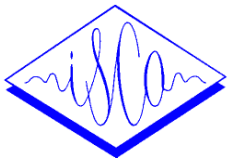


6th International Conference on Spoken Language Processing (ICSLP 2000)
Beijing, China
October 16-20, 2000

Natural Language Generation for Spoken Dialogue

Thomas Portele

Philips Research Laboratories
Weisshausstr. 2, D-52066 Aachen, Germany
Thomas.Portele@philips.com

ABSTRACT

A natural language generation module for spoken dialogue systems has been developed that performs three steps: generating multiple versions of an utterance, choosing the best version by a set of criteria, and annotating the text using structural information accumulated during the generation process. The requirements of spoken output is catered for by several design decisions.

1. MOTIVATION

In many dialogue systems output is generated using sentence templates and prerecorded phrases. This leads to very high quality output but only for very limited and fixed domains. This restriction can be overcome by using natural language generation techniques.

Language generation modules used in state–of–the–art spoken dialogue systems are a compromise between flexible rule–based systems (e.g. [2]) and template systems (e.g. [1]). A hybrid approach [3, 4, 13] maintains the possibility to hard-code output while allowing for rule-based generation of date and time expressions.

Generation for spoken output has to take into account not only supplying a textually correct result but also prosodic annotations [8, 13]. While purely statistic approaches (e.g. [9, 11]) are attractive by allowing automatic training of the generation process, they fail to deliver prosodic information due to missing information about structure.

A hybrid natural language generation module for the Philips research dialogue system architecture [1, 6] has been developed. It should provide

- separation of dialogue-specific and language-specific properties,
- modeling of certain linguistic phenomena (e.g. referencing),
- reusability of basic elements like date or time expressions,
- adaptability to different output channels (text, speech from templates, speech by synthesis from prosodically annotated text).

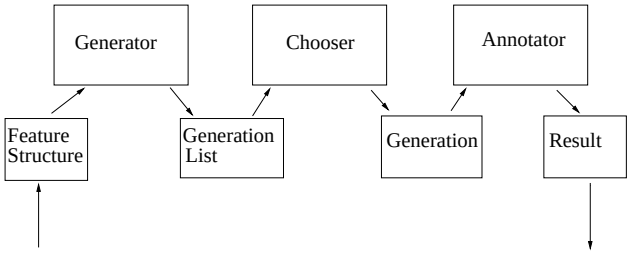


Figure 1: Architecture of the NLG system.

2. SYSTEM OVERVIEW

The system involves three steps: a generator which constructs a set of possible results, a chooser which determines the best version from that set, and an annotator which transforms the chosen version from its representation as tree of feature structures into the desired output format, using the attached information to supply adequate annotations (Figure 1).

The input of the system is a hierarchical feature structure (Figure 2 displays an example) generated by the dialogue module. Within this feature structure, a feature can appear multiple times (e.g. if a database query returned multiple results). The NLG module can omit redundant information but output planning (i.e. grouping/sorting) or dialogue strategy (i.e. constraining/relaxing) is not part of its capabilities.

The output of the NLG system can be used as input for a speech synthesizer. The annotator is able to supply a SABLE–compliant annotation [12] (Figure 3). Other implemented formats include plain text and HTML/XML.

3. GENERATOR

The generator transforms an input feature structure into a set of possible generation results. This is done by means of a tree–adjoining grammar. Generation is controlled by constraining features at the root nodes. A system state consisting of a set of variables can be changed by actions associated with a rule. Mechanisms to allow sequential processing of multiple input features and to convert input feature values into rule terminals are implemented.

```
((Utterance:"result")
 (Number:2)
 (User:(
  (town:"Aachen")
  (starttime:"840")
  (endtime:"900")))
 (Result:(
  (time:840)
  (cinema:"Atlantis")
  (town:"Aachen")
  (movie:"Casablanca")))
 (Result:(
  (time:860)
  (cinema:"Eden")
  (town:"Aachen")
  (movie:"Short Cuts"))))
```

Figure 2: Example feature structure for a movie database query result with two hits; the user had specified `starttime`, `endtime` and `town`.

```
<SABLE>
Between
<EMPH LEVEL=1.0> two p.m.  </EMPH>
<BREAK LEVEL=2.0> and
<EMPH LEVEL=1.0> three p.m. </EMPH>
<BREAK LEVEL=2.0> the following movies are playing
<BREAK LEVEL=2.0> in
<EMPH LEVEL=1.0> Aachen </EMPH>
<BREAK LEVEL=3.0> in the
<EMPH LEVEL=1.0> Atlantis </EMPH>
<BREAK LEVEL=2.0> at
<EMPH LEVEL=1.0> two p.m. </EMPH>
<BREAK LEVEL=2.0> the movie
<EMPH LEVEL=1.0> Casablanca </EMPH>
<BREAK LEVEL=3.0> and in the
<EMPH LEVEL=1.0> Eden </EMPH>
<BREAK LEVEL=2.0> at
<EMPH LEVEL=1.0> two twenty </EMPH>
<BREAK LEVEL=2.0> the movie
<EMPH LEVEL=1.0> Short Cuts. </EMPH>
</SABLE>
```

Figure 3: Example output for the input displayed in Figure 2 with SABLE annotation.

3.1. Rule Structure

Grammar rules resemble syntactic trees in the TAG formalism [5, 10, 13]. Possible operations on trees are substitution and adjoining [5]. The tree structure is later exploited to obtain prosodic annotations, but flat trees are also allowed. The labeling of the nodes is arbitrary (with one exception: the name of the start node must be "Utterance"). Thus, no linguistically motivated restrictions are imposed on the grammar writer; the grammar writer does not have to be a trained linguist.

Each node can have an associated feature structure that contains conditions for its applicability (root node, Figure 5) or information about the node's status (leaf type etc., Figure 6). If the root node feature structure can be unified with the current system state and with optional extra constraints represented as a feature structure at a nonterminal leaf, then the pertinent leaf is substituted by the tree starting at the root node. The system state is represented

```
((Utterance:"result")
 (Number:2)
 (time:720)
 (cinema:"Atlantis")
 (town:"Aachen")
 (movie:"Casablanca")))
 (Result:(
  (time:740)
  (cinema:"Eden")
  (town:"Aachen")
  (movie:"Short Cuts"))))
```

Figure 4: Input feature structure from Figure 2 after the `User` feature has been processed and the target rule for `Result` has just been applied.

by a set of system variables and input features. For example, in Figure 5 the current system state may not have a feature with attribute "hastime" and value not "true" if the rule should be applied. Thus, a flexible and dynamic derivation process is possible.

3.2. "Target" Features

As a set of features with identical attribute names can be part of the input (e.g. for the results of a database query, Figure 2) these features have to be accessed one after another. Special rules, "target" rules that have the feature `target:"true"` attached to their root node, take the first feature with an attribute name equal to the root node name and move all internal features up one feature level. The original feature is discarded. Thus, subsequent instances of features with identical attribute names can be processed sequentially (Figure 4).

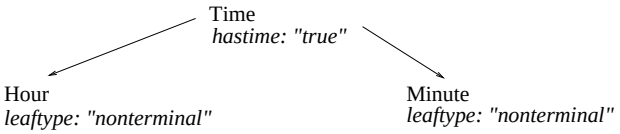
3.3. Actions

Each rule has a (possibly empty) set of actions assigned to it. An action is a simple assignment statement with an optional condition (Figure 5 and 7). The assignments are made to variables that can contain a string value or a float value. The actions can access variables from the system state and features from the current input feature structure. In actions functions can be called that transform one input value into an output value. Predefined functions can be used as well as user–defined transformations. User–defined functions are simple lists of input–output pairs (e.g. "$1 \mapsto one$, $2 \mapsto two$, ..."). Predefined functions are e.g. `wcount`, which returns the number of words in a string, or `hasfeature`, which evaluates whether the current input feature structure has a certain attribute at its top level. The actions of a rule are performed when a rule is applied. The system state, the list of variables, is changed due to these actions.

3.4. Leaves

The leaves of a rule tree can be either terminal, nonterminal or slot leaves. Nonterminal leaves demand substitution with trees that have the same node name as root node name (Figure 5). Terminal leaves are constants. Slot leaves are variables filled by accessing the input feature structure or the variables in the system state (Figure 6).

Slot leaf filling is performed *after* the pertinent rule has been ap-




```
hour = time / 60 ;
minute = time % 60 ;
```

Figure 5: Example rule with two nonterminal leaves, a root feature conditioning applicability, and a set of actions.

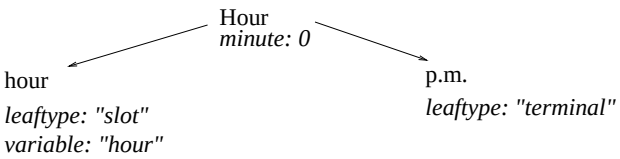
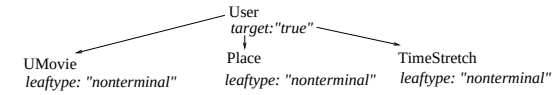


Figure 6: Example rule with a slot leaf and a terminal leaf.




```
isuser="true" ;
(hasfeature("movie")="true"&movie~"all")hasusermovie="true" ;
```

Figure 7: Example rule with nonterminal leaves specifying the values supplied by the user for a database query. The conditional action checks whether the user explicitly specified a movie and sets a system variable accordingly because the wording of the whole utterance is effected by this.

plied. This implies that rule actions may influence the values of the slots. A slot leaf is filled by default with the value of a feature that has the leaf name as attribute. If the leaf has a feature `(variable:"XXX")` (Figure 6) then the value of that variable from the system state is taken instead. If a slot leaf cannot be filled the generation is stopped. A filled slot leaf changes its type in the generated tree to `terminal` . If an input feature was used to fill the slot that feature is erased.

3.5. Generation

The generation starts with "Utterance" as the start symbol, which is also the only feature present at the highest level of the input feature structure. Every rule that is applicable in a given state of a generation process is applied and forks another instance of the generation tree. Generations are invalid if nonterminal symbols remain in the generated tree (i.e. no complete derivation was possible), if slot leaves are not filled (changed to terminal leaves), or if relevant input information (subfeatures from the feature structure in a "target" feature) is not used in the generation tree.

The generation process transforms an input feature structure into a set of possible output versions as pseudo-syntactic trees. The feature structure of suitable nodes is enhanced with information collected during the generation process (e.g. the value of slot leaves). This information is used to assess the different output versions and to supply prosodic annotations for speech synthesizers .

4. CHOOSER

The output versions are evaluated to obtain the one with the highest score. Different methods $sc_i, i = 1 \ldots n$ can be combined to score the generation results r_j: $sc(r_j) = \sum_{i=1}^{n} w_i sc_i(r_j)$. The method weights w_i can be adapted to select e.g. longer or shorter utterances depending on the user preferences. Currently, four methods are implemented:

- random choice,
- a method that favours either the longest or shortest version (measured by the number of words),
- a method that favours the version with the smallest number of words not contained in a given lexicon (e.g. the lexicon of the speech recognizer in order to use as many words as possible on the output side that the system can understand on the input side),
- a method that assesses the quality of referring expressions to entities.

For the last method entities are defined as being associated with those leaves of the output tree that used to be slot leaves (i.e. carry important information). An entity has a class which is the attribute of the pertinent slot feature (e.g. `movie`), and a value which is the slot value (e.g. `Star Wars`). Entities are referenced by parts of text. A reference can be either a pronoun ("he,she,it"), a definite description ("this movie"), or an explicit reference. A set of entities is constructed during the generation process of a dialogue. For each entity a list of references with their types and utterance indices (the number of an utterance in the current dialogue) is kept and updated which constitutes a kind of dialogue history. Work is in progress to include user input as well, but currently, only the system output is taken into account for the entity set.

The referring score for a generated result is computed by summing up positive and negative values associated with constraints, e.g.

- the current reference is a pronoun, and the entity was never mentioned before (negative)
- the current reference is a pronoun, and the last mentioned entity of that class is the current entity, and the entity was last mentioned in the same utterance (positive)
- the current reference is a definite reference, and the last mentioned entity of that class is not the current entity (negative)
- the current reference is neither a pronoun nor a definite reference, and the entity has not been mentioned before (positive)

A further scoring method envisaged is using language model scores [7] based on application–dependent large corpora.

5. ANNOTATOR

The output version with the highest score is chosen and annotated with e.g. prosodic information. Several annotation methods exist, and new ones can easily be added. Each annotation method takes

a generation tree and produces a text string. The information in the tree is employed to perform annotations, e.g. to put accent markers around references to slot entities, to put boundary markers between two words if the route in the generation tree from one word to the next one involves moving up a certain number of levels within the tree, to use phonetic transcriptions from a lexicon etc. Supported annotations are

NoAnnotation just gets the surface string,

SableAnnotation inserts tags compliant with Sable 0.2 [12], a markup language for speech synthesizers (Figure 3),

XMLAnnotation provides XML tags,

HTMLAnnotation provides HTML tags.

6. CONCLUSION

The system presented here is a hybrid system that allows for prosodic annotation derived from the structural information accumulated during derivation, but the rules do not have to adhere to any syntactic theory. The same holds for referencing analysis where non–linguistic criteria derived from the dialogue system specification are applied. Coherence between input and output (what the system understands and what it says) is ensured by using a common lexicon. Thus, performance in spoken dialogues is supported by the design of the generation system.

7. REFERENCES

1. Harald Aust, Martin Oerder, Frank Seide, and Volker Steinbiss. The Philips automatic train timetable information system. *Speech Communication*, 17:249–262, 1995.
2. Tilman Becker. Fully lexicalized head-driven syntactic generation. In *Proc. 9th International Workshop on Natural Language Generation*, pages 208–217, Niagara-on-the-Lake, 1998.
3. Tilman Becker and Stephan Busemann, editors. *"May I speak freely?" - Between templates and free choice in Natural Language Generation: What is the right NLG technology for my application?*, Bonn, 1999. Workshop at KI 99.
4. Stephan Busemann and Helmut Horacek. A flexible shallow approach to text generation. In *Proc. 9th International Workshop on Natural Language Generation*, pages 238–247, Niagara-on-the-Lake, 1998.
5. Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In *Proc. TAG+4 Workshop and Tutorials*, pages 1–56, Philadelphia, 1998.
6. Andreas Kellner, Bernd Rüber, Frank Seide, and Bach-Hiep Tran. PADIS - an automatic telephone switchboard and directory information system. *Speech Communication*, 23:95–111, 1997.
7. Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the COLING 98*, Montreal, 1998.
8. Christine H. Nakatani and Jennifer Chu-Carroll. Using dialogue representations for concept-to-speech generation. In *Proc. ANLP/NAACL Workshop on Conversational Systems*, pages 48–53, Seattle, 2000.
9. Alice H. Oh and Alexander I. Rudnicky. Stochastic language generation for spoken dialogue systems. In *Proc. ANLP/NAACL Workshop on Conversational Systems*, pages 27–32, Seattle, 2000.
10. Peter Poller and Paul Heisterkamp. A compact representation of prosodically relevant knowledge in a speech dialogue system. In Kai Alter, Wolfgang Finkler, and Hannes Pirker, editors, *Proc. ACL Workshop on Concept to Speech Generation Systems*, pages 17–22, Madrid, 1997.
11. Adwait Ratnaparkhi. Trainable methods for surface natural language generation. In *Proc. NAACL*, pages 194–201, Seattle, 2000.
12. Richard Sproat, Andrew Hunt, Mari Ostendorf, Paul Taylor, Alan Black, Kevin Lenzo, and Mike Edgington. SABLE: A standard for TTS markup. In *Proc. 3rd ESCA Workshop on Speech Synthesis*, pages 27–30, Jenolan Caves, NSW, Australia, 1998.
13. Mariët Theune, Esther Klabbers, Jan Odijk, and Jan Roelof de Pijper. From data to speech: A generic approach. Technical Report 1202, Institut for Perceptie Onderzook, Eindhoven, 1997.