A Transformational Approach to Natural Language Understanding in Dialogue Systems

D. H. Lie, J. Hulstijn, R. op den Akker and A. Nijholt
Centre for Telematics and Information Technology, University of Twente,
PO BOX 217, 7500 AE Enschede,
The Netherlands,
anijholt@cs.utwente.nl

Abstract

Natural language understanding in dialogue systems is taken to consist of two subsequent processes: *rewrite* and *understand*. The rewriting proces is a transformational proces in which natural language utterances are mapped via a sequence of context-sensitive string to string transformations onto some semantic normal form. In the understand proces an interpretation of the semantic form is made, with respect to a particular dialogue state. The rewrite proces is completely independent of the state of the dialogue. The transformational approach presented is compared with more traditional grammatical and parsing approaches. The approach is used in a first prototype of a natural language interface for a theatre information and booking system.

Keywords: Dialogue Systems, Robust Parsing, Software Engineering

1 Introduction

One of the problems one has to solve in making a natural language dialogue system is how to obtain from the user utterances the relevant information needed for the best reaction by the system. This is the problem we consider and we report on a corpus based solution that avoids the making of a grammar that covers the user language. The solution consist in a "rewrite and understand" approach to natural language. User input is not parsed with respect to a grammer but transformed using a set of rewrite rules into some "normal form" which is then interpreted in the context of the dialogue as an action performed by the system; a reaction on the users input. The approach is implemented in several natural language processing systems; one of these is a dialogue system that provides information about and allows booking of theatre performances. The particular rewrite rules for this system are based on analysis of a corpus of dialogues collected partly from Wizard of Oz experiments and partly from experiments with a preliminary full-automated version of the dialogue system. The communication language of the system is Dutch.

The expected practical value of the method is based on the following idea. When a user uses some expression in a particular dialogue situation with some particular intention there are many other expressions that could be used by this user in this situation for expressing the same intention. Hence, why shouldn't we try to find rules for transcribing these possible expressions into some normal form, which is then interpreted as a call of a dialogue procedure.

The resulting robust system should deal with complex polite expressions, and be able to parse almost any string containing meaningful information for the task domain. The dialogue state, containing relevant information about the dialogue sofar, can be used to fill in the missing parameters.

We model the proces of 'understanding' a natural language expression as two subsequent processes. The first part transforms the expression into some intermediate semantic form, the second part is an interpretation of this form given the state of the dialogue. So, the systems reaction on a user utterance in principle depends on the state of the dialogue, but the way in which the intermediate semantic form is constructed does not depend on this state.

Within this model of language understanding a natural language expression is seen as a manner (be it speech or typed or written) of wrapping a formal semantic form. For instance, the expression

"please show me the dance performances of next week" is a natural way of uttering a semantic form like [ACT: show ; ARGUMENT: performances ; GENRE: dance ; DATE: next week]. Another way of uttering the same semantic form is: "show dance performances of next week".

The approach assumes a distinction between a *semanti*c meaning of a sentence being something that is independent of the utterence situation and *pragmatic* meaning as something that is dependent on the dialogue situation. The rewriting rules should rewrite user input in such a way that no relevant pragmatic information is lost. It will be clear that this approach inherits its practical value in a great part from the intuiton about the pragmatics of the communication language as it is used for a particular application, i.e. by users in dialogues about a particular domain. This intuition is an intuition about two sentences having "the same prag-

matical meaning" whatever the dialogue context in which these sentences are uttered may be.

In this paper we describe the dialogue state independent part of the natural language understanding proces. In section 2 we describe how the rewriting system is specified. In section 3 an example is given of the rewriting process in the context of the theatre information system and in section 4 it is shown how it is embedded in a dialogue system. Finally in section 5 we compare this approach with more traditional (grammatical) approaches to natural language understanding; we discuss some shortcomings of the current system and ways to handle them.

Examples are in Dutch; they are client sentences and rewriting rules that are used in the theatre information and booking system SCHISMA (Hoeven et al [3], Hulstijn et al [4], Burgt et al [1]).

The complete system for the first state independent proces consists of the following parts (Figure 1).

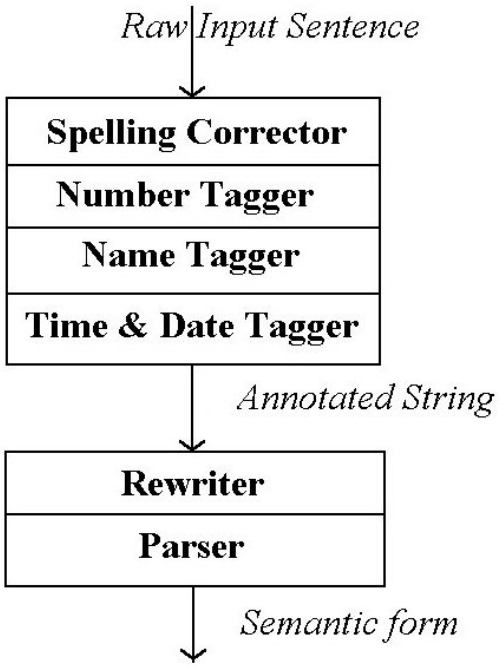

Figure 1: Transformation steps

In this paper we mainly concentrate on the Rewriter and the Parser. The word *parser* will also be used for the whole proces of rewriting and parsing.

The semantic forms output by the parser are feature-value structures, in which features represent particular types of semantic or pragmatic information and values can be either feature-value structures or atomic values. In the current system semantic forms are flat lists of feature value pairs: the first element of the list is interpreted as a dialogue function; the other elements are interpreted as argument values of this function. When a value of an obligatory arguments is not known the dialogue system asks the user for the required information. Semantic forms may contain values that are (typed) indices referring to topics of the dialogue. The reference problem is solved by the dialogue manager by searching for appropriate elementsin a context stack. We concentrate here on the rewriting system, which is independent of the type of semantic forms chosen.

2 Rewriting Rules

There are four kinds of rewrite rules; they are used for:

1. Deletion of semantically irrelevant words
2. Substitution of words by standard synonymous words or expressions
3. Transformation of complex (like polite) forms into simple (imperative) normal forms
4. Changing the order of phrases into a predefined normal order[1]

Each application of a rewriting rule results in a semantically equivalent sequence. An input sentence is submitted to a transformation system that applies the rules in the order in which they are specified. A sequence of words S_0 is transformed into S_1 using rule r_1; then S_1 is transformed into S_2 using rule r_2 and so on until no rule can be applied anymore, finally resulting in a sequence S_n which is the semantic form of S_0.

$$S_0 \xrightarrow{r_1} S_1 \xrightarrow{r_2} S_2 \rightarrow \ldots \rightarrow S_{n-1} \xrightarrow{r_n} S_n$$

Given a particular order of the rules, S_0 will always result in the same semantic form S_n. Since the rules are used in the order in which they are specified and at most once in the transformation proces, there is no possibility that the proces becomes cyclic.

In some applications (for some particular domains) two words or constructs may be considered as synonym, where they should be considered semantically different in other domains. But there are words, phrases or patterns that are semantically equivalent no matter what the particular domain is. Therefore rules are categorized into a set of *global* rules which are domain independent and sets of *local* rules which are domain dependent and which are associated with particular semantic forms and pragmatic dialogue functions, like: the user asks for information about a particular performance, she asks for the price of a ticket or the date of a performance.

[1] Notice that in Dutch there is for instance no "place before time" constraint for adverbial phrases.

A specification for the transformational system consists of the following parts.

GlobalStrip: in which it is specified how the input sentence is prepared for further processing. It contains a list of tokens that should be removed, like punctuation marks and special tags.

GlobalRemove: in which it is specifed which words should be deleted from the sentence.

GlobalAliases: specifies synonyms of words

GlobalRules: this parts contains the global rewrite rules

GlobalMacros: in which global macros are defined

LocalGrammar: in which local grammars are specified for particular language constructs.

Global rules obey the following syntax-rules:

GlobalRule	::= Pattern → Result
Pattern	::= { Expression }
Expression	::= Imp | Opt | Choice | Var | Word | Joker | Wildcard
Imp	::= (Expression)
Opt	::= [Expression]
Choice	::= { Expression | }
Var	::= Varname = (Exp)
Varname	::= "<" Word ">"
Joker	::= "*"
Wildcard	::= "**"
Result	::= { ResultExp }
ResultExp	::= Word | Varname

Notation: { Expression } means a sequence of one or more Expressions. [Expression] means that the Expression is optional: it matches zero or one occurrence of the pattern Expression. { Exp1 | Exp 2 } matches with patterns that match either with Exp1 or with Exp2. Round brackets are used to indicate units in compound patterns: For example the pattern: _date_ { (en _date_) | (of _date_) } matches "3 maart en 4 maart"as well as "3 maart of 4 maart". The joker * matches any word where the wildcard ** matches any sequence of words.

A rewrite rule pattern → result is applicable to any sequence of characters having a subsequence that matches the pattern in the lefthand side of the rule. The result of applying the rule is the original sequence in which the string matching the pattern has been replaced by the sequence on the righthand side of the rule.

Macros

Macros can be used in patterns in any rewrite rule. They allow for a more compact specification of the grammar rules.

Macronames are words enclosed by underscores: like _date_. A Macrovalue is a regular expression that may contain other macronames. However, recursive macro definitions are not allowed.

Example:

The following macrodefinitions define patterns for (de voorstelling <voorst> van <naam>).

det = (de | het | een)

voorst= _det_ voorstelling %play <voorst> play% [van %name <naam> name%]

In the pattern: %play <voorst> play% the tags %play and play% are used to indicate that the pattern in between (a variable name) matches a special word or phrase. In this case a phrase indicating the name or title of some performance. These tags are introduced by a tagger proces, that is run before the transformation proces.

Special features

To indicate that a word or sequence of words may only be replaced by another word or sequence of words when it occurs at the beginning (or end) of sentence, a rule contains the | symbol (bar). For instance the following rule allows the final transformation of: "doe me vier kaartjes" into "geef vier kaartjes".

| doe me → geef

Local Grammars

Local Grammars contain rewriting rules and (locally valid) macro definitions just like the Global Grammar. The local rules of a Local Grammar are used to finally transform the result of earlier rewriting steps (in which only global rules were applied) into a final semantic form. Local grammars are associated with particular categories of semantic forms. These categories of semantic forms are associated with particular interpretations (pragmatic functions or communicative acts). They are domain and application dependent. Together with the global rules they form the complete rewriting grammar for the transformation system.

Each Local Grammar has a permission set of key-words or key-phrases. The rules of a Local Grammar may only be applied to rewrite a sequence if the sequence contains one or more keywords in its permission set.

Also key-words or phrases in permission sets can be marked by a | (bar) to indicate that the word should occur at the beginning or end of the sequence. Permission sets of different Local Grammars are not necessarily disjoint. The parser chooses a permitted Local Grammar and tries to construct a semantic form by applying rules from this grammar. If it doesn't succeed it will try to apply another permitted Local Grammar. Each Local Grammar has a set of forbidden

words. If one of these words occurs in a sequences it is not allowed to apply rules of the Local Grammar. This feature speeds up the parser in finding permitted Local Grammars.

Scoring the resulting semantic form

The parser for the Local Grammars gives a score to the output semantic form it has constructed. Therefore a penalty is accociated with each local rule; a number indicating how good the input fits the result. This final parser is robust: it may skip words in its input in order to match it with the pattern in the lefthand side of the rule it tries to apply. The more words are deleted for this purpose the higher the penalty given to the resulting semantic form. If the parser has found a semantic form with a zero penalty it will stop searching for other applicable rules or Local Grammars. Otherwise it will return the result with the best score. To indicate that the parser in applying a final rewrite rule may not disgard a word or words in between two words the underscore is used in patterns. For instance, the rule voor_ datum → . . . is only applicable if the word 'voor' is followed by the word 'datum' in the sequence being rewritten.

Developing the grammar

An initial set of rewriting rules was based on the analysis of client utterances in a corpus of dialogues. These dialogues were obtained by logging Wizard of Oz experiments with an earlier version of the theatre information system.The grammars are further tuned and developed on the basis of experiments with the current completely automized system. In the theatre information system there are for example sentences that ask for the date of some performance, while another category of sentences ask for information about the contents of some performance, and another one for asking whether there are still seats free for a particular performance. For this particular information system there are 20 Local Grammars; one for each of the 20 categories of semantic forms. These categories are the result of analysing a corpus of dialogues.The Global Grammar consists of about 200 rules; the 20 Local Grammars together contain 400 rules. A specification of a rewriting system consists of several files. Parts of specifications can be taken together by means of include directives.

3 Example of the Transformation Process

We illustrate the whole transformation proces by means of the following example sentence:

input:

Kunt u mij alstublieft vertellen wanneer er volgende week een voorstelling van Karin Bloemen is?

(could you please tell me when there is a performance of Karin Bloemen next week?)

1. remove endmarks, capitals:

"?" → .

result:

kunt u mij alstublieft vertellen wanneer er volgende week een voorstelling van karin bloemen is

2. tag names, dates and performance names:

<date>=_date_macro_ → %date <date> date%.

result:

kunt u mij alstublieft vertellen wanneer er %date volgende week date% %play een voorstelling play% van %name Karin Bloemen name% is

3. remove irrelevant words:

alstublieft → .

result:

kunt u mij vertellen wanneer er %date volgende week date% %play een voorstelling play% van %name Karin Bloemen name% is

4. rewrite aliases:

vertellen → zeggen.

result:

kunt u mij zeggen wanneer er %date volgende week date% %play een voorstelling play% van %name Karin Bloemen name% is

5. rewrite:

mij → me.

je me → je.

kunt u → kan je.

kan je zeggen → zeg.

zeg wanneer → wanneer.

wanneer ** is → wanneer is **.

result:

wanneer is er %date volgende week date% %play een voorstelling play% van %name Karin Bloemen name%

6. shuffle sentence parts:

<play>=(%play ** play%) ** → ** <play>.

<name>=([van] %name ** name%) ** → ** <name>.

<date>=(%date ** date%) ** → ** <date>.

result:

wanneer is er %play een voorstelling play% van %name Karin Bloemen name% %date volgende week%

7. apply local grammar for sentences that ask for dates:

er → .

wanneer is → wanneer speelt.

result:

wanneer speelt %play een voorstelling play% van %name Karin Bloemen name% %date volgende week %date

8. final rewrite:

wanneer speelt <play>=_play_ [[van] <name>=_name_ [<date>=_date_]]

→ (PROC: datum, PLAY: <play>, NAME: <name>, DATE: <date>)

final resulting semantic form:

[PROC: datum, PLAY: een voorstelling, NAME: Karin Bloemen, DATE: volgende week]

Note: (1) In Dutch “kan je zeggen” and “kan je me zeggen" are more polite forms for asking information than “zeg me”. (2) “Zijn er volgende week voorstellingen?" (“Are there any performances next week?”) en “zijn er voorstellingen volgende week" are equally good Dutch with no distinction in meaning. (3) In step 7 above the local grammar for date questions is applied because of the occurrence of the words ‘wanneer’ (‘when’) and ‘is’ in its permission set of key-words.

4 Schisma

In the SCHISMA project the objective is to develop a natural language dialogue sustem for theatre information and reservations. Users may inquire about performances or have tickets reserved. The main problems studied in this project are:

- How to model dialogues?
- How to model and proces the natural language input of users?
- How to model the context of the dialogue?

Starting with a global functional specification of the system we first developed a Wizard of Oz environment allowing people to engage in a dialogue with a database through a human Wizard. The Wizard responded by means of a restricted set of canned sentences in which slots for database information could be filled. These first experiments resulted in a corpus of about 80 dialogues containing about thousand client sentences. This corpus was analysed with respect to the kinds of requests users have and the way they express them in language.

Next we started to develop a context-free unification grammar that should cover most of the client utterences. This showed to be a very difficult task. When the grammar had a few hundred of rules it became unmanagable. Moreover the price for keeping the grammar and head-corner chart parser robust was that it became too ambiguous.

Therefore we decided to follow the rewrite approach. It turned out to be much easier to make the step from the corpus of client sentences to a string-to-string rewriting system than to a full-blown grammar. The first rewriting system for the SCHISMA application was developed by adding about 20 Local Grammars to an original Global Grammar developed by Lie. It can handle 80 percent of the client utterances. Most of the utterances that could not be processed have functions that do not fit the task domain of the system (for instance questions about public transport to the theatre). They do not contain keywords that trigger the parser to apply one of the distinguished Local Grammars. Some difficulties arose with constructs involving denials, compounds, or ellipsis. These can sometimes be handled by adding new rules to the rewriting system only, but sometimes they also require extension of the understanding back-end part of the system. For example, coordinated sentence forms like “Ik wil ** of **” (Eng. I want either ** or **) can transformed into “Ik wil ** or ik wil **” (Eng. I want ** or I want **). The ellipsis in “Ik wil 3 kaartjes voor donderdag en twee voor maandag” (Eng. I want three tickets for thursday and two for monday) is not so easy to resolve by the rewriting system. It can only be solved by adding a special final rewriting rule for the back-end parser: reserveer _cards_ _date_ en _cards_ _date_}.

A problem is how to handle unknown words. When input contains a proper name (for example) that is not in the name list (author or city names) the system skips it before rewriting and the word is not recognized as probably being a name. A more structural input analysis seems to be necessary, or information about the previous dialogue act should be used in the rewriting proces. The proper treatment of unknown words (or typo’s) is not a typical problem for the transformational approach, however.

5 Comparison with Traditional Approaches

The transformational approach presented in this paper is an unorthodox software engineering approach. It has some advantages compared with approaches in which a context-free based unification grammar is used. Given a corpus of sentences it is much easier to set up a rewriting system than to develop a context-free

unification grammar for the same language domain. Large context-free unification grammars that are robust enough tend to be very ambiguous and parsing becomes very time consuming. A probabilistic unification grammar (using a probabilistic left-corner parser) has been generated from the dialogue corpus (Doest [2]). This will be implemented in the system, and we can compare the performance of the rewrite approach with the grammar approach.

Replacement of words or phrases by semantically equivalent ones, reordering of phrases, and deletion can all be specified in the same style. The rewriting system is rather easy to maintain and can easily be adopted by adding or changing rules without the system becoming unpredictable.

The large variety in ways people express their questions and answers causes a serious problem for practical natural language understanding systems. It amounts to the problem of finding "the appropriate level of granularity of language processing for the task at hand" (Martin & Kehler [5]). A simple keyword analysis has shown to be too coarse-grained for most applications (Sijtsma & Zweekhorst [6]), whereas full syntactic analysis is far too complex and most likely impossible. In SWIFTUS (Martin & Kehler [5]), the 'appropriate level of granularity' is obtained by a multi-level finite state automaton. If an input sequence is not accepted by an automaton it is submitted to a higher level automaton for a more coarse grained analysis.

The application of the rewrite approach in a natural language dialogue system shows the following advantages when compared to more traditional grammar based approaches.

1. A grammar for a simplified standard language is much easier to develop and more efficient to implement.
2. The transformation from standard natural expressions onto a semantic form is far more easier to specify.
3. Some syntactic constructs occurring in natural languages are easier to handle with a context-sensitive rewriting system than with a context-free parser.
4. It is easier to add rules for treating 'ungrammatical sentences' or idiomatic phrases.
5. The context-sensitive rewriting system is easier to adapt for a special domain.

An intelligent dialogue system should know how certain it can be of the information it has obtained from the user input. The penalties that result from applications of rewrite rules give some information about the the validity of the internal semantic form given the user utterance. Current research focusses on the proper treatment of several kinds of uncertainties that arise in dialogue.

Bibliography

[1] Burgt, S.P. van de, T. Andernach, H. Kloosterman, R. Bos and A. Nijholt. Building dialogue systems that sell. Proceedings *NLP and Industrial Applications*, Moncton, New Brunswick, 1996, 41-46.

[2] Doest, H. ter. A corpus-based probabilistic unification grammar. Workshop on *Automated Acquisition of Syntax and Parsing (ESSLLI98)*.

[3] Hoeven, G. F. van der, et al. Schisma a natural language accessible theatre information and booking system. In: First International Workshop on *Applications of Natural Language to Data Bases*, Versailles, France.

[4] Hulstijn, J., R. Steetskamp, H. ter Doest, S. van de Burgt and A. Nijholt. Topics in SCHISMA dialogues. In: Luperfoy, Nijholt, and Veldhuijzen van Zanten, editors, *Dialogue Management in Natural Language Systems*, TWLT11. University of Twente, 1996.

[5] Martin, P. and A. Kehler. Speechacts: a testbed for continuous speech applications. In: Proceedings of the *AAAI-94 workshop on the integration of natural language and speech-processing*, Seattle, Washington, 1994.

[6] Sijtsma W. and O. Zweekhorst. Comparison and review of commercial natural language interfaces. In: F.M.G. de Jong and A. Nijholt, editors, *Natural Language Interfaces*, TWLT5. University of Twente, 1993.