

# Natural Language Engineering: Slot-Filling in the YPA

NICK WEBB, ANNE DE ROECK, UDO KRUSCHWITZ,  
PAUL SCOTT, SAM STEEL, RAY TURNER

Department of Computer Science  
University of Essex  
Wivenhoe Park  
Colchester  
Essex, UK  
CO4 3SQ  
+44 (0)1206 2679  
webbnw@essex.ac.uk

August 16, 1999

## Abstract

The YPA project (De Roeck et al., 1998) is building a system to make the information in classified directories more accessible. BT's *Yellow Pages*<sup>1</sup> provides an example of a classified database with which this work would be useful. Accessibility in this context means allowing users (or call center operators) to query the *Yellow Pages* system using Natural Language queries. For this to be possible there must be some method in the YPA for converting these queries into some form which allows the database to be queried. The chosen form for this project is a slot and filler construction, and this paper describes the process of slot-filling employed by the Natural Language Frontend of the YPA.

## 1 Introduction

The YPA is a language engineering project for building an intelligent directory enquiry assistant where various techniques and resources are combined in a realistic application.

To explain this, a brief overview and context of the YPA is given in section 2. In section 3 this paper then describes in more detail the *Frontend* of the YPA, the section that accepts the users input (whether from a keyboard or in some form over a network), parses it and converts it to a form from which it will be easy to construct the query that will go to the *Backend*. This form is a slot and filler structure. The slots are the roles (such as transaction or location) that are expected in a query in the given domain. The fillers are identified by keywords as belonging to a particular slot. The *Backend* contains the preprocessed data on which the answer will be based. How this *Backend* is constructed is explained in detail in (De Roeck et al., 1998).

It is our belief that a method of filling slot and filler constructions with phrasal structures and not just keywords allows us to more accurately manipulate this information during query construction, and return more precise answers to the user. It is the method of filling the slots with complex fillers that is the focus of this paper.

Take the query *kitchen cupboard specialist*. A system based on keywords would first try to match all of these, then should it fail will match combinations of the words, but perhaps not in a coherent way. By retaining the input structure, and assuming as we do that the last noun is the head of a compound noun phrase, we can present some logical choices to the user during the dialogue stage. In this case, the user will be presented with:

- kitchen specialist;

---

<sup>1</sup>Yellow Pages® and Talking Pages® are registered trade marks of British Telecommunications plc in the United Kingdom

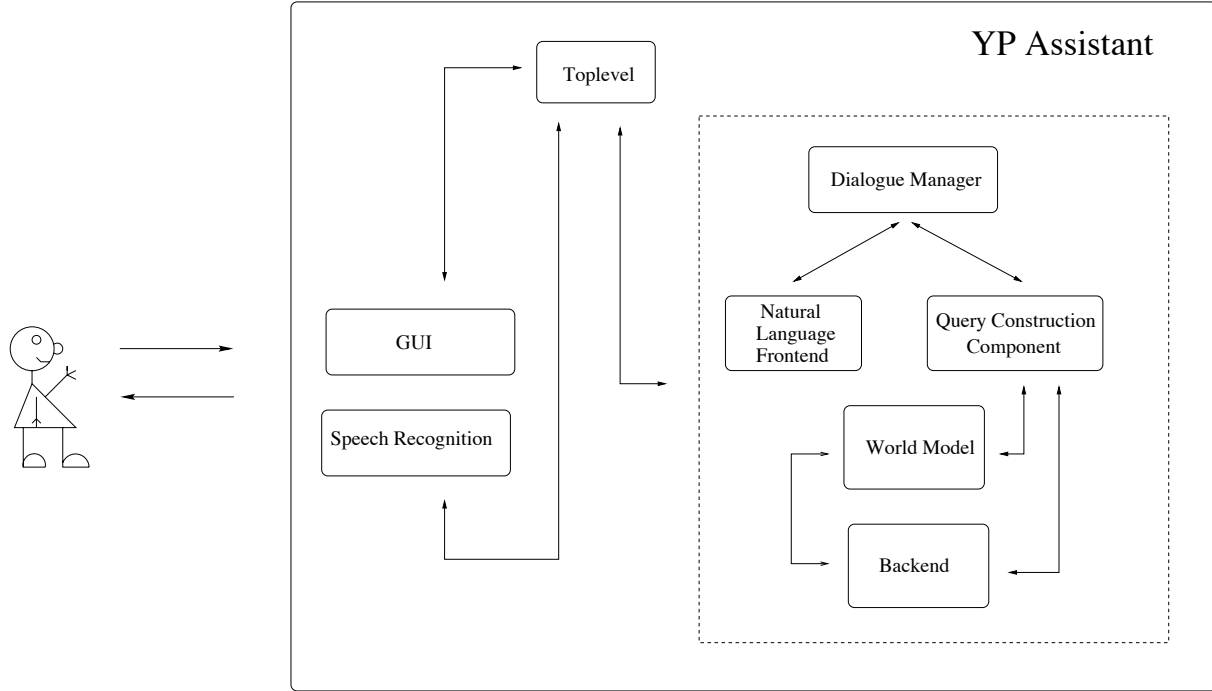


Figure 1: Architecture of the YPA

- cupboard specialist.

However, if presented with the query *take away mongolian restaurant* the system will NOT offer a *take restaurant* and an *away restaurant*, because we know that the compound *take away* behaves in a special way. There are further benefits of keeping structure for query construction. For the query *i want a helmet for my motorbike* the filler of the goods slot will have the structure:

```
goods([helmet],pp[motorbike])
```

which should lead us to some very good answers, by directly applying this structure to the *Backend* database. However, take the query *i want a helmet for my brother*. The query construction component, on finding no addresses that match all the query terms, will be able to drop the modifier *brother* in this case because it knows that prepositional attachments are less important than the main concept. The search is refined to *helmet*, which again should lead us to a useful address set.

Section 4 outlines some preliminary evaluation results of the YPA, and is followed by our plans for future work (section 5) and our conclusions in section 6.

## 2 The YPA: An Intelligent Directory Enquiry Assistant

### 2.1 Overview

The YPA is an interactive system. Figure 1 is an overview of the system architecture (depicting the data flow). A more detailed description of the operation of the YPA and its constituent modules can be found in (De Roeck et al., 1998).

A conversation cycle with the YPA can be roughly described as follows. A user utterance (recognized by the *Speech Recognition* or typed in via the *Graphical User Interface*) is sent to the *Dialogue Manager*. The *Dialogue Manager* keeps track of the current stage in the dialogue and controls the use of several submodules. Before handing back control (together with the relevant data) to the *Toplevel*, the input is first sent to the *Parser* which returns a so-called *slot-and-filler query*. The *Dialogue Manager* then consults the *Query Construction Component*, passing it the result of the parsing process (possibly modified depending on the *Dialogue History*

etc). The purpose of the *Query Construction Component* is to transform the input into a *database query* (making use of the *Backend* and possibly the *World Model*), to query the *Backend* and to return the retrieved addresses (and some database information) to the *Dialogue Manager*. Finally the *Dialogue Manager* hands back control to the *Toplevel* which for example displays the retrieved addresses. It could also put questions to the user which were passed to it by the *Dialogue Manager*, if the database access was not successful (i.e. did not result in a set of addresses).

At this stage the cycle starts again.

## 3 Engineering the YPA

### 3.1 Motivation

In order to have a base for evaluation we have to define the aims of the YPA. The final system should offer:

- the functionality of the *Talking Pages* (where a user can call to find out addresses listed in the *Yellow Pages*);
- the coverage of the whole *Yellow Pages* data;
- a user friendly dialogue.

We can assume that a user knows what the *Yellow Pages* are and that we are dealing with a cooperative user who is truly interested in a set of addresses.

It was a primary goal of the *Frontend* software that it be robust enough to handle the kind of mal-formed input expected from users, return a complete slot and filler construction and to do it fast enough to make a natural language front end a practical solution.

The development of these kinds of natural language interfaces is well documented (Hendrix et al., 1978; Grosz et al., 1987; Alshawi, 1992).

### 3.2 Frontend Techniques

#### 3.2.1 Brutal Parsing

Parsing methods for handling real NLP applications are being developed all the time. Notably, there are methods such as the LR parsing algorithm (Tomita, 1985), phrasal parsing as seen in the SPARKLE project (Briscoe et al., 1997) and chunking (Abney, 1991).

It is a primary aim of these systems that they return some linguistically correct parse tree for the input, and have the ability to choose correctly between syntactically (and semantically) ambiguous analyses. For example, the different attachments of prepositional phrases.

What was required by the YP Domain was not the linguistically correct syntactic tree structure, but the identification of those parts of the input which would allow us access to the meaning of the query, and that this process be a fast one. To that end, we began developing our own parsing process.

For our purposes, it made little difference in the construction of a query if input such as *green lawnmower* was seen as an adjective/noun construction or a compound of two nouns. Both terms would be used in the search regardless.

The requirement to handle ill-formed input meant that there was little need for a fully featured grammar. Instead a simple DCG-like grammar, without any feature requirements was adopted, easy both to understand and modify, and a basic bottom-up chart parser was implemented (Gazdar and Mellish, 1989).

What we wanted to do was rapidly assign a skeleton to any input based on the closed class words. This would provide us with a frame, and any other words in the input would be given a limited freedom to try out different classes until one fitting a parse could be generated. As soon as one parse tree exists, it is accepted. We make no claim about the resolution of ambiguity on the basis of syntactic structure, but rather let the domain database resolve such ambiguities as it sees fit.



Furthermore, domain dependent knowledge was used to remove stop words from our slots - words which we had identified through the construction of the back end database as bearing little information content with relation to the domain, such as *address* or *phone number*.

This gives us a final output structure for our example of:

- Domain Dependent Pragmatic Analysis:

```
[[transaction, [[]],  
 [goods, [[n(plumber)], pp([n(boiler)])]],  
 [payment, [[n(visa)]]],  
 [opening, [[]]],  
 [street, [[]]],  
 [location, [[n(ipswich)]]]]]
```

Initially, we had three slots for the YP domain:

- *Transaction*
- *Goods*
- *Location*

As the system developed, more slots were identified from the information contained within individual adverts within the yellow pages data set, such as:

- *Payment Methods*
- *Street Name*
- *Opening Hours*

The attribute information for these slots was found almost exclusively within the modifiers fragment created by the previous process. Each of our slots had a series of predicates associated with it, whose function was to scan the modifier fragment looking for terms which are identified with specific slots. Once found, that information (atomic or phrasal) is removed from the modifier and placed in its relevant slot.

Fortunately, because the slot-filler process had been re-written in a more generic manner, addition of new slots, and the process of filling them became an easier task. The developer has only to write a new set of predicates which enable the new slot to identify its fillers from the input, and introduce the new slot in the final mapping process.

Any information left in the modifiers slot after this series of processes was deemed to be some other sort of modifier to the goods, and was added as a modifier in the goods slot.

E.g.

```
I want a lawnmower with a hover action  
⇒  
goods([lawnmower], pp[hover,action])
```

Finally, the complete set of slots was passed to the *Dialogue Manager*.

### 3.3 Dialogue Management

The *Dialogue Manager* plays an important role by controlling the access to the various modules. The overall architecture of the *Dialogue Manager* is very similar to the *PURE* system (see (Agarwal, 1997)).

The user input is passed to the *Dialogue Manager*, which calls the Natural Language Frontend and, depending on the result, decides whether for instance the database should be accessed (by calling the *Query Construction Component*) or whether the dialogue should be restarted.

Our *Dialogue Manager* consists of a *Core Dialogue Manager* which is the domain independent heart of the system and an extension (also domain independent) which adds the basic functionality of a dialogue manager to a frame based approach and which is called the *Default Dialogue Manager*. The administrator has to (1) set interfaces to the *Core Dialogue Manager* and the *Default Dialogue Manager* and (2) customize the system for the specific application.

The general idea about a domain independent basic dialogue manager is to have a core dialogue manager that covers all tasks to be handled by any similar dialogue system without having to access the *database system*. It should detect

- that a user wants to *quit* (or *restart*) the dialogue;
- *meta queries* (where the user asks for some help etc);
- that a user uttered some *correction*.

The *Default Dialogue Manager* is this core engine expanded by adding coverage of the other states that can occur in a general spoken dialogue system:

- *mandatory slots* (which must be filled in order to submit a query to the database system) are not filled;
- *unknown concepts* occurred in the input;
- some *inconsistency* occurred;
- a database access was *successful*;
- a database access results in *too many matches*;
- a database access results in *too few matches*.

This outline compares to the two-layered dialogue architecture in (Agarwal, 1997), where the *Default Dialogue Manager* covers the upper layer of dialogue states, and where customization may refine those and add a second (domain dependent) layer. Differences, however, are the set of dialogue states and the distinction made between the various possible states.

In order to keep the dialogue manager truly parametric we do not assume anything about the structure of the query except that it is some sort of *slot and filler* list. In the YPA, we customized its current *Dialogue Manager* by defining the appropriate interfaces in the setup files.

### 3.4 Query Construction Component

This component is the most important to the quality of the results of a user's query. The structure passed to the *Query Construction Component* is a *slot-and-filler* query. The task is to match this query to a set of addresses by consulting different sources of knowledge, namely the transformed *Yellow Pages* data (part of the *Backend*) and knowledge sources which can be summarized as the *World Model*. While the *Backend* supplies indices as well as ranking values, the shallow *World Model* delivers information which can be employed on the *Backend* (e.g. for query extension). It is therefore the task of the *Query Construction Component* to evaluate the various information sources (e.g. indices *versus* ranking values) and retrieve a set of addresses from the *Backend* if possible.

The constructed query is sent to the address database. If this results in a set of addresses (up to a maximum number defined by the administrator in the setup), then the *Query Construction* is finished. If too many addresses are retrieved, then the query will be further constrained. A query that resulted in no matching addresses at all could be relaxed (if possible). This involves exploiting the *World Model*, e.g. checking for synonyms or cross-references in the *Yellow Pages* heading structure.

It is our claim that using complex fillers in our slot construction allows us greater flexibility during these stages. A standard keyword search attaches equal importance to those terms in the query. For a 'one-shot' system, where the input terms directly match some index in the database, this is not necessarily important. However, when it becomes necessary to manipulate the input query in order to return coherent data to the user, the structure of the input can prove very useful.

Take for example the query *I want a plumber with an emergency service in Wivenhoe*. Using the methods described, this would give us a slot structure of:

```
[[transaction, [[]]],
 [goods, [[n(plumber)], pp([n(emergency, service)])]],
 [payment, [[]]],
 [opening, [[]]],
 [street, [[]]],
 [location, [[n(wivenhoe)]]]]
```

We can demonstrate a number of ways that our slot structure is beneficial. Hopefully, we will find a direct match for this query within our database, however if we cannot, we have a number of options.

Firstly, we could extend the location slot, knowing that Wivenhoe is close to Colchester, or even part of the county of Essex, and retrieve any addresses matching our other fillers from this wider area.

Alternatively, we may find a list of plumbers in Wivenhoe, but none offering an emergency service. With 'emergency service' being located in a prepositional phrase, we can choose to ignore this information and return a list of plumbers. A simple keyword system might return not only plumbers in Wivenhoe, but emergency services in Wivenhoe, not useful or related to the original query at all.

At this stage the best option is to use the dialogue management of the system, and ask the user which of the two possible paths they want to choose - allowing them to specify the important part of their query, the location or the offer of an emergency service.

### 3.5 Related Work

Most natural language dialogue systems are concerned with the problem of schedules or time (among others: (Aust et al., 1995), (Wahlster, 1993), (Sikorski and Allen, 1996), (McGlashan et al., 1992; Heisterkamp, McGlashan, and Youd, 1992)), something that does not apply for the YPA. More closely related to the YPA is the *Voyager* dialogue system (Zue, 1994; Glass et al., 1995) which also deals with addresses from *Yellow Pages* (*NYNEX*). However, as noted above, we have source data that is more than a pure address database. Hence it seems there are *IR* systems that are even more closely related. Our system uses concepts related to those in (Strzalkowski, 1996) and (Strzalkowski et al., 1997). On the other hand, it does not make sense only to follow the *IR* approach for our project since the data is far too small for that — only 5 MB for the Colchester 1997 edition of the *Yellow Pages*. Furthermore, flat *IR* would overlook much of the information and structure it does have.

A number of online directory enquiry systems are actually in use. Some of the existing systems that offer information one would expect from the *Yellow Pages* are: *Electronic Yellow Pages (U.K.)*, *Scoot (U.K.)*<sup>2</sup>, *NYNEX Yellow Pages (U.S.A)*<sup>3</sup>, *BigBook (U.S.A)*<sup>4</sup> and *Switchboard (U.S.A)*<sup>5</sup>.

This type of commercial system functions quite differently from the previously mentioned *NLP* or *IR* systems. While they access large address databases of the same sort that we deal with they all share a number of limitations we hope to address with the development of the YPA:

- a very flat front end which at most offers pattern matching;
- a very simple lookup database which does not permit the access of free text hidden in the addresses;
- the inability to cope with words not found in the database (i.e. in the categories or names).

Therefore the YPA has a heterogeneous structure which employs *NLP* as well as *IR* ideas. Naturally this is important in the *online* dialogue, but it also affects the modular design of the system and the *off-line* construction of the *Backend*.

---

<sup>2</sup><http://www.scoot.co.uk>

<sup>3</sup><http://www.bigyellow.com>

<sup>4</sup><http://www.bigbook.com>

<sup>5</sup><http://www.switchboard.com>

## 4 Evaluation

During a two week period, members of British Telecom's Intelligent Systems Research Group had access to the YPA via the World Wide Web. They were given an outline of the information the system contained, but not told how the system should be queried. Over this time, some 238 *unique* queries were collected by the administrators of the system. These queries were used to conduct the technology focused evaluation of the YPA system. Each query was evaluated to see if it presented a list of addresses to the user, either first time or as the result of some dialogue with the system. If a list of addresses was not returned, we recorded the reason for this, and presented the figures as percentages of the total number of queries.

Of these queries, some 68% were deemed to be a system success, where the system performed as it was designed to, taking no account of quality of results or potential user satisfaction.

Of the queries that failed, 10% were due to some failure to correctly process the input (mainly a problem with our natural language frontend, or the stemmer). A further 12% failed due to a problem with our World Model - caused during the query expansion phase.

The final 10% of problems were caused by users - some typing errors, some people using out-of-bounds locations and some making statements of fact rather than asking queries (so we are less able to cope with input of the type *I am dying for a beer*).

We took the results of this evaluation to conclude that the system was performing as it was designed to, and that the robustness of the system was evident in that there were no fatal hardware or software errors during the time the system was online. We managed to collect a useful corpus of *Yellow Pages* queries, which could be used to guide the development of a new version of the system.

Furthermore, we could use the results of this evaluation to produce figures for the performance of the parsing strategy employed by our Natural Language Frontend.

The evaluation of parsing strategies used in Natural Language Frontends is an art in itself. There have been a large number of suggested methods for the purpose of guiding and monitoring the development of a parsing system (see (Carroll, Briscoe, and Sanfilippo, 1998) for a good survey).

We used the method of *coverage*, where we calculate the percentage of sentences assigned an analysis from an unannotated corpus. This method of evaluation is very crude, given that the rules in the grammar allow a variety of parse trees, which might not necessarily be correct. Although linguistic correctness was not a requirement, it is equally important that we have some form of correct structural identification, as we aim to preserve some structural information throughout the slot-filling procedure.

90% were successful from the parser point of view. 10% were recorded a *Frontend* failure - where either queries failed the parse completely, or information was misplaced, and put in incorrect slots.

At this stage, we wanted a measure of how good the results produced by the YPA were (and we knew from the first evaluation that 90% of the queries produced some sort of results). In fact we wanted to measure how closely retrieved addresses matched the input query. Importantly this takes no account of how satisfied prospective users would be with these answers, nor whether the system produced all available addresses. However, we did presume there would be some sort of correlation between the number of relevant addresses and user satisfaction.

For the precision evaluation, we used a corpus of 75 queries, collected from the Talking Pages call centre in Bristol. Of these 75 queries, 62 (83%) asked for information about addresses contained within our sample set (ie. existed within the Colchester area). For those, we had a value of 74% relevant to the input query.

The issue of how to deal with 'negative return queries' that is how to satisfactorily inform the user that no answers can be found, is one that we wish to address in the next stage of dialogue management development, and is probably key to the notion of user satisfaction with any system.

## 5 Future Work

The development of the *Frontend* is now largely complete for the purposes of the initial prototype. The overall system is currently progressing in two parallel directions.

Having gained access to a much richer data set, we are concentrating on better extraction techniques, developing methods of dealing with such *semi-structured data*. At the same time the dialogue management component is under redevelopment, to achieve a level of abstraction leading hopefully to a number of claims which can be made about general dialogue management.

Finally, once these are complete attention will focus on a full scale evaluation, with the emphasis very firmly on a measure of user satisfaction.

## 6 Conclusions

We have implemented a *Frontend* to our system which preserves *some* syntactic structure in the slots. This gives us two distinct advantages over other systems. We do not have simple key words, which allows us to use a degree of intelligence during manipulation in query construction. This enables us to return addresses which more closely match the users requirements. Furthermore, it allows the user to ask specific questions of the system based on those requirements. The aim of our system is to provide a high degree of *precision*.

## 7 Acknowledgments

We would like to thank K.C. Tsui and Wayne Wobcke for their input to this project, and all of the users at BT Laboratories who participated in the evaluation exercise.

## References

- Abney, S. 1991. Parsing by chunks. In S. Abney R. Berwick and C. Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers.
- Agarwal, R. 1997. Towards a PURE Spoken Dialogue System for Information Access. In *Proceedings of the ACL/EACL Workshop on "Interactive Spoken Dialog Systems: Bringing Speech and NLP Together in Real Applications"*, pages 90–97, Madrid.
- Alshawi, H., editor. 1992. *The Core Language Engine*. Cambridge, MAS: MIT Press.
- Aust, H., M. Oerder, F. Seide, and V. Steinbiss. 1995. The Philips automatic train timetable information system. *Speech Communication*, 17:249–262.
- Bannacef, S. K., H. Bonneau-Maynard, J. L. Gauvain, L. Lamel, and W. Minker. 1994. A Spoken Language System for Information Retrieval. In *Proceedings of the International Conference on Speech and Language Processing*, Yokohama, Japan.
- Bannacef, S. K., L. Devillers, S. Rosset, and L. Lamel. 1996. Dialog in the RAILTEL Telephone-Based System. In *Proceedings of the International Conference on Speech and Language Processing*, pages 550–553, Philadelphia.
- Briscoe, T., J. Carroll, G. Carroll, S. Federici, G. Grefenstette S. Montemagni, V. Pirrelli, I. Prodanof, M. Rooth, and M. Vannocchi. 1997. Phrasal Parser Software - Deliverable 3.1. <http://www.ilc.pi.cnr.it/sparkle.html>.
- Carroll, J., T. Briscoe, and A. Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1<sup>st</sup> International Conference on Language Resources and Evaluation*, pages 447 – 454, Granada, Spain.
- De Roeck, A., U. Kruschwitz, P. Neal, P. Scott, S. Steel, R. Turner, and N. Webb. 1998. YPA - an intelligent directory enquiry assistant. *BT Technology Journal*, 16(3):145–155.
- Gazdar, G. and C. Mellish. 1989. *Natural Language Processing in PROLOG: An Introduction to Computational Linguistics*. Addison Wesley.
- Glass, J., G. Flammia, D. Goodine, M. Phillips, J. Polifroni, S. Sakai, S. Seneff, and V. Zue. 1995. Multilingual Spoken-Language Understanding in the MIT VOYAGER System. *Speech Communication*, 17:1–18.
- Grosz, B. J., D. E. Appelt, P. A. Martin, and F. C. N. Pereira. 1987. TEAM: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32:173–243.
- Heisterkamp, P., S. McGlashan, and N. Youd. 1992. Dialogue Semantics for an Oral Dialogue System. In *Proceedings of the International Conference of Spoken Language Processing*, Banff, Canada.
- Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. 1978. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, June:105–147.

- McGlashan, S., N. Fraser, N. Gilbert, E. Bilange, P. Heisterkamp, and N. Youd. 1992. Dialogue Management for Telephone Information Systems. In *Proceedings of the International Conference on Applied Language Processing*, Trento, Italy.
- Minsky, M. 1975. A Framework for Representing Knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, pages 211–277.
- Sikorski, T. and J. F. Allen. 1996. A task-based evaluation of the TRAINS-95 dialogue system. In *Proceedings of the Workshop on Dialog Processing in Spoken Language Systems, ECAI-96*, Budapest.
- Strzalkowski, T. 1996. Natural Language Information Retrieval: TREC-4 Report. In *Proceedings of the Fourth Text Retrieval Conference (TREC-4)*, NIST Special Publication 500-236.
- Strzalkowski, T., L. Guthrie, J. Karlgren, J. Leistensnider, F. Lin, J. Perez-Carballo, T. Straszheim, J. Wang, and J. Wilding. 1997. Natural Language Information Retrieval: TREC-5 Report. In *Proceedings of the Fifth Text Retrieval Conference (TREC-5)*, NIST Special Publication 500-238.
- Tomita, M. 1985. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic.
- Wahlster, Wolfgang. 1993. Verbmobil: Translation of Face-to-Face Dialogues. In *Proceedings of the 3<sup>rd</sup> European Conference on Speech Communication and Technology*, pages 29–38, Berlin, Germany.
- Zue, V. 1994. Toward Systems that Understand Spoken Language. *IEEE Expert Magazine*, February:51–59.