

SPIN: LANGUAGE UNDERSTANDING FOR SPOKEN DIALOGUE SYSTEMS USING A PRODUCTION SYSTEM APPROACH

Ralf Engel

DFKI GmbH
66123 Saarbrücken, Germany
ralf.engel@dfki.de

ABSTRACT

This paper describes a language understanding module for spoken dialogue systems producing frame based semantic output. The presented approach adapts ideas from production systems to the task of language understanding. It interleaves in a new manner template-driven cascaded word-to-frame transformation with syntactic analysis. The advantages over conventional parsers are the flexible output structure being independent of the syntactic structure in a wide range, the ability to use different levels of syntactic analysis at the same time and better support for relatively free word order languages like German. Other important properties are robustness, the capability to process complex utterances, and the easy creation of knowledge bases. A preliminary evaluation shows promising results.

1. INTRODUCTION

This paper presents SPIN, a language understanding module for spoken dialogue systems used among others within the SmartKom project [1] and NaRaTo (a tool for querying databases using natural language). The task of this module is to transform the output of a speech recognizer (usually a word lattice) into a list of alternative semantic representations describing the user intention. The produced output consists of nested frames which can also be seen as typed feature structures. The output is interpreted by a dialogue manager where it usually triggers an action, e.g., to query a database.

Most common for the task of robust language understanding are context free grammars (CFGs) [2] and finite state transducers (FSTs) [3],[4],[5]. For our goals we found these approaches insufficient and followed a new approach basing on ideas influenced from production system (e.g., OPS5). The chosen approach provides the following advantages over conventional parsing methods:

- The result structure can look very different from the syntactic structure of the utterance.
- It allows the use of different syntactic analyses (even at the same time) without changing the semantic knowledge base.
- It is more suitable for relatively free word order languages like German.

Further aspects influencing the design are the easy creation of the knowledge bases, the ability to process complex utterances (e.g., complex database queries), and the robustness against syntactically incorrect input and recognition errors. Finally performance is an issue since dialogue systems are interactive systems.

User written templates process on a working memory (WM) that consists of a mixture of words embedded in a syntactic structure, and nested frames. Initially the WM is filled with the syntactically analyzed utterance. The templates are responsible for a cascaded transformation from words to nested frames regarding the syntactic structure. Since the templates are able to operate on nested frames, they allow greater structural changes (in contrast to CFGs).

Unique to this approach is the way in which the syntactic analysis is integrated. The templates can be written independently of the syntactic analysis. This bears two advantages: First, different levels of syntactic analysis can run concurrently using the same semantic knowledge base, e.g., a shallow one for robust processing and a deep one being able to handle more complex utterances. This allows a dynamic adaptation to unsteady speech recognizer error rates. The second advantage is that the syntactic analyses can be exchanged or developed further without or with only little modifications to the templates.

Usually more than one result is produced resulting from different paths through the word lattice, from different levels of syntactic analysis and from syntactic and semantic ambiguities. A semantics-based scoring function is used for choosing the most plausible result regarding the portion of processed words and expectations provided by the dialogue manager.

In the following section, the application of templates is presented without any syntactic analysis, in section 3 the integration of syntactic analyses is shown. Section 4 describes an preliminary evaluation.

2. TEMPLATE APPLICATION

As already mentioned, SPIN is inspired by production systems. So first the working memory is described, then the design of a single template and the application order is discussed. Finally processing of referential expressions and segmentation of user input are explained. Example templates taken from the SmartKom project are shown in figure 1.

2.1. Working memory

The working memory (WM) can consist of words, phrases, frames and set of frames (for coordination). All possible elements of the WM are called tokens. The order of the tokens is stored and can be considered during matching. Initially the WM is filled with the words of the input structure. These words are transformed step-by-step to nested frames.

<p><i>I would like to watch a movie with Kevin_Spacey that ends at 11 p.m.</i></p> <p>(1) movie → avMedium(type:movie)</p> <p><i>I would like to watch a avMedium(type:movie) with Kevin_Spacey that ends at 11 p.m.</i></p> <p>(2) [\$H=word(semCat:CARD(0,12)) p.m.] → time(hour:number+(\$H,12))</p> <p><i>I would like to watch a avMedium(type:movie) with Kevin_Spacey that ends at time(hour:23)</i></p> <p>(3) \$N=word(semCat:actor) → actor(name:\$N)</p> <p><i>I would like to watch a avMedium(type:movie) with actor(name:Kevin_Spacey) that ends at time(hour:23)</i></p> <p>(4) [\$M=avMedium() with \$A=actor()] → \$M(\$A)</p> <p><i>I would like to watch a avMedium(type:movie,actor(name:Kevin_Spacey)) that ends at time(hour:23)</i></p> <p>(5) [\$M=avMedium() that ends % at \$T=time()] → \$M(broadcast(endTime:\$T))</p> <p><i>I would like to watch a avMedium(type:movie,actor(name:Kevin_Spacey), broadcast(endTime:time(hour:23)))</i></p> <p>(6) I % would like % to watch % a \$M=avMedium() % [% on or(TV,\$C=channel())] % \$T=time() → watchTV(broadcast(\$M,\$C,beginTime:\$T))</p> <p><i>watchTV(broadcast(avMedium(type:movie,actor(name:Kevin_Spacey),broadcast(endTime:time(hour:23))))</i></p> <p>(7) \$B=broadcast(\$M=avMedium(broadcast(*\$X=any())) → \$B(\$M,\$X)</p> <p><i>watchTV(broadcast(avMedium(type:movie,actor(name:Kevin_Spacey)),endTime:time(hour:23)))</i></p>

Fig. 1. Step-by-step processing of an example utterance without syntactic analysis. (This is the translated and slightly simplified version of the original German example.) Intermediate results in the WM are written in italic. (“[...]” activates list based processing, “%” means optional.)

During processing alternative intermediate results can emerge. They are handled using more than one WM in parallel and storing the corresponding processing states in a priority queue.

2.2. Template design

A template consists of a conditional part checking if the template can be applied and a substitution part replacing the matched tokens in the input stream.

2.2.1. Matching part

The conditions test the existence or absence of words or frames. Matched input tokens can be bound to variables. The conditions

extend CFGs allowing nested tests which enable larger transformations and they are not restricted to operate on top level, but also within frames. This allows deferred reordering within a frame. An example is template (7) in figure 1.

Both set-oriented processing (the tokens only have to be included somewhere in the WM) and list-oriented processing (the tokens have to occur in the specified order in the WM) are available. Set-based processing substantially simplifies processing of relatively free word order languages like German. To reduce the amount of templates, conditions can be marked as optional and conditions can be grouped to a set of alternatives.

A template is applied on the WM as often as possible, but conditions can be marked to be applied only once, e.g., to avoid infinite loops if tokens are only added to the WM and not deleted. The matching process is not always unique, different alternatives lead to multiple WMs.

2.2.2. Substitution part

The substitution part can consist of words and frames to be newly created and variables (bound in the matching part) for moving matched tokens. These tokens replace the matched tokens in the WM. To modify frames bound to variables, tokens can be specified which are inserted into these frames. To move matched tokens into frames, variables can be mentioned within these frames.

This approach allows greater structural changes which are sometimes necessary when the syntactic structure of the utterance does not match the frame structure very well (see also figure 1). It is also very suitable for dealing with nested database queries.

2.3. Template application order

Searching through all possible application orders is of course too time consuming since many incomplete solutions with unprocessed words are produced. These solutions are not useful at all. Also common optimizations used in production systems like RETE [6] are not helpful since they are designed for slowly changing WMs, which is not the case here.

For efficient processing the templates are applied in a predefined order. The order is either specified by the template writer or is automatically determined offline. Automatic ordering is realized using a partial order relation to construct a dependency graph which is later linearized. The partial order relation sorts templates creating frames of a specific type before templates modifying frames of that type and these before templates deleting frames of that type. Also more specific templates are processed first. Additionally templates containing lists are considered, e.g., templates processing adjectives.

The dependency graph can contain cycles. It is tried to handle them automatically, e.g., by grouping the members of a cycle to a set of alternative templates. If this fails the template-writer is requested to change the affected templates.

This approach has the effect that the first solution is found very quickly, since the templates are applied in the predefined order and no backtracking is necessary for finding the first solution. For finding the next solutions backtracking is used, of course.

2.4. Input segmentation

If the user speaks more than one utterance without a pause, a problem occurs using set based processing combined with sequential bottom-up application. A template can “steal” tokens from another

```

I want to watch this one
(1) [this one]
    → refObject(subObject|)

I want to watch refObject(subObject|)
(2) watch $M=avMedium() [%on $C=channel()
    %$T=time()
    → watchTV(broadcast($M,$C,beginTime:$T))

1. I want to watchTV(broadcast(refObject(avMedium|)))
2. I want to watch refObject(subObject\avMedium|)

```

Fig. 2. Processing referential expressions. Two parallel WMs are created.

utterance due to the lack of segmentation. To overcome this problem frames can inherit from a special frame called utterance. The used matching algorithm ensures that such frames do not overlap w. r. t. the original input words.

The idea is that alternative segmentations are produced by ignoring a part of the optional template conditions that match tokens which are located at the border of the matched input tokens. Since this can lead to too many solutions, those solutions are ignored which do not lead to different maximal segmentations.

Finally it is the task of the scoring function to decide which segmentation is regarded the best. It is planned to include information from a prosody module which annotates probabilities of a sentence boundary for each word.

2.5. Reference handling

There is built-in support for referential expressions. This allows writing templates without caring about processing referential expressions. A special frame called refObject exists which can specialize its own type during template application. Initially its type is set to the union of all types that can be referenced. Each time a template is applied its type is refined depending on the conditions of the template.

Due to the sequential application of the templates, a possible alternative branch excluding this refinement has to be followed too. Otherwise other solutions resulting from different restrictions are not found. Semantic restrictions are exploited by the multimodal fusion and for resolving referential expressions in a focus stack.

An example for reference handling is shown in figure 2.

3. INTEGRATING SYNTACTIC ANALYSIS

To be robust and capable of processing more complex sentences at the same time, it is possible to run several levels of syntactic analysis concurrently. The currently used configuration uses three levels of analysis: none, shallow (only nominal and prepositional phrases) and full analysis. At the end the scoring function is responsible for selecting the best result. An example for processing using a syntactic analysis is shown in figure 3.

3.1. Offline analysis of the templates

To be able to use the same templates for all different syntactic analyses and enable the exchange of the syntactic analyses, the

templates do not contain any syntactic elements (except in some cases, markers are needed, see below).

Since the matching between the syntactically analyzed input and the “flat” user-written templates would not work, the templates are syntactically analyzed offline by the corresponding grammars. During online matching the syntactically analyzed input is matched with syntactically analyzed templates, so two syntactical structures are compared. Phrases are matched the same way as frames.

Parsing templates leads to two problems: The frames occurring in the templates must be integrated in the syntactic analysis. This is solved by assigning parts of speech to frames (either in the class hierarchy or directly in the template). The second issue is that in many cases more than one parse tree exists. In such a case the template-writer has to insert markers if another than the default parse is preferred. A scoring function compares the different parsing results regarding the markers. (E.g., if a PP should be embedded in a NP instead of being assigned to the verb then this can be expressed by a marker.)

```

which movies are on TV
    syntactic analysis: parsing NP
NP[which movies] are on TV
(1) movie
    → avMedium(type:movie)
(1*) NP[movie]
    → NP[avMedium(type:movie)]
NP[which avMedium(type:movie)] are on TV
(2) which $M=avMedium()
    → $M(focus:whQuery)
(2*) NP[which $M=avMedium()]
    → NP[$M(focus:whQuery)]
NP[avMedium(type:movie,focus:whQuery)] are on TV
    syntactic analysis: parsing NP and PP
NP[avMedium(type:movie,focus:whQuery)] are
PP[on NP[TV]]
    syntactic analysis: parsing VP
VP[NP[avMedium(type:movie,focus:whQuery)] are
PP[on NP[TV]]]
(3) $M=avMedium() are on TV %$T=time()
    → watchTV(broadcast($M,beginTime:$T))
(3*) VP[NP[$M=avMedium()] are PP[ on NP[TV]]
    %$T=time()]
    → VP[watchTV(broadcast($M,beginTime:$T))]
VP[watchTV(broadcast(avMedium(type:movie,
focus:whQuery))]

```

Fig. 3. Example using deep syntactic analysis. (This is the translated version of the original German example.) The templates marked with * are the offline syntactically analyzed templates.

3.2. Online processing

The input utterance is syntactically analyzed. Instead of processing the complete syntax tree at once, inner phrases are processed before outer phrases. This prevents unintended deletion of sub-phrases due to a wrong order of template application (e.g., a relative clause that is processed after the NP to which it belongs).

If supported by the syntactic analysis, the template application can be started directly after a phrase is recognized (as shown in figure 3). This allows to prune the search space as the phrase is scored immediately and compared to already existing results for the same part of the word lattice. Possibly, the result can be discarded.

3.3. Implemented sample grammar

In principal in SPIN any syntactic analysis can be used. We developed a syntactic analysis for German using a combination of finite state transducers [7] and syntactic templates. The grammar supports among others subordinate clauses, relative clauses and infinitive constructions. Basis for the grammar is [8].

The implemented algorithms work directly on the word lattice and insert intermediate results there to avoid multiple analyses. Special measures are implemented to maintain the anytime-property also during syntactical parsing.

Two features raise the robustness of the syntactic analysis: Agreement is handled laxly, so mismatches lead only to lower confidence values. Also the syntactic role for each NP is assigned only with a confidence value. During template application the assignment can be changed.

4. EVALUATION

The implementation of the described system in Java has been recently completed. Also, associated knowledge bases for German have been developed. It is planned to extend the system for English in the near future.

Some preliminary results are presented in table 1 based on a sample corpus for EPG (electronic programming guide for TV) and pedestrian route planning within a city. The corpus contains 136 utterances spoken by 5 different speakers. The used template set contains 293 templates with a lexicon of 2500 words. No syntactic analysis was used evaluating this corpus, since the collected utterances are too short and the error rate of the recognizer is too high to gain any advantage of using a syntactic analysis. The average processing time for one path in the word lattice is about 5 ms.

<i>time-out</i>	<i>fully correct</i>	<i>part. correct</i>	<i>only sub-frames</i>	<i>no intention</i>	<i>in-correct</i>
b. c.	52.2%	5.9%	10.3%	28.7%	2.9%
2s	67.7%	3.7%	5.1%	19.1%	4.4%
20s	68.4%	3.7%	4.4%	18.4%	5.1%
trans.	89.7%	2.9%	3.7%	2.9%	0.8%

Table 1. Evaluation results. *b. c.* means that only the best chain in the word lattice is considered, *trans.* that the transliterated utterances are used instead of the audio data. *No intention* means that the dialogue manager has to ask the user to repeat the utterance, *incorrect* that the wrong action is triggered.

The results show that using the word lattice instead of only the best hypothesis improves the number of correct interpretations substantially. A very long processing time does not improve the results any more using this corpus.

5. CONCLUSION AND FUTURE WORK

The goal was to develop a robust and powerful language understanding module for spoken dialogue systems providing a flexible output structure, different levels of syntactic analysis and support for free word order languages. Other important aspects are the easy creation of knowledge bases and short processing times. We achieved this goal by adapting ideas from production systems to natural language processing.

Future plans include the optimization of search in the word lattice. The idea is to consider first only the "more important" words and expanding only these paths that promise semantically useful results. Another issue is corpora based learning of knowledge bases.

6. ACKNOWLEDGMENTS

This work was funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the SmartKom project under Grant 01 IL 905 K7. The responsibility for the contents lies with the author.

I would like to thank Tilman Becker and Norbert Reithinger for their invaluable and helpful comments on earlier versions of this paper.

7. REFERENCES

- [1] W. Wahlster, N. Reithinger, and A. Blocher, "Smartkom: Multimodal communication with a life-like character," *Proceedings EUROSPEECH*, 2001.
- [2] J.F. Allen, B. Miller, E. Ringger, and T. Sikorski, "A robust system for natural spoken dialogue," *Proc. 34th Meeting of the Assoc. for Computational Linguistics*, 1996.
- [3] Wayne Ward, "Understanding spontaneous speech: the phoenix system," *Proceedings of ICASSP 91*, 1991.
- [4] Stefanie Seneff, "Tina: A natural language system for spoken language application," *Computational Linguistics*, vol. 18, no. 1, 1992.
- [5] E. Kaiser, M. Johnston, and P. Heeman, "Predictive, robust finite-state parsing for spoken language," *Proceedings of ICASSP 99*, 1999.
- [6] C. L. Forgy, "Rete: A fast algorithm for the many pattern / many object pattern match problem," *Artificial Intelligence*, vol. 19, pp. 17-37, 1982.
- [7] W.A. Woods, "Cascaded ATN grammars," *American Journal of Computational Linguistics*, vol. 6, no. 1, 1980.
- [8] Stefan Müller, *Deutsche Syntax deklarativ*, Max Niemeyer Verlag, Tübingen, Germany, 1999.