Hybrid Natural Language Generation for Spoken Dialogue Systems

Michel Galley, Eric Fosler-Lussier, and Alexandros Potamianos

Dialogue Systems Research Department
Bell Labs, Lucent Technologies, New Jersey (USA)
{galley,fosler,potam}@research.bell-labs.com

Abstract

The natural language generation component of most dialogue systems is based on templates. Template-based generators are hard to maintain and reuse, and the sentences they produce lack the variability and robustness needed by conversational systems. In this paper, a flexible and domain-independent natural language generator for spoken dialogue systems is proposed which combines fixed surface expressions with freely generated text. The generation algorithm follows a hybrid approach, combining finite state machine (FSM) grammars and corpus-based language models. In this approach, the FSM grammar (a reversible parser grammar) is constrained by a word and concept n-gram that takes terminals and non-terminal co-occurrences into account. The n-gram grammar helps prevent inappropriate derivations, therefore improving the quality of the generated texts. The proposed algorithm achieves faster than real-time performance because of the limited number of derivations.

1. Introduction

Natural language generation (NLG) has been extensively studied for 'single-interaction' systems, e.g., summarizers, translators, and report generators, but little is known about effective ways of performing NLG in dialogue systems. Current approaches to NLG have limited success when applied to conversational systems. The naturalness and perceived intelligence of a spoken dialogue interface depends heavily on the qualities of the natural language generation module. The language use must be extensive and varied [13]. In addition, the NLG module has to be robust against missing and incomplete data [6], a problem that occurs often in spoken dialogue systems. Another requirement in spoken dialogue systems is real-time performance; the NLG module should operate in a fraction of real time.

Template-based systems require little linguistic expertise and are cost-effective solutions to NLG in the early stages of prototyping. However, templates tend to become unmanageable as the system grows, since the number of templates needed to cover all situations while maintaining a reasonable quality can become quite large. In addition, templates are application specific and have to be entirely rewritten when switching to a new application domain. Other problems that template-based NLG systems face include an overly restrictive use of language that limits variability in the generated text, and vulnerability to missing data, since they merely rely on slot-filling techniques. Overall, the time and energy needed to achieve a reasonable quality, variety, and robustness using templates is very high for large applications.

General-purpose rule-based generation systems like FUF/SURGE [5] and KPML [3] sidestep these problems, offering a broad coverage of English, but because of their generality, sophistication, and their need for a large amount of linguistic knowledge, such systems tend to be difficult to adapt to task-oriented applications. In addition, typical rule-based generators do not achieve real-time performance and are unsuitable for dialogue systems [9, 13].

Recently, attempts have been made to overcome the problems of both template-based and rule-based systems by introducing a hybrid approach incorporating both models [4, 8, 13]. In [1], Axelrod proposes a model where sub-phrases in the templates can be "turned-on" depending on the semantic input. In [9], Oh and Rudnicky propose a purely stochastic surface realization. Stochastic surface realization is flexible and fast but it runs the unavoidable risk of generating ungrammatical and ill-formed sentences.

In this paper, we propose a flexible framework for natural language generation in dialogue systems that combines both rule-based and stochastic models. Statistical language models reduce the need for formal linguistic characterization. Simple and intuitive semantic rules can effectively be used to obtain an output of sustainable quality, "by relegating some aspects of lexical choice ... to the statistical component" [6]. In addition, we show how this framework is robust against missing data, and how it is possible to introduce variability in the output with minimal efforts. The proposed algorithm can also be implemented to run in a fraction of real-time. This framework has been implemented in the Bell Labs Communicator system, a task-based information-seeking spoken dialogue system. It is currently dedicated to the travel reservation domain, but its components are portable across multiple domains. Our overall goal, is to decrease prototyping time and effort by creating application-independent tools and algorithms that automate the design process and that can be used by non-expert application developers.

2. System Overview

In dialogue systems, natural language generation is the task of automatically producing utterances from an abstract semantic representation. In [13], Stent proposes that natural language generators should consist of three layers: planning intention, content, and form. In dialogue systems, determining the system's intent, or communicative goal, is out of the scope of the generator and is usually handled by the dialogue manager. The two remaining tasks are respectively performed by the utterance planner and the utterance realizer described next.

3. Utterance Planner

The main function of the utterance planner is to determine the content of the system prompt according to the communicative goal selected by the dialogue manager. A typical goal frame produced by the dialogue manager is shown below:

```
[(GOAL
   :intention     inform
   :predicate     updatedValues
   :context       .trip.flight.leg1)]
```

System prompts are categorized according to the system's 'intention' which corresponds to a general communicative goal, e.g., 'inform'. The fields 'predicate' and 'context' further constrain the prompt type by specifying a sub-goal, e.g., inform the user about changes to the value of attributes, and the semantic sub-tree that the prompt refers too, e.g., the first leg of a flight (see [2, 10] for details on the semantic tree representation). The task of the utterance planner is to select the appropriate semantic content for each category of system prompts. The semantic content is defined here as a set of attribute-value pairs, (e.g., `CITY:NEWARK`).

Selecting the appropriate set of attribute-value pairs to include in the utterance is a hard problem that is beyond the scope of this paper. Many sources of information have to be combined to determine what information is relevant to the user including: communicative goal and subgoal, dialogue context, ontological representation of the domain, user input, system output, dialogue history and user preferences.

In order to introduce variability in the system prompts, several syntactic structures and lexicalisations are considered for the same communicative goal. Such alternative forms, or phrasal patterns, e.g., `Now you're flying` <FROMCITY> <TOCITY>, contain non-terminal symbols, e.g. <FROMCITY> that abstract away all the details about the semantic content, e.g., city, state, airport, country. Such a symbol corresponds to a concept of the application domain, which can have different lexicalisations as discussed in the next section e.g., `Chicago`, `Chicago O'Hare`, and `O'Hare airport in Chicago` are alternative lexicalisations of the <CITY> symbol. The output of the utterance planner consists of a list of phrasal patterns and scores, and a list of values (constraints on the patterns). For example, given a change in the value of the departure city and the aforementioned communicative goal the output of the utterance planner might be:

```
[(PATTERNS
   'Now I have you leaving <FROMCITY> <TOCITY>'      0.92
   'Now you're flying <FROMCITY> <TOCITY>'           0.92
   'Now I have you leaving <FROMCITY>'               0.8
   'Ok, your new flight leaves <FROMCITY>')          0.8
(CONTENT
   :depCity     'Athens'
   :depState    'Georgia'
   :arrCity     'Newark')]
```

Note that each phrasal pattern may contain a different set of attributes. The scores associated with each phrasal pattern are a by-product of the context selection algorithm and are solely a function of which attributes the phrasal pattern contains.

4. Utterance Realizer

Utterance realization is the process of producing text that is syntactically and morphologically correct, while consistent with the input from the utterance planner. In this process, a language model, e.g., a generative grammar, is used. As discussed in the previous section, the utterance planner provides the utterance realizer with different phrasal patterns, e.g., `What time would you like to get in` <TOCITY>`?`. The main role of the utterance realizer is to convert into surface expressions the non-terminals, e.g., <TOCITY>, for each of the phrasal patterns and rank order the produced surface realizations. Value constraints, e.g., ':arrCity' is 'Newark', need to be satisfied. Robustness towards missing or underspecified semantic information is also expected.

A fully linguistically motivated utterance realizer can produce high-quality text output and is easy to maintain and extend; however, such a system depends on large amounts of hand-crafted knowledge. Statistical models don't require a lot of linguistic resources, but tend to generate ill-formed or ungrammatical sentences. In this section, a hybrid generation approach is described that combines the flexibility and simplicity of template-based systems with the power, scalability, and maintainability of complex rule-based NLG systems.

4.1. Rule-based Language Model

Phrasal patterns contain non-terminal symbols, which correspond to concepts in the application domain. These symbols are recursively expanded by the rules of a semantic generative grammar, until the phrasal patterns contain only terminal symbols (words). We say that this grammar is semantic because it does not contain a detailed analysis of the syntax of the language; rather, it encodes knowledge about how lexical patterns co-occur within the application domain to form concepts.

The recursive weighted finite-state grammar in our system is roughly equivalent to a probabilistic context free grammar (PCFG). Unfortunately, expanding phrasal patterns via a PCFG has some limitations. By the definition of context-free, the expansion of any PCFG rule is performed independently of any other expansion. In natural languages, however, the realization of a phrase usually depends on its location in the parse tree [7]. For example, consider the difference between placing a pronoun in a sentence (e.g. `he gave` *it* `to Bill`), and placing a corresponding noun phrase (`He gave Bill` *the ball*); in the latter sentence you cannot replace *the ball* with *it*. PCFGs lack the ability to model horizontal dependencies within and across rules. Similarly, another problem with PCFGs is their lack of sensitivity to words: the probability of their parses only depends on structural factors; lexical co-occurrences are not taken into account, and consequently many inter-lexical constraints are ignored.

In the context of our NLG component, an even more important problem with PCFGs is the assumption we have made about the application developer: we shouldn't expect him to possess any linguistic expertise. Badly hand-written rules might lead the NLG component to generate ungrammatical sentences. Furthermore, the developer is likely to keep adding new rules to expand the coverage and the variability of the output language, thus leading to an over-generation problem: the combinatorial factors might lead the NLG module to generate a vast number of possible utterances, rendering the problem of storing them and choosing the best one difficult. There do exist ways to represent large amount of alternative sentences in compact structures like lattices. However, it would be best to reduce the search space by weeding out bad candidates in the first place.

4.2. Statistical Language Model

Given the shortcomings of a pure PCFG generation approach detailed in the previous section, we propose an approach which prunes out sentences that are unlikely to be generated by ranking them with a second, lexically-oriented model. A word-based n-gram grammar lets us take lexical co-occurrences into account within surface word strings. Furthermore, with a

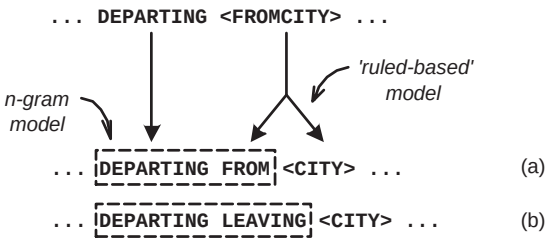

Figure 1: Generation by the rule based system of two possible instantiations of DEPARTING <FROMCITY>. (a) A possible expansion, rated highly by the n-gram grammar. (b) A less likely expansion, rated poor by the n-gram grammar.

'word-concept' n-gram, i.e., an n-gram trained on terminals and nonterminals of the semantic grammar, we can put some contextual conditions on the expansion of a node according to its location in the parse tree. This technique is similar to a parsing strategy currently used in our lab [11].

In Fig. 1, we show an example where we can expand <FROMCITY> into FROM <CITY>, but not into DEPARTING <CITY>, because of the context: the previous word is LEAVING, and the n-gram model tells us that the probability of LEAVING DEPARTING is very close to zero. While zero-frequency counts are a problem in speech recognition language models, from a generative point of view, we can use this information to delete very unlikely lexical co-occurrences from output sentences.

This approach has many advantages: by using the n-gram grammar to prune unlikely expansions, the combinatorial explosion problem we would encounter if we let the semantic grammar generate all sentences it can produce is greatly reduced. Furthermore, this architecture enables us to achieve our goal of not relying on linguistic expertise: it is flexible enough to cope with semantic grammar rules written by non-experts (like those of the previous example). Finally, by training an n-gram on a sequence of concepts and words like <AIRPORT> in <CITY>, we can represent co-occurrences in concepts in a more compact way than if were were modeling strings like JFK AIRPORT IN NEW YORK. It offers a better understanding of the underlying language and increases the power and robustness of the statistical language model.

5. Implementation

As stated above, the utterance realizer creates utterances given phrasal patterns and a set of attributes by integrating two statistical models. In our implementation, the models have been encoded as finite state machines (FSM), using the AT&T Bell Labs Finite State Machine (FSM) library [12]. The encoding scheme is very similar to that of our finite-state recursive parser [11].

In our architecture, the set of context-free rules are implemented as a finite state transducer (FST), labelled R. Left-hand sides of the rules are mapped to the input alphabet of the FST, and right-hand sides to the output alphabet. Epsilon (null) labels, denoted as "eps", are added to the left-hand side of the rule, so that the number of input and output symbols are equal. Fig. 2 illustrates the encoding of some semantic rules are in the FST R.

We also define a second FSM, N, which is a weighted finite state automaton (FSA) that encodes all transitions of the 'word-concept' n-gram, including back-off probabilities. Finally, a

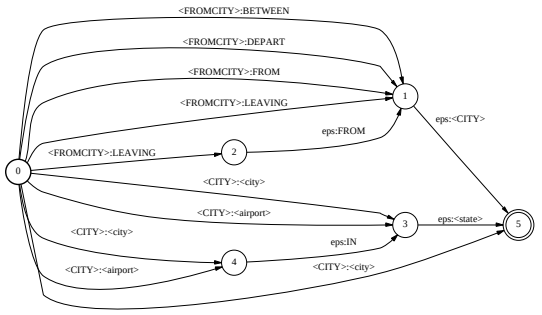

Figure 2: The FST R of a simple semantic grammar

weighted FSA P is constructed that represents all alternative phrasal patterns.

The expansion of the phrasal patterns is done by recursively composing P with both language models, followed by taking the output projection (i.e., discarding the input symbols):

$$L = R \circ N \quad (1)$$

$$P_0 = P \quad (2)$$

$$P_n = (P_{n-1} \circ L)_{\text{proj 2}} \quad (3)$$

The recursion terminates when P_n equals P_{n-1} — when a recursion doesn't create any new text span[1]. Checking whether two automata are equivalent is computationally expensive, since both automata have to be minimized in order to be compared. But a simple analysis of the rules can determine what is the maximum number of recursion, since it is equivalent to the maximum depth of the generative trees. The resulting automaton is a string composed of only words and attributes. Fig. 3 illustrates the expansion of a phrasal pattern.

The resulting FSM is then composed with FSAs encoding the constraints. For example, if the utterance planner specifies that the utterance should contain exactly the attributes <airport>,<state>,<digit>, and <ampm>, all paths in the output FSM that don't satisfy those conditions are eliminated. A *n*-best search through the resulting FSM determines a candidate set of generated sentences to choose from.

6. Discussion

The example in Fig. 4 shows how the 'word-concept' n-gram can help reducing the number of ungrammatical sentences. We have run the NLG module in two different situations, once solely using the semantic rule-based model, and once with both the rule-based and n-gram (trigram) models. Both systems were seeded with the same phrasal pattern and the same values for the database attributes. Since a n-gram model is well suited to catch short-term dependencies (e.g., agreement between a subject and a verb), many ungrammatical generations are avoided. We noticed that most of the incorrect words appear at the edges of either the fixed part or the freely generated part of the phrasal pattern. This is the case in error E1, where an error occurs right between the edges of both fixed (flying) and variable (Newark at 7:20am) parts. N-gram models also help to avoid error like in errors E3 and E4, because they correspond to unseen sequence of words. However, error E2 may not be avoidable, even with a trigram.

[1] A span is a portion of a sentence or utterance.

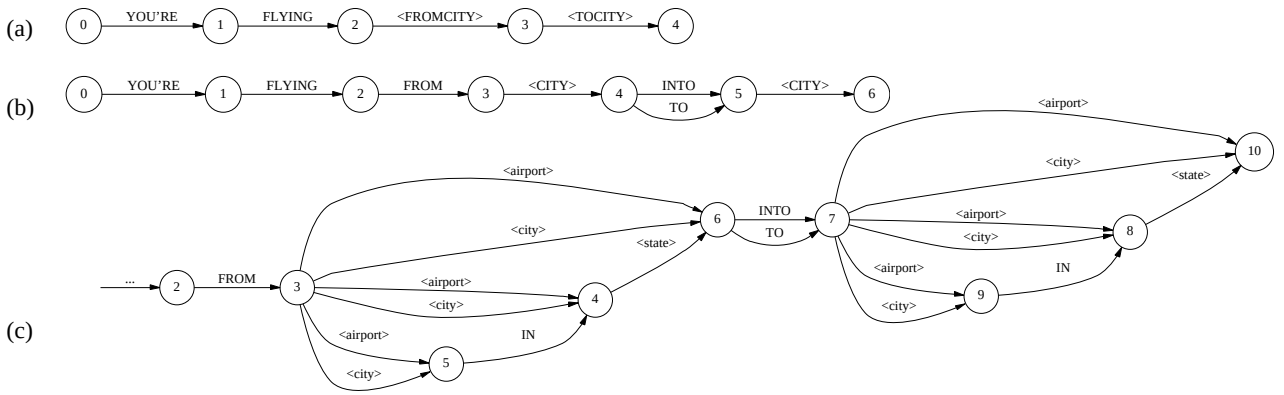

Figure 3: The FSA representing a single phrasal pattern and the two first steps of the surface realization mechanisms.

phrasal pattern:
. . . flying <FROMCITY> <FROMTIME> <TOCITY> <TOTIME>

a: no n-gram weights
. . . flying from Newark at 7:20am to Boston arriving in Boston at 8:00am . . .
. . . flying Newark^{E1} at 7:20am to Boston that arrive^{E2} at 8:00am . . .
. . . flying from Newark leaving at at^{E3} 7:20am to Boston arriving at 8:00am . . .
. . . flying from Newark leaving at 7:20am to Boston arriving 8:00am^{E4} . . .

b: with n-gram weights
. . . flying from Newark at 7:20am to Atlanta arriving in Boston at 8:00am . . .
. . . flying from Newark leaving at 7:20am and arriving in Boston at 8:00am . . .
. . . flying from Newark leaving at 7:20am and arriving at 8:00am in Boston . . .
. . . flying at 7:20 from Newark and arriving in Boston at 8:00am . . .

Figure 4: Sample four best outputs from rule-based generator (a) without n-gram weights, and (b) with n-gram weights. Errors are marked as a superscript En.

While anecdotally our system seems to be getting good results, evaluating a NLG system is, unfortunately, a difficult task. Quantitative aspects like speed and memory usage are easy to assess, but it is more difficult to appraise qualitative aspects. Evaluating the NLG component of a spoken dialogue system is even more problematic, since it is hard to separate it from other components of the system, especially the text-to-speech engine [9]. In future work, we hope to develop an experimental paradigm to evaluate this system against our older, template-based generator.

7. Conclusions

We have proposed a new application-independent approach for natural language generation in dialogue systems that is robust against missing data and that renders the expected variation in the output.

Statistical methods provide the flexibility to deal with underspecified linguistic knowledge, while semantic rules provide some control as to what phrases should or should not be generated. The system can produce sentences in a fraction of real-time, since most of the ungrammatical expansions are automatically pruned by the stochastic language model.

Our natural language generation module also minimizes development effort: the stochastic language model can accommodate simple semantic rules that do not require linguistic expertise, making rules far simpler to write than detailed syntactic rules. Furthermore, the rules from the semantic grammar of the dialogue system's parser can be used to seed rule construction. We anticipate that, due to its flexibility, the system will be easy to adapt for other domains.

Acknowledgments: This work was partially funded by DARPA under the auspices of the Communicator project. The authors would like to express their sincere appreciation to Chin-Hui Lee, Joe Olive, Gerald Penn, Jeff Kuo, Andy Pargellis and Egbert Ammicht for many helpful discussions.

8. References

[1] S. Axelrod. "Natural Language Generation in the IBM Flight Information System", in *Proc. of ANLP-NAACL'2000, pp. 21-26, (Seattle, Washington), May 2000.*

[2] E. Ammicht, A. Potamianos, and E. Fosler-Lussier, "Ambiguity Representation and Resolution in Spoken Dialogue Systems," submitted to *EUROSPEECH*, (Aalborg, Denmark), 2001.

[3] J. Bateman. "KPML Development Environment: multilingual linguistic resource development and sentence generation.", *German National Center for Information Technology (GDM), IPSI, (Darmstadt, Germany), Jan. 1997.*

[4] S. Busemann, H. Horacek. "A Flexible Shallow Approach to Text Generation", in *Proc. INLG'98, pp. 238-247, (Niagara-on-the-Lake, Canada), Aug. 1998.*

[5] M. Elhadad, J. Robin. "An overview of SURGE: A reusable comprehensive syntactic realization component." *Technical Report 96-03, Dept. of Mathematics and Computer Science, Ben Gurion University, (Beer Sheva, Israel), 1996.*

[6] K. Knight and V. Hatzivassiloglou. "Two-Level, Many-Paths Generation", In *Proc. of the ACL-95. (Boston, Mass.), 1995.*

[7] S. Kuno. "Functional Sentence perspective: A case study from Japanese and English", *Linguistic Inquiry, volume 3, pp. 269-320, 1972.*

[8] I. Langkilde, "Forest-Based Statistical Sentence Generation", in *Proc. ANLP-NAACL'2000, pp. 170-177, May 2000.*

[9] A.H. Oh and A.I. Rudnicky. "Stochastic Language Generation for Spoken Dialogue Systems", In *Proc. of ANLP-NAACL'2000, pp. 27-32, (Seattle, Washington), May 2000.*

[10] A. Potamianos, E. Ammicht, and H.-K. Kuo, "Dialogue management in the Bell Labs communicator system," in *Internat. Conf. Speech Language Processing*, (Beijing, China), Oct. 2000.

[11] A. Potamianos and H.-K. Kuo, "Speech understanding using finite state transducers," in *Internat. Conf. Speech Language Processing*, (Beijing, China), Oct. 2000.

[12] M. Riley, F. Pereira, "Weighted-finite automata tools with applications to speech and language processing," Technical report, AT&T Bell Laboratories, 1995.

[13] A. Stent. "Content Planning and Generation in Continuous-Speech Spoken Dialog Systems", in *Proc. of the KI'99 workshop, "May I Speak Freely?", (Bonn, Germany), Sep. 1999.*