Developing A Flexible Spoken Dialog System Using Simulation

Grace Chung[*]
Corporation for National Research Initiatives
1895 Preston White Drive, Suite 100
Reston, VA, 20191
gchung@cnri.reston.va.us

Abstract

In this paper, we describe a new methodology to develop mixed-initiative spoken dialog systems, which is based on the extensive use of simulations to accelerate the development process. With the help of simulations, a system providing information about a database of nearly 1000 restaurants in the Boston area has been developed. The simulator can produce thousands of unique dialogs which benefit not only dialog development but also provide data to train the speech recognizer and understanding components, in preparation for real user interactions. Also described is a strategy for creating cooperative responses to user queries, incorporating an intelligent language generation capability that produces content-dependent verbal descriptions of listed items.

1 Introduction

Spoken dialog systems have traditionally been difficult to instantiate because of the extensive efforts required for configuring each component from the natural language (NL) understanding to the domain-specific context tracking and dialog engines. This task is particularly involved when building systems that empower users with greater flexibility at the spoken input via mixed-initiative interaction (Zue et al., 2000; Ferguson and Allen, 1998), and systems that enable querying across large and frequently changing databases.

The goal of this work is to assemble natural spoken dialog interfaces that enable flexible interactions through mixed initiative dialog and cooperative responses. Such interfaces aim to help users navigate large information spaces such as on-line databases.

Conventional systems generally guide the users through a series of scripted prompts, either through a finite-state grammar or a dialog control table. In Table 1, the system prompts present a sequence of questions in an attempt to solicit more constraints from the user, with the goal of obtaining a small data subset from the database. The system responses are generated from a set of rules that only anticipate one of a handful of situations: (1) when the set of entries returned is too large, (2) the set of entries is adequately small to enumerate, and (3) no available entries have been returned.

User:	Are there any Thai restaurants?
System:	Can you provide a neighborhood or city?
User:	Boston.
System:	There are 21 entries found. Can you be more specific?
User:	A restaurant in Dorchester.
System:	There are no Thai restaurants in Dorchester. Is there anything else I can help you with?
User:	What about in the South End?
System:	There are 2 entries found. Thai Village and House of Siam.

Table 1: Example human-computer dialog.

A more flexible scenario would allow the user to browse the content by specifying one or more constraints in *any* order. The system should then return a succinct summary of the content upon user specification of each constraint. This would provide improved feedback to the user about the available choices so far, guards against stilted conversations with a fixed number of dialog turns for every interaction, and mitigates against repeated scenarios where user queries return no items. However, much effort is then required in configuring the numerous scenarios for users to make sequences of queries in various orders. User queries are likely to differ if the database contents shift over time, changing the frequency and availability of certain entries. Furthermore, there remains the well-known "chicken-and-egg" problem of obtaining real-user data. With no real examples of human-computer interactions, it is difficult for developers to instantiate and configure

[*] This research is sponsored in part by SPAWAR SSC-SD. The content of this paper does not necessarily reflect the position or policy of the Government, and no official endorsement should be inferred.

a robust system. Yet without a reasonably operational system, it is equally difficult to convince real users to generate dialogs, particularly those which achieve successful completion. Hence, the usual development process consists of multiple iterations of expensive data collections and incremental system improvements.

This paper presents an alternative paradigm for designing such a spoken dialog system. Our methodology employs *simulations* to reduce the time and effort required to build the system. Simulations facilitate prototyping and testing of an initial version of the system that automatically produces cooperative responses to user queries. We advocate the use of a suite of simulation techniques to create large numbers of synthetic user interactions with the system, including both typed and spoken inputs, where the speech is generated using a speech synthesizer.

The resulting dialogs can be used to (1) diagnose the system for any problematic interactions, (2) enable a developer to examine system responses for large numbers of possible user queries, and (3) create an initial corpus for training the language models and probabilistic NL grammar. Thus, the initial phase of development comprises simulating hundreds of dialogs and iterative refinements prior to real-user data collection.

In the next sections, we first describe our spoken dialog system architecture. This is followed by a description of a simulator, which operates in concert with a language generation system to output synthetic user queries. We elaborate on how the architecture can simulate coherent dialogs, and can be tuned to simulate a cooperative or uncooperative user. Then, methods for generating cooperative responses for a restaurant information domain are described. We detail how simulations have accelerated these developments.

2 System Architecture with Simulator

Figure 1 depicts a spoken dialog system architecture functioning with simulator components, which create synthetic user inputs. Simulations can be customized to generate in text or speech mode. In text mode, text utterances are treated as user inputs to the understanding components. The dialog manager creates reply frames that encode information for generating the system reply string. These are also used by the simulator for selecting a random user response in the next turn. In speech mode, synthetic waveforms are created and recognized by the speech recognizer, yielding an N-best list for the understanding components.

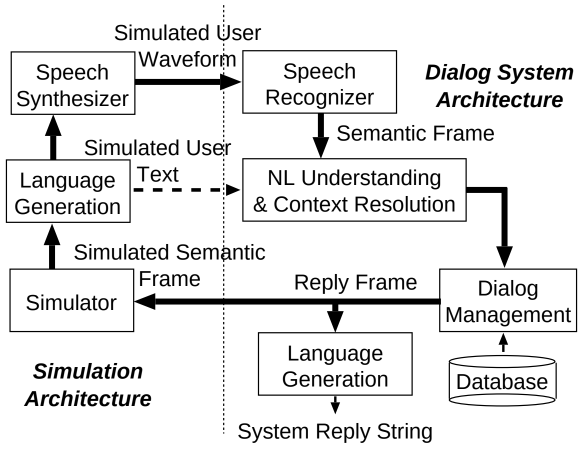

Figure 1: A spoken dialog system architecture integrated with user simulation components.

Examples and experiments in this paper are drawn from a Boston restaurant information system. Obtained from an on-line source, the content offers information for 863 restaurants, located in 106 cities in the Boston metropolitan area (e.g., Newton, Cambridge) and 45 neighborhoods (e.g., Back Bay, South End). Individual restaurant entries are associated with detailed information such as cuisines, phone numbers, opening hours, credit-card acceptance, price range, handicap accessibility, and menu offerings. Additionally, latitude and longitude information for each restaurant location have been obtained.

2.1 Instantiation of a System

The concept of driving the instantiation of a dialog system from the data source was described in (Polifroni et al., 2003). In the following, the steps envisioned for creating an initial prototype starting with on-line content are summarized below:

1. Combing the web for database content
2. Identifying the relevant set of keys associated with the domain, and mapping to the information parsed from the content originator
3. Creating an NL grammar covering possible domain queries
4. Configuring the discourse and dialog components for an initial set of interactions
5. Defining templates for system responses

The above steps are sufficient for enabling a working prototype to communicate with the proposed simulator in text mode. The next phase will involve iteratively running simulated dialogs and refinements on the spoken dialog system, followed by

```
{c summary
   :count 14
   :categories
   ( {c cuisine
         :ordered_counts ( 4 2 2 2 ...
         :ordered_values ( "american" "indian" ..
   {c price_range
         :ordered_counts ( 7 2 2 1)
         :ordered_values ( "cheap" "low" "medium" ..
```

Table 2: Example summary frame derived from the system reply frame.

examination of successive corpora of simulated dialogs. Later phases will then incorporate the speech recognition and text-to-speech components.

2.2 Simulation with User Modeling

The simulator, Figure 1, is composed of several modular components. The core simulator accepts reply frames from the dialog system, and produces a meaning representation of the next synthetic user response. A text generation component paraphrases the meaning representation into a text string. In text mode, this poses as a typed user input, whereas in speech mode, the text is passed to a synthesizer as part of a synthesize/recognize cycle. Configuring a simulation for any domain involves customizing a simple external text file to control the behavior of the domain-independent simulator module, and tailoring text generation rules to output a variety of example user input sentences from the meaning representation.

One simulated dialog would commence with an initial query such as "what restaurants do you provide?". The synthetic user makes successive queries that constrain the search to data subsets. It may (1) continue to browse more data subsets, or (2) when a small list of data entries is in focus, choose to query attributes pertaining to one or more individual items, or (3) terminate the conversation. The entire system is run continuously through hundreds of dialogs to produce log files of user and system sentences, and dialog information for subsequent analyses. The simulator also generates generic kinds of statements such as asking for help, repeat and clearing the dialog history.

2.2.1 Generation of Semantic Frames

The simulator takes input from the system-generated reply frame, and outputs a flat semantic frame, encapsulating the meaning representation of the next intended user query. The system reply frame contains the essential entities, used in the paraphrase for creating the system prompt. But also, a sub-frame, shown in Figure 2, retains pre-

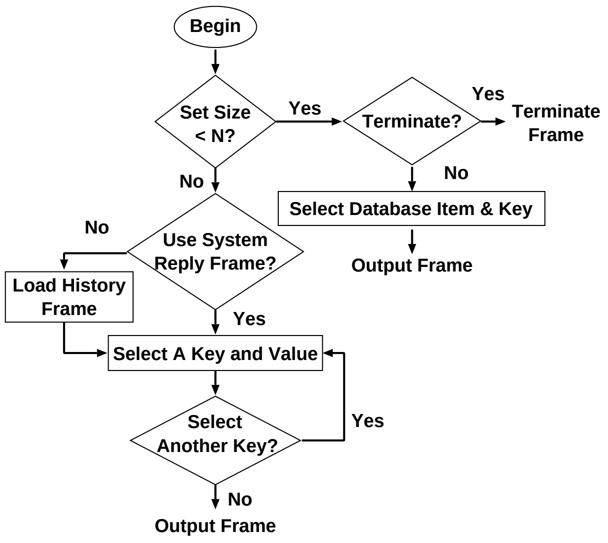

Figure 2: A schematic showing the decision making procedure for the simulator.

computed counts associated with the frequency of occurrence of values for every key pertaining to the data subset within the discourse focus. During the browsing stage, the simulator randomly selects a key (e.g, a cuisine) from the given frame, and then makes a random selection on the value, (e.g., "Chinese."). The simulator may choose one or more of these key-value pairs as constraints to narrow the search. For each key, more than one value from the list of possible values may be specified, (e.g., querying for "Chinese or Japanese restaurants."). When querying about individual restaurants, the simulator randomly selects one restaurant entry from a small list, and then seeks to obtain the value for one key characteristic for a restaurant entry. For example, this could be a phone number or an address.

Figure 2 illustrates the decision making performed by the simulator at each turn. At each decision point, the system "throws the dice" to determine how to proceed, for example, whether to select an additional key for constraint within the same turn, and whether to persist in querying about the available attributes of the small list of restaurants or to start over.

The behavior of the simulator at each decision point can be tuned from an external text file, which allows the following to be specified:

- Probability of combining several constraints into a single query
- Probability of querying a different value for a previous key versus selecting from among other keys presented by the reply frame

- Probability of continued querying of the attributes of restaurants from a list of one or more restaurants
- Probability of the user changing his goals, hence querying with alternative constraints

A simple user model is maintained by the simulator to track the key-value pairs that have already been queried in the current dialog. This tracks the dialog history so as to enable the synthetic user to further query about a previously mentioned item. It also prevents the dialog from cycling indefinitely through the same combinations of constraints, helping to make the dialog more coherent.

The external configuration file can effectively tune the level of cooperative behavior for the synthetic user. If the synthetic user selects a single key-value pair from the reply frame at each turn, a non-empty and successively smaller data subset is guaranteed to result at each turn. Moreover, selections can be configured to bias towards frequencies of instance values. The basis for this stems from the hypothesis that locations populated with more restaurants are likely to be queried. That is, the statistics of the database instances can directly reflect on the distribution of user queries. For instance, users are more likely to query about, "Chinese restaurants in Chinatown." Hence, the output dialogs may be more suitable for training language models. Alternatively, the synthetic user may be configured to select random combinations of various keys and values from the current or stored summary frame at a turn. Under these circumstances, the subsequent database retrieval may yield no data for those particular combinations of constraints.

2.2.2 Generation of Simulated Utterances

Each semantic frame is input to Genesis, a text generation module (Seneff, 2002), to output a synthetic user utterance. Genesis executes surface-form generation via recursive generation rules and an associated lexicon. A recent addition to Genesis is the ability to randomly generate one of several variant sentences for the same semantic frame. A developer can specify several rules for each linguistic entity allowing the generator to randomly select one. Due to the hierarchical nature of these templates, numerous output sentences can be produced from a single semantic frame, with only a few variants specified for each rule. Table 3 depicts example semantic frames and corresponding sample sentences from the simulator.

In total, the full corpus of simulated sentences are generated from approximately 55 hand-written rules in the restaurants domain. These rules distinguish themselves from previous text generation tasks by the incorporation of spontaneous speech phenomena such as filled pauses and fragments. In the initial phase, this small rules set is not systematically mined from any existing corpora, but is handcrafted by the developer. However, it may be possible in future to incorporate both statistics and observations learned from real data to augment the generation rules.

2.2.3 Synthetic User Waveforms

A concatenative speech synthesizer (Yi et al., 2000) is used to synthesize the simulated user utterances for this domain. The parameters and concatenative units employed in this synthesizer were tailored for a previous domain, and therefore, the naturalness and intelligibility of the output waveforms are expected to be poor. However, the occurrence of some recognition errors may help in assessing their impact on the system.

3 Cooperative Response Strategies

We have aimed to design a more cooperative spoken dialog system in two respects. First, the information is delivered so that at each turn a dynamic summary of the database items in focus is presented. Secondly, the dialog manager is augmented with a domain-independent algorithm to handle over-constrained queries. The system gives alternative suggestions that are integrated with the dynamic summaries.

3.1 Flexible System Responses

Response planning is performed both in the dialog management and the language generator, Genesis. To enable flexible responses, and avoid rigid system prompts, the dialog manager accesses the database at every turn with the current set of user-specified constraints in focus. With this data subset returned, a data refinement server (Polifroni et al., 2003) then computes frequency characteristics of relevant keys for the subset. This is incorporated into the system reply frame as shown in Table 2.

Following this, Genesis provides a summary of the characteristics of the data set, utilizing context information provided by the dialog manager and the frequency statistics. Genesis provides control on how to summarize the data linguistically via explicit rules files. The developer can specify variables m, n, and $\%age$ which control how lists of items are summarized, separately for different classes of data. If the number of items is under n, all options are enumerated. If the top m frequency counts cover more than $\%age$ of the data, then these categories will be suggested, (e.g. "Some choices are Italian

Frame	Example Sentences
{c seek :neighborhood “Back Bay” :price_range “low” }	I’m interested in some low end restaurants in Back Bay please. Inexpensive restaurants in Back Bay. Okay a cheap restaurant in Back Bay. <uh> Are there any cheap restaurants in Back Bay?
{c request_property :property ”hours” :name ”Emma’s” }	Can you please tell me the hours for Emma’s? When is Emma’s open? Well what are the hours for Emma’s? Okay then what are the opening hours of Emma’s?

Table 3: Sample semantic frames from the simulator, along with examples of generated sentence outputs. For each example frame above, hundreds of simulated variant sentences can be obtained.

and Chinese.”). Alternatively, summaries can indicate values that are missing or common across the set, (e.g. “All of them are cheap.”).

By accessing the database and then examining the data subset at each turn, the system informs the user with a concise description of the choices available at that point in the dialog. This is a more flexible alternative than following a script of prompts where in the end the user may arrive at an empty set. Moreover, we argue that performing the summary in real time yields greater robustness against changes in the database contents.

3.2 Dialog Management

The domain-independent dialog manager is configurable via an external dialog control table. A set of generic functions are triggered by logical conditions specified in formal rules, where typically several rules fire in each turn. The dialog manager has been extended to handle scenarios in which the user constraints yield an empty set. The aim is to avoid simply stating that no data items were found, without providing some guidance on how the user could re-formulate his query. Domain-independent routines relax the constraints using a set of pre-defined and configurable criteria. Alternate methods for relaxing constraints are:

- If a geographical key has been specified, relax the value according to a geography ontology. For instance, if a particular street name has been specified, the relaxation generates a subsuming neighborhood constraint in place of the street name.
- If a geographical key has been specified, remove the geographical constraint and search for the nearest item that satisfies the remaining constraints. The algorithm computes the nearest item according to the central latitude/longitude coordinates of the neighborhood or city.
- Relax the key-value with alternative values that have been set to defaults in an external file. For instance, if a Vietnamese restaurant is not available at all, the system relaxes the query to alternative Asian cuisines.
- Choose the one constraint to remove that produces the smallest data subset to speak about. If no one constraint is able to produce a non-empty set, successively remove more constraints. The rationale for finding a constraint combination that produces a small data set, is to avoid suggesting very general alternatives: for instance, suggesting and summarizing the “337 cheap restaurants” when “cheap fondue restaurants” were requested.

The routine will attempt to apply each of these relaxation techniques in turn until a non-zero data set can be attained.

4 Experiments

4.1 Simulations in Text Mode

The first stage of development involved iteratively running the system in text mode and inspecting log files of the generated interactions for problems. This development cycle was particularly useful for extending the coverage of the NL parser and ensuring the proper operation of the end-to-end system.

Simulations have helped diagnose initial problems overlooked in the rule-based mechanisms for context tracking; this has served to ensure correct inheritance of attributes given the many permutations of sequences of input sentences that are possible within a single conversation. This is valuable because in such a mixed-initiative system, the user is free to change topics and specify new parameters at any time. For instance, a user may or may not follow up with suggestions for restaurants offered by the system. In fact, the user could continue to modify any of the constraints previously specified in the conversation or query any attributes for an alternate newly spoken restaurant. There are vast numbers of dialog contexts that can result, and simulations have assisted greatly in detecting problems.

Furthermore, by generating many variations of possible user constraints, simulations have also helped identify initial problems in the summarization rules for system response generation. The text generation component is handcrafted and benefits largely from examples of real queries to ensure their proper operation. These kinds of problems would otherwise normally be encountered only after many user interactions have occurred.

Table 4 shows a typical simulated dialog. In the interaction shown, the simulator provides one or more constraints at each turn. It also selects alternative values according to the previous chosen key. After the dialog has arrived at a small data set, the simulator randomly asks questions about individual items.

During one simulation run, we completed 2000 dialogs in text mode. There were a total of 8147 input utterances, resulting in an average of 4.07 input utterances per dialog. Of the input utterances, 5446 were unique. These were generated from 3349 unique semantic frames. There were 4320 unique system replies.

4.2 Simulating Over-Constrained Queries

By configuring the simulator to query with multiple constraints in single turns, we were able to synthesize over-constrained queries, in which, initially, empty data sets were retrieved. In earlier development cycles, simulated dialogs were crucial in finding combinations that yielded problematic responses due to over-constrained queries. In one run after refinements, we tested 300 dialogs whereby two or more constraints were combined in each query by the simulator. As a result, 113 queries necessitated the system to invoke the relaxation algorithm in order to present a suggested alternative data set. Under all these cases, the system gave suggestions without fail. Some examples of the replies are shown in Table 5.

4.3 Simulation with Speech Recognition

The next step is to implement the synthesize-and-recognize cycle whereby the text-to-speech output is directly piped to the recognition engine. A speech recognizer (Glass, 2003) is built using the utterances produced by the text mode as training data for the language model. This recognizer uses class n-gram models. With the training set of 8147 utterances, the perplexity of the bigram and trigram models are respectively 6.6 and 5.1. The speech recognizer, including the restaurant names contains 2325 vocabulary items. The understanding component would select one sentence of an N-best list output from the speech recognizer.

Presently, when a recognition error causes a hard parse failure in the understanding, the simulator is configured to output the same semantic frame as contained in the previous utterance. This will cause the text generator to output a different variant of the same query. If the parse failures occur multiple times in sequence, the simulated user aborts and terminates the dialog.

SIM:	What restaurants do you know?
SYS:	There are 863 restaurants. Some of the options are American, Pizza, and Italian. Most of them are located in Boston and Cambridge.
SIM:	Any restaurants in Back Bay?
SYS:	There are 57 restaurants in Back Bay. Many of them are American, and Nouvelle Cuisine.
SIM:	So how about food in The South End?
SYS:	There are 42 restaurants in The South End. They are mostly American, and Nouvelle Cuisine. Many of them are located on Columbus Avenue, and Tremont Street.
SIM:	I would like a moderately priced restaurant.
SYS:	There are 9 medium priced restaurants in The South End. Some of the options are Nouvelle Cuisine, and American. Many of them are on Tremont Street and Appleton Street.
SIM:	<oh> What about any cheap American restaurants?
SYS:	There are 7 cheap American restaurants in The South End. They are located on Columbus Avenue, Tremont Street, Dartmouth Street, and Washington Street.
SIM:	Columbus Avenue.
SYS:	I have found 3 cheap American restaurants in The South End on Columbus Avenue. Their names are Charlie's Sandwich Shoppe, Bob The Chefs, and Grill Fish.
SIM:	What is the phone number of first one?
SYS:	The phone number of Charlie's Sandwich Shoppe is 617 536 7669.
SIM:	What is the price range for Grill Fish?
SYS:	The price range of Grill Fish is cheap.

Table 4: Typical example of a simulated dialog. SIM are the simulated inputs and SYS are the system responses.

4.4 Results from Spoken Dialog System

Our initial run of 36 dialogs yielded 213 sentences. To simplify dialogs, the simulator is configured to specify just one constraint at each turn. After the

1. Cheap Restaurants on Rowes Wharf: There are no cheap restaurants on Rowes Wharf. However, there are in total 5 cheap restaurants in the Financial District. They are on Broad Street, Post Office Square, Federal Street, and Bromfield Street.
2. Cheap Thai Restaurants in Dorchester: There are no cheap Thai restaurants in Dorchester. However, there are in total 14 cheap Thai restaurants. They are predominantly in Brighton, and The South End. The nearest one is Thai Village in the South End. It is less than 3 miles away from Dorchester.
3. Expensive Asian Restaurants on Hale Street: There are no high end Asian restaurants on Hale Street. Alternatively Yanks serves high end American food. It is located at 717 Hale Street in Beverly Farms.

Table 5: Example summary responses from simulation generated over-constrained queries after relaxation of user constraints. Various schemes for relaxation are shown. (1) relaxes on the geographical location, (2) offers a nearest alternative, and (3) removes the cuisine constraint, outputting a single alternate selection.

data subset has been narrowed down to six items or less, the simulator queries focus on one of the six items. For the 213 utterances, the recognition word error rate is 11.2%, and the sentence error rate is 32.4%. Because the synthesizer is highly domain specific and was originally trained on another domain, the synthetic waveforms were in fact highly unnatural. However, the relatively good recognition performance can be attributed to segmental units being well matched to the segment-based recognizer, an exact match to the trained n-gram model and the lack of spontaneous speech phenomena such as disfluencies. These 36 dialogs were analysed by hand. All dialogs successfully arrived at some small data subset at termination, without aborting due to errors. 29 (80.1%) of the dialogs completed without errors, with the correct desired data set achieved. Of the errorful dialogs, 3 exhibited problems due to recognition errors and 4 dialogs exhibited errors in the parse and context tracking mechanisms. All the questions regarding querying of individual restaurants were answered correctly.

5 Discussion

The above evaluations have been conducted on highly restricted scenarios in order to focus development on any fundamental problems that may exist in the system. In all, large numbers of synthetic dialogs have helped us identify problems that in the past would have been discovered only after data collections, and possibly after many failed dialogs with frustrated real users. The hope is that using simulation runs will improve system performance to a level such that the first collection of real user data will contain a reasonable rate of task success, ultimately providing a more useful training corpus. Having eliminated many software problems, a final real user evaluation will be more meaningful.

6 Related Work

Recently, researchers have begun to address the rapid prototyping of spoken dialog applications. While some are concerned with the generation of systems from on-line content (Feng et al., 2003), others have addressed portability issues within the dialog manager (Denecke et al., 2002) and the understanding components (Dzikovska et al., 2003).

Real user simulations have been employed in other areas of software engineering. Various kinds of human-computer user interfaces can be evaluated for usability, via employing simulated human users (Riedl and St. Amant, 2002; Ritter and Young, 2001). These can range from web pages to cockpits and air traffic control systems. Simulated users have also accounted for perceptual and cognitive models. Previous work in dialog systems has addressed simulation techniques towards the goal of training and evaluation. In (Scheffler and Young, 2000), extensive simulations incorporating user modeling were used to train a system to select dialog strategies in clarification sub-dialogs. These simulations required collecting real-user data to build the user model. Other researchers have used simulations for the evaluation of dialog systems (Hone and Baber, 1995; Araki and Doshita, 1997; Lin and Lee, 2001). In (Lopez et al., 2003), recorded utterances with additive noise were used to run a dialog system in simulation-mode. This was used to test alternate confirmation strategies under various recognition accuracies. Their methods did require the recording of scripted user utterances, and hence were limited in the variations of user input.

Our specific goals have dealt with creating more cooperative and flexible responses in spoken dialog. The issues of mismatch between user queries and database contents have been addressed by others in database systems (Gaasterland et al., 1992), while the potential for problems with dead-end dialogs caused by over-constrained queries have also been recognized and tackled in (Qu and Green, 2002).

7 Conclusions and Future Work

The use of a simulator has greatly facilitated the development of our dialog system, with the availability of thousands of artificial dialogs. Even relatively restricted synthetic dialogs have already accelerated development. In the next phase, real user data collection will be conducted, along with full-scale evaluation. We plan to compare the efficacy of our language models built from simulated data with those trained from real user data.

Future research will address issues of graceful recovery from recognition error. We believe that the framework of using simulated dialogs possibly with synthesized speech input augmented with controlled levels of additive noise can be an effective way to develop and evaluate error recovery strategies.

Current methods for simulating dialogs are quite rudimentary. The text only produces certain variants that have been observed but does not respect corpus statistics, nor, in the case of synthetic speech, do they account for spontaneous speech phenomena. Improved simulations could use a set of indexed real speech waveforms invoked by the core simulator to create more realistic input.

The main functionalities in the simulator software are now customizable from an external file. The simulator is domain independent and can be tailored for development of similar spoken dialog systems for browsing and navigating large databases. However further work is needed to incorporate greater configurability to the dialog flow. Increased flexibility for customizing the model of the dialog is needed to enable the software to be applied to the development of other kinds of dialog systems.

8 Acknowledgment

The author wishes to thank Stephanie Seneff for her valuable feedback and the anonymous reviewers for their insightful comments and suggestions.

References

M. Araki and S. Doshita. 1997. Automatic evaluation environment for spoken dialog system evaluation. In *Dialog Processing in Spoken Language Systems*, 183–194.

M. Denecke et al. 2002. Rapid Prototyping for Spoken Dialog Systems. *Proc. COLING*, Taipei, Taiwan.

M. Dzikovska et al. 2003. Integrating linguistic and domain knowledge for spoken dialog systems in multiple domains. *Proc. IJCAI*, Acapulco, Mexico.

J. Feng et al. 2003. Webtalk: Mining Websites for Automatically Building Dialog Systems. *Proc. IEEE ASRU*, Virgin Islands.

G. Ferguson and J Allen. 1998. TRIPS: An Integrated Intelligent Problem-Solving Assistant. *Proc. of the Fifteenth National Conference on AI (AAAI-98)*, 26–30. Madison, WI.

T. Gaasterland et al. 1992. An Overview of Cooperative Answering. Journal of Intelligent Information Systems, 1(2), 123–157.

J. Glass. 2003. A Probabilistic Framework for Segment-Based Speech Recognition. *Computer Speech and Language*, 17, 137–152.

K. Hone and C. Baber. 1995. Using a simulation method to predict the transaction time effects of applying alternative levels of constraint to user utterances within speech interactive dialogs. *ESCA Workshop on Spoken Dialog Systems*.

B. S. Lin and L. S. Lee. 2001. Computer-aided analysis and design for spoken dialog systems based on quantitative simulations. *IEEE Trans. on Speech and Audio Processing*, 9(5), 534–548.

R. Lopez-Cozar et al. 2003. Assessment of dialog systems by means of a new simulation technique. *Speech Communication*, 40, 387–407.

J. Polifroni, G. Chung and S. Seneff. 2003. Towards automatic generation of mixed-initiative dialog systems from web content. *Proc. EUROSPEECH*, 193–196. Geneva, Switzerland.

Y. Qu and N. Green. 2002. A Constraint-Based Approach for Cooperative Information-Seeking Dialog. *Proc. INLG*, New York.

M. Riedl and R. St. Amant. 2002. Toward automated exploration of interactive systems. *Proc. IUI*, 135–142.

F. Ritter and R. Young. 2001. Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. International Journal of Human-Computer Studies, 55, 1–14.

K. Scheffler and S. Young. 2000. Probabilistic simulation of human-machine dialogs. *Proc. ICASSP*, 1217–1220. Istanbul, Turkey.

S. Seneff et al. 1998. Galaxy-II: A Reference Architecture For Conversational System Development. *Proc. ICSLP*. Sydney, Australia.

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

V. Zue, et al. 2000. JUPITER: A Telephone-Based Conversational Interface for Weather Information *IEEE Transactions on Speech and Audio Processing*, 8(1).

J. Yi et al. 2000. A flexible, scalable finite-state transducer architecture for corpus-based concatenative speech synthesis. *Proc. ICSLP*. Beijing, China.