzi and Glass, 2002) to at least provide the possibility of predicting that the city might not be known. In such a case, the system can 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, the system could likely identify the city. If necessary, it could engage the user in a subdialogue to resolve any ambiguity due to either slight spelling variants or multiple cities with the same name in different states or countries.

Once the system knows the name of the city, it should be able to provide useful information to the user, for instance, by finding latitude/longitude data for the city on the Web or from a geography database, and then providing the weather for the closest known city. Furthermore, by invoking a letter-to-sound system (Chung, Wang, et al., 2004) that incorporates information gleaned from the spoken pronunciation of the city originally provided by the user, the system can add the new city to the recognizer's working vocabulary (Chung, Seneff, et al., 2004), so that future references to it should be understood.

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

One problematic aspect of developing and evaluating different 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 appropriate scenarios. 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 for both development and evaluation.

An attractive solution is to employ user simulation, both to help develop the system and to provide highly controlled evaluation experiments (Scheffler and Young, 2000, 2001; Eckert et al., 1997; Chung, 2004). Increasingly, researchers in spoken dialogue systems have been exploiting user simulation to aid in system development. For instance, in developing a restaurant domain (Chung, 2004), thousands of dialogues were generated automatically both to evaluate system performance and to produce training data for the recognizer language model. The simulated user consults the system's response frame to determine an appropriate query for each subsequent turn. In a recent experiment in the flight domain, we were able to locate points of communication breakdown in prior user dialogues, and continue those conversations with a different error recovery approach, using a synthesizer to generate waveforms for a solicited speak-and-spell subdialogue (Filisko and Seneff, 2004).

We report here on a novel approach to user simulation, which exploits an extensive database of preexisting user queries, but recombines the utterances using essentially an example-based template approach. The system deals with gaps in the corpus by splicing in synthetic waveforms for missing words, such as newly introduced rare cities. This allows us to retain the richness of variety in the ways users can pose questions, while introducing in a controlled way rare cities that would require an error recovery subdialogue to be resolved. We can then develop our error recovery strategies in the context of these simulated user dialogues. We can experiment with different settings for thresholds and measure the effectiveness in concrete terms such as number of turns and/or time to completion and overall success rate in understanding the artificially introduced rare cities.

We have also been developing a configurable generic dialogue manager, which aims to enable the system developer to assess the utility of modifying the system to behave in a certain way. For example, the system can be programmed to request a speak-and-spell utterance for an unknown city, and to implicitly confirm a city with a confidence score above a given threshold. The configurability of both the simulated user and the dialogue manager facilitates exploration and assessment of a wide variety of system strategies and user behaviors.

Our experiments have been conducted within the Mercury flight reservation domain (Seneff, 2002). We have access to over 30,000 utterances spoken by users in interactive dialogues with Mercury, which we are recycling with creative splicing to produce realistic simulated speech. While the original Mercury system knew only about 550 cities, we are now expanding its capability, with the goal of eventually supporting all the cities for

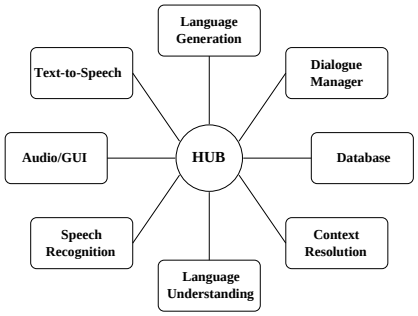

Figure 2: A typical GALAXY configuration for a spoken dialogue system.

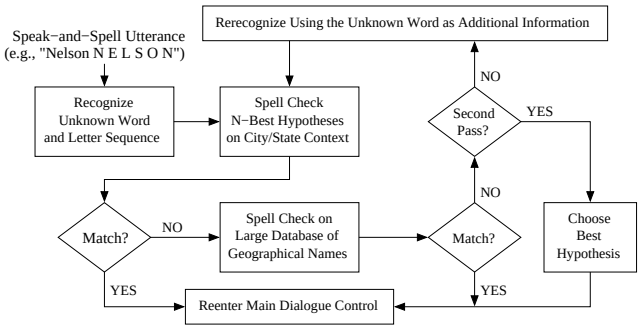

Figure 3: Flow chart detailing the two-pass speak-and-spell strategy for recovering from a problematic source or destination.

which we have airport codes. Our experiments focus on the subtask of resolving the source and destination, as well as the date of travel, three essential parameters for accessing our Sabre database.

2 Error Recovery Strategy

In previous work (Filisko and Seneff, 2004), we presented an analysis of error recovery in the Mercury domain, which utilized the technique of requesting a spelling for a problematic word via the telephone keypad. The algorithm for invoking keypad entry was conservative, so the recovery prompt was a relatively rare occurrence. Surprisingly, users only complied with the request a little over 50% of the time, and only 69% of those responses were valid spellings of a city or airport name. These results led us to believe that allowing the user to speak the spelling of the word may be a more intuitive recovery method, requiring less cognitive load.

Our newly designed error recovery mode is intended to greatly improve the likelihood of understanding rare city names. We require both the capability to detect a plausible recognition error and to initiate an error recovery mechanism that is likely to be successful. We decided to exploit a speak-and-spell capability that had been previously developed to handle name enrollment in an offline delegation task (Seneff et al., 2000). Its usage here was adapted to take advantage of the known offline lexicon, matching spelling hypotheses against the lexicon, and allowing for a modest amount of error correction.

All of our experiments are configured within the Galaxy Communicator (Seneff et al., 1998) framework. A suite of specialized servers communicate with one another via a central hub, as schematized in Figure 2, and control is specified in a *hub program*, implemented in a simple scripting language. Galaxy enables the configuration of complex interactions among the servers with relative ease, a feature that is essential for implementing our speak-and-spell error recovery strategy.

A block diagram of the speak-and-spell strategy is illustrated in Figure 3. The recognizer has been augmented to support an unknown word model as described in Bazzi and Glass (2002), and the language model has been trained to predict its likely occurrence as a substitution for a city name. In addition, we have incorporated a word-level confidence score, based on the algorithm described in Hazen et al. (2002). Whenever a triggering condition exists (e.g., the best hypothesis from the recognizer is a low-confidence or unknown city), the system prompts for a speak-and-spell entry: “Please speak the city name followed by its spelling, like this: Bel Air, B E L A I R.” The speak-and-spell utterance is redirected to a specialized recognizer, and the resulting N-best hypotheses are delivered to the dialogue manager (DM) for processing. The DM first proceeds to find a match with any hypotheses available from the N-best list associated with the previous user query. If a match is found, it presumes that the hypothesis is correct, and returns to the main dialogue control, without further confirmation from the user. If no match is found, the DM consults the lexicon of 20,000 city names and, finding a match, returns to the main dialogue control, confirming the city with the user. Not finding a match, the DM obtains a small N-best list of potentially matching cities and dynamically augments the speak-and-spell recognizer with a vocabulary consisting of these words, so that the spoken city name can now be recognized. The speak-and-spell waveform is then reprocessed. If no match is found during the second pass, the best spelling hypothesis is given to the dialogue manager, which resumes the main dialogue control.

2.1 Proof-of-Concept

The speak-and-spell strategy was first tested in a simple simulation scenario (Filisko and Seneff, 2004) in Mercury. A set of dialogue situations was selected, in which the system had requested keypad entry of either the source or destination city. Instead of providing a keypad spelling when prompted with one of these requests, the simulated user generated a speak-and-spell waveform utilizing the DECtalk (Hallahan, 1995) speech synthesizer. The waveform was then processed by the dialogue man-

ager using the strategy shown in Figure 3. Nearly 90% of the city names were correctly identified, indicating that this strategy could potentially be useful in real situations of error recovery.

It should be noted that this simulation assumed a completely compliant user who never hesitates or makes 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 (Zue et al., 2000) for four days. Although there were several instances of successful recovery, many types of noncompliance were revealed. One common action was for the user to completely ignore the speak-and-spell request, and to simply repeat the previous utterance.

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

3 User Simulation Studies

Although we could conceivably simply implement the complete error recovery strategy outlined above and incorporate it into our online Jupiter system, we were concerned about the possibility of alienating our pool of devoted users due to potentially annoying system behavior in unanticipated circumstances. We felt that an extensive stress-testing stage would reveal many of the possible dialogue situations that could arise due to recognition error or noncompliant user behavior. This could arguably be achieved by soliciting paid subjects to participate in various experiments involving carefully defined scenarios. However, this process is time-consuming and costly, and hence did not seem like a particularly attractive solution.

We decided instead to explore the option of an approach that involves *simulating* user behavior. We chose to develop this idea within the Mercury flight domain rather than the Jupiter weather domain, because Mercury is inherently more complex and will therefore support more interesting and challenging dialogue scenarios. The simulated user can replace the real user in an otherwise identical Galaxy configuration. The telephony server is replaced with the user simulator server, which acts based on the system response and calls upon the language generation and speech synthesis servers to convert its intentions into a speech waveform.

The simulated user is configurable by the developer, so that various behaviors may be adjusted and evaluated. For example, the user may initially speak one, two, or all required attributes for the domain, or the user may be set

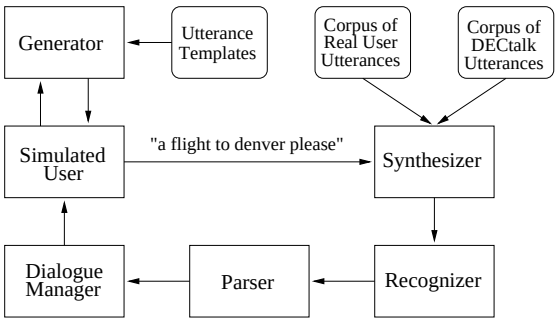

Figure 4: Flow within the dialogue system during a typical dialogue turn with the simulated user.

up to be noncompliant to a speak-and-spell request; that is, it will not respond with a speak-and-spell utterance when prompted to do so. Additionally, we employ a configurable dialogue manager, 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 speak-and-spell recovery attempts delivered before giving up on a given attribute.

In the following section, we describe our strategy for generating a rich set of realistic simulated user utterances in the flight domain, through creative recycling of an existing in-domain speech corpus. We then discuss a set of experiments that involve a simulated user attempting to convey its intentions to a system.

3.1 Utterance Generation from Existing Corpora

In order to make our simulated user's utterances as realistic as possible, we utilized a method of creatively combining segments of real users' speech from an existing corpus of in-domain utterances, along with speech segments synthesized by DECtalk. This method is beneficial in two ways:

1. it models the variety of ways that real users with similar goals pose questions and statements to convey their intentions, and
2. it facilitates the controlled exploration of specific classes of words via splicing.

A diagram of this creative utterance generation process is shown in Figure 4. 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 our experiments, the user's initial utterance contained only *source* and *destination*.

The simulated user sends this constrained set of beliefs via an eform (electronic form) frame to a generator, which consults a database of utterance templates, selects

“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 5: Examples of date patterns extracted from a corpus of Mercury utterances.

a template according to a set of developer- specified rules, and splices in the values provided in the eform. 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.

3.1.1 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 are 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 5.

An example database entry is shown in Figure 6. 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 eform 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 eform:

{*c eform :source “detroit” :destination “denver”* }

The system consults the corpus of eforms, like the one in Figure 6, 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 eforms. A header file specifies match constraints. For example, we can allow any value for “:clause” (e.g., “exist”, “clarifier”, “wh-query”) while rejecting any eforms that contain meaningful additional attributes such as “:departure_time” or “:airline”.

If the third template were chosen from the set in Figure 6, the following string would be produced:

{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 6: An entry from the database of utterance templates. Each unique eform is associated with a set of corresponding sentences. This eform contains a total of 58 unique sentence templates.

`i am flying from detroit to denver`

which would then be passed to the synthesizer.

3.1.2 Concatenative Speech Synthesis

The simulation strategy presented here employs the Envoice concatenative speech synthesizer (Yi and Glass, 2002). Envoice synthesizes a given utterance by generalizing from existing speech corpora, concatenating 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 is provided with a pronunciation of the word.

While the corpus is not likely to contain a requested utterance in its entirety, the synthetic waveforms are nonetheless composed of real speech segments, and therefore model the variety of syntactic forms and phonetic realizations typically produced by real users. Many of the utterances will contain segments from multiple speakers, but this should not disturb recognition since our recognition algorithm has no speaker-specific 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 size 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 1. The set of original waveforms showed a word error rate (WER) of 12%, while the DECtalk set performed significantly better with a WER

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 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).

of only 8.5%. The Envoice sets both showed about a 13% 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 developing 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.

3.2 Dialogues with a Simulated User

The replacement of a real user with a simulated one 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 to measure the efficiency or success of the given strategy.

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 three attributes in the flight domain: source city, destination city, 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.

3.2.1 Overall Dialogue Strategy

Figure 7 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 speak-and-spell requests and explicit confirmations, until the attribute's value has been confidently acquired, or its acquisition has been aborted. The DM then continues to look for more attributes in the user's input.

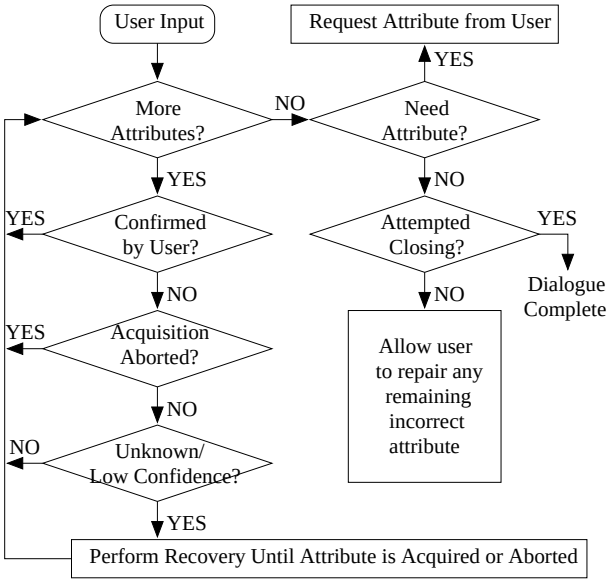

Figure 7: Flow chart of the dialogue manager's strategy for acquiring values for a set of attributes.

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. 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.

3.2.2 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 a speak-and-spell (SAS) subdialogue, as previously described in Figure 3. In all configurations, the system runs a main recognizer alongside the specialized SAS recognizer whenever it prompts for a SAS utterance, and both hypotheses are given to the DM for consideration. The system attempts only one error correction cycle per attribute, although it allows 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.

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 are 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 2 describes the configuration for each of the five simulation experiments. *Source* and *Destination* indicate whether a city name spoken by the user was known or unknown 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 a speak-and-spell utterance upon request. Noncompliant users would respond, for example, by rephrasing their previous utterance.

The appendix provides several examples of conversations between the simulated user and the Mercury system. Figure A-1 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 A-2, 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 A-3 shows how the simulated user can reword 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 A-4 shows a dialogue with a noncompliant simulated user. The system prompts for a 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 a 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 A-5 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. In future work, we intend to enable a much more robust acquisition of dates.

3.2.3 Simulation Experiments and Results

Table 3 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 varies among different runs of a given configuration, mainly due to the quality of the concatenated speech, we ran 10 simulations of 50 dialogues each, for configuration A. A mean of 33.9 ± 2.5 on the number of successful dialogues was obtained.

The best dialogue success rate is realized in configuration A, in which the simulated user provided known city names, as well as their associated state names. The user is also compliant to SAS requests. While only 36 of 50 dialogues were successful, more than 88% of the attribute values were correctly obtained. As expected, configuration A also yielded the lowest average of 6.6 turns per successful dialogue. This result reflects the efficiency of packing more information into a single utterance.

Configuration B is similar to A, except that only the city name was provided in the user's initial utterance. Consequently, the state name was requested more frequently, and this increased the average number of turns per successful dialogue. This also appears to be a less effective user strategy, since the number of successful dialogues dropped by 5 to 31.

Configuration C models a noncompliant user, who responded to a SAS request by rephrasing the previously spoken utterance. The simultaneous use of two recognizers enables the system to handle this situation appropriately, as confirmed by the results. Interestingly, the over-

all success rate was higher than in B, suggesting that this strategy is actually a productive one. One would hope that real users might be inclined to ignore a SAS request precisely in those situations where the system is likely to know the intended city.

In configuration D, the system never adopted a strategy of soliciting a 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 cities.

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 (oov), the recognizer would be expected to propose the oov city "word" 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 B 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 observing 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 D, with the noncompliant user, would have been disastrous.

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

Table 2: 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	36	31	32	33	30
AvgSuccTurns	6.6	7.5	7.5	7.7	9.0
AvgFailTurns	8.4	9.0	10.2	9.1	10.3
AvgAccuracy (%)	88.4	87.2	88.8	91.4	85.9

Table 3: Results for the simulation configurations in Table 2. Each configuration consisted of 50 simulated dialogues. *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.

4 Related Work

4.1 User Simulation

While user simulation has not been a standard component of spoken dialogue system development, a number of researchers have reported on experiments conducted and evaluations made on the basis of user simulation. Eckert et al. (1997) argued the case of user simulation to provide a potentially infinite corpus at little cost. In their experiments, conducted in the ATIS flight domain, the user simulation was represented only at the intention level, thus bypassing both the recognizer and the language understanding. Simulated user responses were conditioned only on the previous system response, which could lead to a rambling dialogue without a goal.

Scheffler and Young (2000, 2001) explored the use of user simulations in two domains, an online banking system and a movie guide. They also represented the simulated user's queries only at the intention level, but initiated two advances over the Eckert et al. approach. 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.

López-Cózar et al. (2002) made use of a simulated user model to compare the performance of two different recognizers and two different strategies for handling confirmation (implicit vs. explicit). The domain involved ordering at a fast-food restaurant, and the simulated user was tasked with solving a specific scenario. The simulated user's intentions were mapped to speech by randomly selecting a *read utterance* associated with the same defined subgoals.

The research reported by Chung (2004) most closely matches our own work. Chung utilized a language generation component to convert the simulated user's intentions into natural English. The simulated user was assigned scenarios to solve within a restaurant guide domain. The text strings were converted to *synthetic speech* using a speech synthesizer, and the synthetic speech was then processed through the standard dialogue system. A distinction between her work and ours is that we use a large corpus of in-domain, spontaneously produced utterances to provide a much greater variety of linguistic patterns, as well as a more realistic synthetic speech waveform, for each utterance.

4.2 Error Detection and Recovery

While many researchers have addressed the topic of error recovery subdialogues in one way or another, we highlight here two papers that we feel are most relevant to our own work, distinguished by their use of an alternate input mode to correct an error.

Suhm and Waibel (1997) have developed a flexible error recovery mechanism to handle corrections in a speech transcription task. The user is allowed to correct errors in three ways: by repeating the misrecognized word, by spelling it, or by handwriting. The user has full control over identifying where the errors have occurred through an intuitive mouse-based interface to the displayed text. A novelty of their system is its use of the local context to improve language modeling for the error recovery task.

Bauer and Junkawitsch (1999) describe research 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.

5 Summary and Future Work

This paper describes our work in the area of error recovery subdialogues, in order to more confidently acquire a user's intentions. An analysis of the error recovery mechanism in the Mercury flight reservation domain revealed that both system and users were not very successful in handling problematic city names using spelling via a telephone keypad. A preliminary simulation experiment demonstrated that the use of a speak-and-spell (SAS) mechanism had the potential to be successful with real users: nearly 90% of the city names were correctly identified. Incorporating the SAS mechanism into the live Jupiter weather system revealed that real users are not always compliant with SAS requests.

These observations inspired us to develop a simulated user, which would replace a real user in the dialogue system cycle. This enables batch processing of many dialogues in order to observe the performance of specific dialogue strategies in a controlled environment. To more realistically model the variability of real user utterances in the simulations, a method of creative utterance generation was developed, in which existing in-domain corpora are used to create a database of utterance templates. Sets of synthesized words may be spliced into the templates, facilitating the performance evaluation of very specific semantic classes, such as U.S. city names or dates.

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 is 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.

In future work, we hope to refine this simulation method to provide a useful tool for system development in new domains. We will extend the repertoire of simulated user behaviors and, in parallel, increase the sophistication of the dialogue model to handle them. The use of simulation is an inexpensive yet powerful method to reveal mistakes in the logic of the dialogue strategies.

There are at least two ways in which we hope to modify the system in the future, and we will assess these changes via user simulation studies. One is to explore expanding the original vocabulary to license a much larger number of cities, perhaps all the cities for which we have airport codes. We expect that SAS mode will turn out to be needed for the rare cities, because it will be difficult for the system to get them right. But it may be that city-state combinations are sufficient by themselves. The other direction to explore is to allow *users* to initiate a SAS mode subdialogue. This would mean running the SAS recognizer in parallel all the time, but paying attention to its result only when it appeared legitimate.

Once we have verified that our error recovery subdialogue is effective in simulation runs for both compliant and noncompliant users, we would like to reintroduce it into the Jupiter weather domain system. We also plan to experiment with its utility in other systems such as directory information, where a SAS mechanism could be extremely useful in identifying peoples' names.

References

Bauer, J.G. and J. Junkawitsch (1999). Accurate Recognition of City Names with Spelling as a Fallback Strategy, In *Proc. EUROSPEECH*, Budapest, Hungary, 263–266.

Bazzi, I. and J. Glass (2002). Multi-Class Approach for Modelling Out-of-Vocabulary Words, In *Proc. ICSLP*, Denver, Colorado, 1613–1616.

Chung, G., C. Wang, S. Seneff, E. Filisko, and M. Tang (2004). Combining Linguistic Knowledge and Acoustic Information in Automatic Pronunciation Lexicon Generation, In *Proc. INTERSPEECH*, Jeju Island, Korea.

Chung, G., S. Seneff, C. Wang, and L. Hetherington (2004). A Dynamic Vocabulary Spoken Dialogue Interface, In *Proc. INTERSPEECH*, Jeju Island, Korea.

Chung, G. (2004). Developing a Flexible Spoken Dialog System Using Simulation, In *Proc. ACL 2004*, Barcelona, Spain, 63–70.

Dahlbäck, N., A. Flycht-Eriksson, A. Jönsson, and P. Qvarfordt (1999). An Architecture for Multi-Modal Natural Dialogue Systems, In *Proc. ESCA Tutorial and Research Workshop on Interactive Dialogue in Multi-Modal Systems*.

Denecke, M. (2002). Rapid Prototyping for Spoken Dialogue Systems, In *Proc. COLING*, Taipei, Taiwan.

Eckert, W., E. Levin, and R. Pieraccini (1997). User Modeling for Spoken Dialogue System Evaluation, In *Proc. IEEE ASR Workshop*.

Filisko, E. and S. Seneff (2004). Error Detection and Recovery in Spoken Dialogue Systems, In *Proc. Workshop on Spoken Language Understanding for Conversational Systems*, Boston, Massachusetts, 31–38.

Glass, J. and S. Seneff (2003). Flexible and Personalizable Mixed-Initiative Dialogue Systems, In *Proc. HLT-NAACL Workshop on Research Directions in Dialogue Processing*, Edmonton, Canada.

Gorin, A., G. Riccardi, and J. Wright (1997). How May I Help You? *Speech Communication*, 23:113–127.

Gustafson, J., N. Lindberg, and M. Lundeberg (1999). The August Spoken Dialogue System, In *Proc. EUROSPEECH*, Budapest, Hungary.

Hallahan, W.I. (1995). DECtalk Software: Text-to-Speech Technology and Implementation, *Digital Technical Journal*, 7(4):5–19. Accessed 04/15/05, http://www.hpl.hp.com/hpjournal/dtj/vol7num4/toc.htm

Hazen, T.J., S. Seneff, and J. Polifroni (2002). Recognition Confidence Scoring and its Use in Speech Understanding Systems, *Computer Speech and Language*, 16:49–67.

Jurafsky, D.S. and J.H. Martin (2000). Speech and Language Processing. Prentice Hall, Inc., Englewood, New Jersey.

L´opez-C´ozar, R. et al. (2002). A New Method for Testing Dialogue Systems Based on Simulations, In *Proc. ICSLP*, Denver, Colorado.

Pieraccini, R., E. Levin, and W. Eckert (1997). AMICA: The AT&T Mixed Initiative Conversational Architecture, In *Proc. EUROSPEECH*, 1875–1878.

Polifroni, J. and G. Chung (2002). Promoting Portability in Dialogue Management, In *Proc. ICSLP*, Denver, Colorado, 2721–2724.

Quast, H., T. Scheideck, P. Geutner, and A. Korthauer (2003). RoBoDiMa: A Dialogue-Object Based Natural Language Speech Dialog System, In *Proc. ASRU Workshop*, Saint Thomas, Virgin Islands, 174–179.

Scheffler, K. and S. Young (2000). Probabilistic Simulation of Human-Machine Dialogues, In *Proc. ICASSP*, Istanbul, Turkey, 1217–1220.

Scheffler, K. and S. Young (2001). Corpus-Based Dialogue Simulation for Automatic Strategy Learning and Evaluation, In *Proc. Adaptation in Dialogue Systems Workshop at NAACL*, Pittsburgh, Pennsylvania.

Seneff, S., C. Chuu, and D.S. Cyphers (2000). ORION: From On-line Interaction to Off-line Delegation, In *Proc. ICSLP*, Beijing, China, 142–145.

Seneff, S., E. Hurley, R. Lau, P. Schmid, and V. Zue (1998). Galaxy-II: A Reference Architecture for Conversational System Development, In *Proc. ICSLP*, Sydney, Australia, 931–934.

Seneff, S. (2002). Response Planning and Generation in the Mercury Flight Reservation System, *Computer Speech and Language*, 16:283–312.

Suhm, B. and A. Waibel (1997). Exploiting Repair Context in Interactive Error Recovery, In *Proc. EUROSPEECH*, Rhodes, Greece.

Walker, M. et al. (2001). DARPA Communicator Dialog Travel Planning Systems: The June 2000 Data Collection, In *Proc. EUROSPEECH*, Aalborg, Denmark, 1371–1374.

Wessel, F., R. Schluter, K. Macherey, and H. Hey (2001). Confidence Measures for Large Vocabulary Continuous Speech Recognition, *IEEE Trans. on Speech and Audio Proc.*, 9(3):288–298.

Yi, J. and J. Glass (2002). Information-Theoretic Criteria for Unit Selection Synthesis, In *Proc. ICSLP*, Denver, Colorado, 2617–2620.

Zue, V., S. Seneff, J. Glass, J. Polifroni, C. Pao, T.J. Hazen, and L. Hetherington (2000). Jupiter: A Telephone-Based Conversational Interface for Weather Information, *IEEE Trans. on Speech and Audio Proc.*, 8(1):85–96.

Appendix

This appendix provides five example dialogues between the simulated user (U) and the Mercury system (S).

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 A-1: 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 A-2: 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 A-3: In this example, the system requests a date from the user, but misunderstands the user's response in U3. The user rewords 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 A-4: 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 rewords 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 A-5: 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 rewords 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.