1-1-2002

Dialog Annotation for Stochastic Generation

Alexander I. Rudnicky
Carnegie Mellon University, ar28@andrew.cmu.edu

Alice H. Oh
Carnegie Mellon University

Dialog Annotation for Stochastic Generation

Alexander I. Rudnicky
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213 USA
air@cs.cmu.edu

Alice H. Oh
Massachusetts Institute of Technology
MIT Artificial Intelligence Laboratory
Cambridge, MA 02139, USA
aoh@mit.edu

Abstract

Individuals who successfully make their livelihood by talking with others, for example travel agents, can be presumed to have optimized their language for the task at hand in terms of conciseness and intelligibility. It makes sense to exploit this effort for the purpose of building better generation components for a spoken dialog system. The Stochastic Generation technique, introduced by Oh and Rudnicky (2002), is one such approach. In this approach, utterances in a corpus of domain expert utterances are classified as to speech act and individual concepts tagged. Statistical n-gram models are built for each speech-act class then used generatively to create novel utterances. These have been shown to be comparable in quality to human productions. The class and tag scheme is concrete and closely tied to the domain at hand; we believe this produces a distinct advantage in speed of implementation and quality of results. The current paper describes the classification and tagging procedures used for Stochastic Generation, and discusses the advantages and limitations of the techniques.

1 Introduction

In the process of developing and maintaining a natural language generation (NLG) module for a spoken dialog system, we found current NLG technologies to be of limited value. At the time, two approaches were available, *rule-based* (so-called "linguistic" approaches) and *template-based* approaches.

While several general-purpose rule-based generation systems have been developed (cf. Elhadad and Robin (1996)), because of their generality they are often quite difficult to adapt to small, task-oriented applications. Several different solutions have been proposed to overcome this shortcoming. For example, Bateman and Henschel (1999) have described a lower cost and more efficient generation system for a specific application using an automatically customized subgrammar. Busemann and Horacek (1998) describe a system that mixes templates and rule-based generation. This approach takes advantages of templates and rule-based generation as needed by specific sentences or utterances. Stent (1999) has also proposed a similar approach for a spoken dialog system. However, each one of these approaches still imposes the requirement of writing grammar rules and acquiring the appropriate lexicon, a specialist activity.

Because comparatively less effort and linguistic expertise is needed, many current dialog systems use template-based generation. But there is one obvious disadvantage to templates: the quality and appropriateness of the output depends entirely on the set of templates. Even in a relatively simple domain, as travel reservations, the number of templates necessary for reasonable quality can become quite large to the extent that maintenance becomes a serious

problem. There is an unavoidable tradeoff between the amount of time and effort in creating and maintaining templates and the variety and quality of the output utterances. This will become clear when we present the details of the template system we developed, in section 3.1.

In response to the limitations of rule-based and template-based techniques, we developed a novel approach to natural language generation. It features a simple, yet effective, corpus-based technique that uses statistical models of task-oriented language spoken by domain experts to generate system utterances. We have applied this technique to sentence realization and to content planning, and have incorporated the resulting generation component into a working spoken dialog system.

To evaluate our new generation module, we ran comparative evaluations on our spoken dialog system using different NLG components. We found that our new technique performs well for our spoken dialog system (i. e., users liked the output), and thereby demonstrated that the corpus-based NLG is a promising direction for further exploration. In addition, we gained some insight into the effectiveness of usability-based evaluation methods for NLG modules embedded within a larger system.

Full details of our work on stochastic generation are reported in Oh and Rudnicky (2002).

2 Natural Language Generation for Spoken Dialog Systems

Since current NLG systems have been mostly developed for text-based applications, applying existing NLG techniques directly to spoken dialog systems poses some problems. Recently, research has begun that looks at how to design effective NLG modules for spoken dialog systems (see for example, Ratnaparkhi (2000), Baptist and Seneff (2000), and Walker et al. (2001)).

A spoken dialog system enables human-computer interaction via spoken natural language. A task-oriented spoken dialog system speaks as well as understands natural language as part of the process of completing a well-defined task. Examples of research systems include a complex travel planning system (Rudnicky et al., 1999), a publicly available worldwide weather information system (Zue, 2000), and an automatic call routing system (Gorin et al., 1997).

Building an NLG component for a spoken dialog system differs from more traditional applications of NLG, such as generating documents, and provides a novel way of looking at the generation problem. The following are some characteristics of task-oriented spoken dialog systems that define unique challenges for spoekn dialog NLG.

1. *The language used in spoken dialog is different from the language used in written text.* Spoken utterances are generally shorter in length compared to sentences in written text. Spoken utterances follow grammatical rules, but much less strictly than written text. Also, the syntactic structures used tend to be much simpler and less varied than those in written text. In a spoken dialog, simple and short utterances may be easier to say and understand, since these impose a smaller cognitive load on the listener. More effective communication is thereby possible.

2. *The language used in task-oriented dialogs tends to be domain-specific.* The domains for these systems are fairly narrow. This means that the lexicon for a given spoken dialog system can be fairly small and domain-specific.

3. *Generation is often not the main focus in building or maintaining dialog systems.* Yet the NLG module is critical in development and system performance: In a telephone-based dialog system, NLG and text-to-speech (TTS) synthesis are the only components that users will experience directly. But with limited development resources, NLG has traditionally been overlooked by spoken dialog system developers.

Taking these characteristics into account, NLG for task-oriented spoken dialog systems must be able to

1. generate language appropriate for spoken interaction; system utterances must be easily comprehended, that is, not be too lengthy or be too complex (e. g., syntactically).

2. generate domain-specific language; the lexicon must contain appropriate words and expressions for the domain.

3. enable fast prototyping and development of the NLG module. A serviceable generation capability needs to be available fairly early in the development cycle.

Also, to promote the overall goals of the spoken dialog system, the NLG component must do its part in supporting a natural conversation. That is, the NLG output must elicit appropriate responses from the user, prevent user confusion, and guide the user in cases of confusion. We have observed that while the NLG component is not wholly responsible for creating these desirable characteristics of a spoken dialog system, it certainly can contribute (as seen in commercial dialog systems where system prompts are designed with great care to resolve these issues, see Kotelli (2001)).

3 Language Generation in the Carnegie Mellon Communicator

The Carnegie Mellon Communicator (Rudnicky et al., 1999) is a telephone-based spoken dialog system. It enables users to arrange complex travel itineraries via natural conversation. It is made up of several modules working together to provide a smooth interaction for the users while completing the task of arranging travel to fit the user's constraints.

The SPHINX-II automatic speech recognizer takes the user's speech and outputs a string. The PHOENIX semantic parser takes the string and turns it into a semantic frame. The AGENDA dialog manager (Xu and Rudnicky, 2000) handles the semantic frames from the PHOENIX parser and interacts with additional domain agents such as the date/time module, the user preferences database, and the flight information agent. When the dialog manager decides that something needs to be communicated to the user, it sends a frame to NLG, which then generates an utterance to be synthesized by the TTS module. The input frame from the dialog manager specifies a general "act", which is similar to a speech act, plus a domain-relevant "content" tag. Together these define the utterance class, or the equivalent of a dialog act. The information to be relayed to the listener is specified by attribute-value pairs, some of which may be hierarchically structured. The input frame contains no traditional linguistic features such as subject, patient, tense, head, etc. An example of the input frame is in Figure 1.

```
{
act query
content depart_time
depart_date {
year 2000
month 10
day 5
}
depart_airport BOS
}
```

What time on October fifth would you like to leave Boston?

Figure 1: An input frame to NLG in the Communicator

The Communicator NLG module is implemented in Perl. These and other implementation details have not changed much since its inception. What has changed, however, is that the system went from a purely template-based generation to corpus-based, stochastic generation.

3.1 Template-based generation

The Communicator NLG module started off with around 50 templates. The number of templates grew as we added more functionality to our system. The largest expansion came with the addition of a "help" speech act, which added 16 templates to provide context-sensitive help messages. Note that templates are not simple sentence frames with variable slots. They also need to include a computational component that deals with options For example, for the template "What time would you like to travel from {departure_city} on {departure_date}?", if the input frame did not contain values for the attributes {departure_city} and {departure_date}, instead of generating the ungrammatical sentence "What time would you like to travel from on?", it would generate "What time would you like to travel?". This reduces the number of templates significantly, but only at the expense of introducing more complexity to the

templates, especially for templates that can have as many as ten different attributes. Hence, the amount of time the developer needs to spend on crafting and maintaining the templates does not decrease significantly. At one point, the number of templates grew to nearly one hundred, some of them very complex and cumbersome to maintain. Axelrod (2000) has alluded to similar requirements in the system that he has described.

3.2 Development of corpus-based stochastic generator

What is perhaps more important than reducing development time is being able to generate utterances that promote a natural interaction with the user. One of the difficulties for a template writer is choosing the right words, the template system's equivalent of lexical selection. Often, the words that the template writer chooses for a given utterance are different from what the domain expert would use. This mismatch may hamper a smooth interaction because when a system utterance contains unfamiliar words in that domain, not only does it sound unnatural, but it may also lead the user to confusion or an inappropriate response.

One solution might be to base the generator on a corpus of task-oriented human-human conversations between a domain expert and a client. We could, for example, take the expert's utterances and use them directly as templates. This is very simple, but is not practical, as one would need to find an utterance for every possible combination of attributes.

The statistical n-gram language model provides an alternative representation. The n-gram language model has the advantage that it is simple to build and understand, and tools, such as the CMU-Cambridge Language Modeling Toolkit (Clarkson and Rosenfeld, 1997), are readily available. There is precedent in Langkilde and Knight's ((1998)) application of n-gram models to the problem of smoothing the output of a translation system. A possible objection is that while the n-gram language model captures well the relationships among words in the window of a length specified by the parameter n, it does not capture any long-distance dependencies beyond that window. We argue that this is not an issue for spoken dialog systems because the utterances are simpler in structure, and we can choose the parameter n to optimize performance. We will revisit this issue in a later section.

4 Modeling Human-Human Interaction

Since it is not clear what the "best practices" in designing human-computer spoken interactions are, especially in deciding what the system prompts ought to be, the obvious choice would be to use models of human-human interactions. Boyce and Gorin (1996) support this argument by their definition of a natural dialog: "[a dialog] that closely resembles a conversation two humans might have". Applying this definition to the language generation module of the spoken dialog system, we can build computational models of a domain expert having a conversation with a client, and use those models to generate system utterances.

For many domains, acquiring the correct lexicon items or grammar rules is not a trivial task, and to date, most researchers relied on informal methods of knowledge acquisition. Reiter et al. (2000), with their recent experiment of structured knowledge acquisition techniques, have begun exploring more principled ways of knowledge acquisition. Although the technique presented here is much simpler than theirs, concentrating mostly on acquisition of lexicon, it can be thought of as an efficient and effective way of automatically acquiring knowledge needed for a flexible language generation system.

4.1 Corpora

When building a statistical model, it is important to choose a data set that will provide statistics most similar to the underlying distribution. In many applications, such as a large vocabulary speech recognition system, this is not trivial. Collecting appropriate data for our generation engine, however, was not problematic. First of all, unlike the speech recognition system where the language model should be able to predict a wide variety of sentences that have never occurred, a language model for our generation engine does not require that predictive power. Secondly, collecting in-domain data is in any case the first step in building a task-oriented dialog system. Therefore, we can leverage the data collection effort and significantly reduce the amount of labor needed to build a corpus-based generation system.

counts	CMU	CMU	SRI	SRI
dialogs	39		68	
utterances	970	946	2245	2060
words	12852	7848	27695	17995

Table 1: Corpora used and their statistics. Counts are for Agent speech only

```
query arrive_city
query arrive_time
query confirm
query depart_date
query depart_time
query pay_by_card
query preferred_airport
query return_date
query return_time
hotel car_info
hotel hotel_chain
hotel hotel_info
hotel need_car
hotel need_hotel
hotel where
inform airport
inform confirm_utterance
inform epilogue
inform flight
inform flight_another
inform flight_earlier
inform flight_earliest
inform flight_later
inform flight_latest
inform flight_returning
inform not_avail
inform num_flights
inform price
other
```

Figure 2: Utterance classes

We examined two available corpora in the travel reservations domain for suitability for building n-gram language models for generation. One corpus (henceforth, the CMU corpus) consists of 39 dialogs between a travel agent and clients (Eskenazi et al., 1999). Another corpus (henceforth, the SRI corpus) consists of 68 dialogs between a travel agent and users in the SRI community (Kowtko and Price, 1989). Table 1 presents some statistics for the two corpora.

4.2 Speech act and concept tagging

Instead of just one language model for all the system utterances, we use a language model for each utterance class. This requires that the domain expert's utterances in the training corpora be tagged with the utterance classes. Figure 2 shows the list of utterance classes that was used to tag the corpora. A total of 29 classes were used, reflecting the communicative needs of the dialog system. Utterances that did not fall into these categories were classed as `other` and were not subsequently used for generation.

```
airline
am
arrive_airport
arrive_city
arrive_date
arrive_time
car_company
car_price
connect_airline
connect_airport
connect_city
depart_airport
depart_city
depart_date
depart_time
depart_tod
flight_num
hotel
hotel_city
hotel_price
name
num_flights
pm
price
```

Figure 3: Word classes

Another set of tags is necessary for word classes (see Figure 3). This is similar to the use of word classes in class-based language models, where, for example, words indicating numerals (e.g., one, ten, twelve hundred) are treated as a single unique word in the n-gram model, and then there is a separate model for the distribution of those words in the class. Similarly, in our model, we replace the words in a word class with the tag, treating them as a single unique word for purposes of sentence modeling.

A separate set of class models is not needed, however, since the word classes represent the attributes, for which the values are passed in the input frame from the dialog manager. This is similar to having slots in templates, and then filling the slots with the values in the input.

Much of the tagging effort can be automated. A list of words for most of the word classes is already available as a byproduct of overall system development: city names, airports, airlines, etc. However, as can be seen in Figure 3, some of the word classes need to be more detailed. For example, the class "city" needs to be differentiated into *departure city* (`depart_city`) and *arrival city* (`arrive_city`). Tagging first with the more general class, and then applying simple rules such as "from {city}" becomes "from {depart_city}" produces the desired result.

Tagging utterance classes requires a little bit more

```
<utt> okay {airline1} flight leaves {depart_city1} at {depart_time_actual1} {pm} <\utt>
<utt> okay {airline1} has a {nonstop} at {depart_time_actual1} {am} <\utt>

<utt> and what time would you like to leave {depart_city} on {depart_date} <\utt>
<utt> and what time would you like to leave {depart_city}? <\utt>
```

Figure 4: Examples of fully marked up utterances

care. Still, it can be semi-automated using machine learning. We first tag a subset of utterances manually, then iteratively learn the rules from the tagged utterances (with corrections applied to the classification on each iteration). What results from this is a decision tree, where the features can be word classes, phrases or single words. This technique is a simple application of an unsupervised learning technique (with some seed rules) used for named entity extraction in Collins and Singer (1999). Figure 4 shows an example of marked up utterances.

4.3 Using models of human-human interaction for human-computer interaction

Several issues arise when using computational models of human-human interaction for spoken dialog systems. First, there are some inherent differences between human-human conversations and human-computer conversations. As Boyce and Gorin (1996) point out, there are "user" and "system" utterances in human-computer interactions that do not occur in normal human-human interactions. These include more frequent confirmations, error messages, and help messages. Similarly, there are some utterances that occur frequently in human-human conversations but not in human-computer conversations, the most obvious being backchanneling (e.g., "uh-huh", "okay").

The second issue is that the quality of the output depends very much on the speaker whose language is modeled. This means the selection of a speaker is crucial for system performance. In the case of a travel reservation system, recruiting an experienced travel agent (as we did) may be sufficient, but in other domains it may not be as simple. Another issue is that while the models of human-human interaction may result in natural dialogs, they may not lead to the most efficient dialogs. That is, it may be possible that one could design a computer agent that can carry out a more efficient task-oriented dialog than a human expert. Such issues are beyond the scope of the current paper.

5 Surface Realization

If a natural human-computer dialog is one that closely resembles a human-human conversation, the best method for generating natural system utterances would be to mimic human utterances. In our case, where the system is acting as a travel agent, the solution would be to use a human travel agent's utterances (see Section 5 for details about the training data). The computational model we chose to use is the simple n-gram model familiar from speech recognition (Jelinek, 1998).

5.1 Implementation

We have implemented a hybrid language generation module incorporating three different techniques: fixed templates (e.g., "Welcome to the Carnegie Mellon Communicator."), variable templates (e.g., "Hello *Alice*"), and corpus-based stochastic generation. Certain prompts, for example a system greeting at the outset of the dialog, can be generated by a fixed expression. Other simple utterances can be generated efficiently by templates; these include utterances that are specifiic to the operation of the system and would otherwise not be observed in a natural human–human corpus. Then, for the remaining utterances where there is a good match between human-human interaction and human-computer interaction, we use statistical language models. In the CMU Communicator, approximately half of all utterances produced by the system will be stochastically generated, across typical calls. In normal operation, we observed that approximately half the utterances output by the system were generated by the stochastic process.

There are four aspects to our stochastic surface realizer: building language models, generating candidate utterances, scoring then selecting among the utterances, and filling in the slots. Figure 5 shows

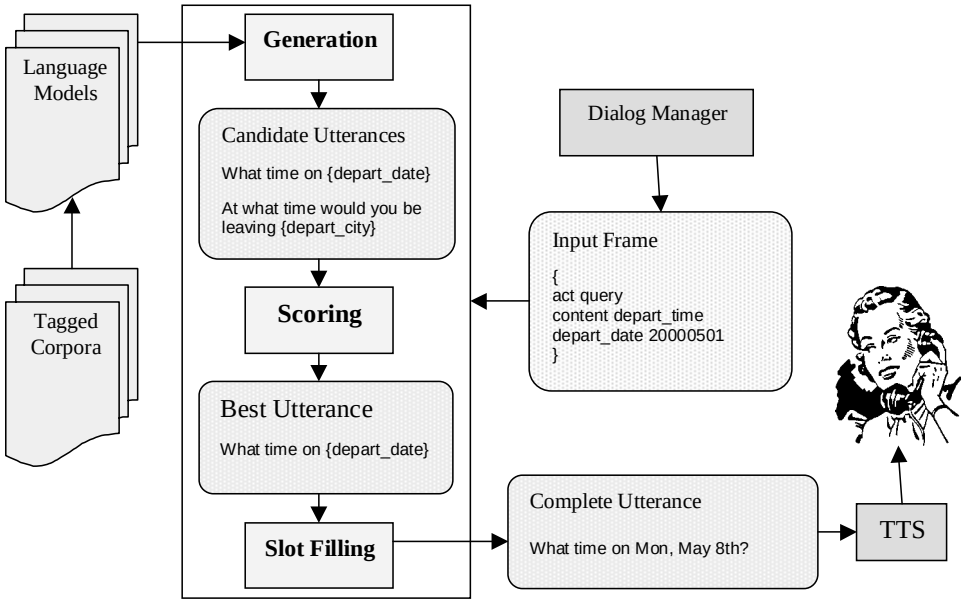

Figure 5: Online and offline components of a stochastic natural language generator

how the four components work together, and we describe each in detail below.

5.1.1 Building Language Models

Using the tagged utterances as described in the introduction, we built an unsmoothed n-gram language model for each utterance class. We settled on 5-gram models so as to balance variability in the output utterances with the need to minimize the generation of nonsense utterances. In fact utterances acceptable to humans can be generated with 2- and 3-gram models (see the results of a user experiment varying the parameter n in Table 2). It's an open question whether smoothing will have a desirable or undesirable effect on the generative use of language models; however it does not appear to us to be a well-formed question: there is indeed a critical need, in recognition, to allow for unseen word sequences (at whatever low probability). There doesn't appear to be a corresponding need for this property in generation: the models are quite fertile as is (see Table 3) and there is no need for novelty per se.

Note that language models are not used here in the same way as in speech recognition. In speech recognition, the language model probability acts as a prior in determining the most probable sequence of words given the acoustics. In other words,

$$W^* = argmax P(W|A) \quad (1)$$

n	Mean (*inform*)	Mean (*query*)
1	4.48	4.53
2	2.25	1.75
3	2.10	1.62
4	1.88	1.48
5	1.86	1.32

Table 2: How 63 subjects rated sentences generated by n-grams (1 is very easy to understand, 5 is very difficult to understand) for the *inform* speech acts and the *query* speech acts.

Utterance Class	Corpus	Output
inform_flight	37	445
inform_flight_earlier	6	54
inform_flight_later	9	340
query_depart_date	22	61
query_depart_time	20	74

Table 3: The number of unique utterances in the output generated by 5-grams compared with the number of unique utterances in the original corpus.

$$= argmax P(A|W)P(W) \quad (2)$$

where W is the string of words, $w_1, ..., w_n$, and A is the acoustic evidence (Jelinek, 1998).

Although we use the same statistical tool to compute the model, we use language model probabilities directly to generate the next word. In other words, the most likely utterance is $W^* = argmax P(W|u)$, where u is the utterance class. We do not, however, look for the most likely hypothesis, but rather generate each word randomly according to the distribution, as illustrated in the next section.

5.1.2 Generating Utterances

The input to NLG from the dialog manager is a frame of attribute-value pairs. The first two attribute-value pairs specify the utterance class. The rest of the frame contains word classes and their values. See Figure 1 for an example of an input frame to NLG.

The generation engine uses the appropriate language model for the utterance class and generates word sequences randomly according to the language model distributions. As in speech recognition, the probability of a word using the n-gram language model is

$$P(w_i) = P(w_i|w_{i-1}, w_{i-2}, ...w_{i-(n-1)}, u) \quad (3)$$

where u is the utterance class. Since we have built separate models for each of the utterance classes, we can ignore u, and say that

$$P(w_i) = P(w_i|w_{i-1}, w_{i-2}, ...w_{i-(n-1)}) \quad (4)$$

using the language model for utterance class u.

Stochastic generation is highly fertile, as seen in the number of distinct utterances generated for two representative speech acts (one simple, one complex), shown in Table 3. There is the problem, as in speech recognition using n-gram language models, that long-distance dependency cannot be captured. However this does not appear to significantly affect the comprehensibility of the utterances.

We speculate that this is because listeners (as opposed to readers) are not particularly sensitive to minor grammatical errors in what a talker produces. After all, errors in production are relatively common in spoken language, and their occurrence is likely to be considered unremarkable, particularly in the context of goal-directed conversations where the focus of the interaction is on information transfer and problem-solving. Based on the results shown in Table 3, we might speculate further that in concrete domains listeners might be satisfied as long as the are able to easily extract task-relevant conceptual content from an utterance.

5.1.3 Scoring Utterances

For each randomly generated utterance, we compute a penalty score. The score is based on the following heuristic rules. Penalties are assigned for the following:

1. The utterance is too short or too long (determined by an expectation derived from the length observed for that class of utterance in the corpus).

2. The utterance contains repetitions of any of the slots.

3. The utterance contains slots for which there is no valid value in the frame.

4. The utterance lacks some of the required slots (see section 6 for deciding which slots are required).

The generation engine generates a candidate utterance, scores it, keeping only the best-scored utterance up to that point. It stops and returns the best utterance when it finds an utterance with a zero penalty score, or it reaches the limit of 50 iterations. The average generation for the longest utterance class (10-20 words long) is about 200 milliseconds (on a 400MHz Pentium computer).

This search strategy is depth-first, where the full sentence is $w = \{w_1, ..., w_n\}$, the search tree depth is n, and each word w_i is a path down the search tree. A similar corpus-based system in (Ratnaparkhi, 2000) employs breadth-first search.

5.1.4 Filling Slots

The last step is filling slots with the appropriate values from the input frame. For example, the utterance "What time would you like to leave {depart_city}?" becomes "What time would you like to leave New York?". A final filter smoothes

local agreement (e. g., *a american flight → an american flight*). There is no attempt to control for non-local agreement.

6 Annotation and Processing issues in Stochastic Generation

The procedure for creating a stochastic generator, as outlined in this paper, is quite straightforward and we believe that it is reducible to a set of tools and principles that can be readily understood and used by language technologists and other developers to create high-quality generation modules for a range of applications.

The stochastic technique has the advantage of modeling a domain expert's speech, and this can be done without direct participation of the domain expert in the development process. It's sufficient to enlist their cooperation in the collection of speech data. We have found that word-level transcription (as needed for this use) is straightforward and is a skill that can be taught to a broad range of individuals with otherwise no specialized linguistic training (Bennett and Rudnicky, 2002). The classification of utterances and concept tagging does require some expertise, minimally familiarity with the ontology used by the automatic system and the ability to deal with, for example, utterances with multiple speech acts which need to be split prior to coding.

We have not attempted to reduce the coding process to a fixed procedure supported by appropriate tools or to systematically investigate whether a non-specialist can learn to do it at an acceptable level of accuracy. We are however confident that this would be possible to do.

References

S. Axelrod. 2000. Natural language generation in the ibm flight information system. *Proceedings of ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 21–26, May.

L. Baptist and S. Seneff. 2000. Genesis-ii: a versatile system for language generation in conversational system applications. *Proceedings of International Conference on Spoken Language Processing*.

J. Bateman and R. Henschel. 1999. From full generation to 'near-templates' without losing generality. *Proceedings of the KI'99 workshop, "May I Speak Freely?"*.

C. Bennett and A. I. Rudnicky. 2002. The Carnegie Mellon Communicator corpus. In *Proceedings of ICSLP*, pages 341–344.

S. Boyce and A. Gorin. 1996. User interface issues for natural spoken dialog systems. *Proceedings of the International Symposium on Spoken Dialog*, pages 65–68, October.

S. Busemann and H. Horacek. 1998. A flexible shallow approach to text generation. In Eduard Hovy, editor, *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 238–247. Association for Computational Linguistics, New Brunswick, New Jersey.

P. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the cmu-cambridge toolkit. *Proceedings of Eurospeech97*.

M. Collins and Y. Singer. 1999. Unsupervised models for named entity classification. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

M. Elhadad and J. Robin. 1996. An overview of surge: A reusable comprehensive syntactic realization component. *Technical Report 96-03, Dept of Mathematics and Computer Science*.

M. Eskenazi, A. Rudnicky, K. Gregory, P. Constantinides, R. Brennan, C. Bennett, and J. Allen. 1999. Data collection and processing in the carnegie mellon communicator. *Proceedings of Eurospeech*, 6:2695–2698.

A.L. Gorin, G. Riccardi, and J.H. Wright. 1997. How may i help you? *Speech Communication*, 23:113–127.

F. Jelinek. 1998. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts.

B. Kotelli. 2001. Blancusi, neo-plasticism, and the art of speech-recognition applications. *Proceedings of the Automatic Speech Recognition and Understanding Workshop*.

J. Kowtko and P. Price. 1989. Data collection and analysis in the air travel planning domain. *Proceedings of DARPA Speech and Natural Language Workshop*, October.

I. Langkilde and K. Knight. 1998. The practical value of n-grams in generation. *Proceedings of International Natural Language Generation Workshop*.

A. H. Oh and A. I. Rudnicky. 2002. Stochastic natural language generation for spoken dialog. *Computer Speech and Language*, 16(3/4):387–407.

A. Ratnaparkhi. 2000. Trainable methods for surface natural language generation. *Proceedings of the ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 194–201.

E. Reiter, R. Robertson, and L. Osman. 2000. Knowledge acquisition for natural language generation. *Proceedings of the First International Natural Language Generation Conference*, pages 217–224, June.

A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. 1999. Creating natural dialogs in the carnegie mellon communicator system. *Proceedings of Eurospeech*, 4:1531–1534.

A. Stent. 1999. Content planning and generation in continuous-speech spoken dialog systems. *Proceedings of the KI'99 workshop, "May I Speak Freely?"*.

M. Walker, O. Rambow, and M. Rogati. 2001. Spot: A trainable sentence planner. *Proceedings of the North American Meeting of the Association of Computational Linguistics*.

W. Xu and A. Rudnicky. 2000. Task-based dialog management using an agenda. *Proceedings of ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 42–47, May.

V. Zue. 2000. Jupiter: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1), January.