A Natural Language Shell and Tools for Customizing the Dialogue in Natural Language Interfaces

Arne Jönsson

Department of Computer and Information Science
Linköping University
S- 581 83 LINKÖPING, SWEDEN
Phone: +46 13281717
Email: ARJ@IDA.LIU.SE

Abstract

The language used in an interaction between a human and a computer depends on the application as well as the user. This paper describes a method for building domain specific natural language interfaces by updating knowledge bases for a domain independent natural language shell.

The interface uses a uniform knowledge representation for linguistic, dialogue and domain knowledge. Further, it contains a dialogue manager which can be customized for different applications by changing or updating dialogue objects describing different interaction situations. Information about the interactions is gathered from experiments with a simulated human-computer interface, Wizard of Oz experiments. These are incorporated into the interface using a system which extracts information from a tagged corpus and inserts it into the knowledge bases. Information on all the different levels of an NLI can be extracted from the corpus in this way, but in this paper emphasis is laid on extracting and incorporating dialogue management information.

1 Introduction

The building of one single domain independent natural language interface is becoming less interesting today. Instead it is agreed that one needs to build a new system for each new application. This can be done in different ways, either by updating an existing system with domain specific information or by rewriting the whole system. Our approach favours the former.

1.1 Acquisition of Sublanguages

Natural language interfaces can be used in many different applications, e.g. data base systems, consultation systems and configuration systems. Each new application has its own vocabulary, phrase structure and interaction style. This is what is called a sublanguage (Grishman & Kittredge, 1987), i.e. a subset of a natural language. A sublanguage is not only defined by its grammar and lexicon, but also by its form of interaction, i.e factors like how the user and system handle clarifications, who takes the initiative, what is cooperative in a certain application, what are the user categories etc. This means that we are interested in finding which sublanguages that are used in various domains. We do this by simulations where users interact with what they believe is a natural language interface but in fact their utterances are interpreted by a human. These experiments are often called Wizard of Oz-experiments. In Jönsson & Dahlbäck (1988) and Dahlbäck & Jönsson (1989) we have presented the results of our experiments conducted with the purpose of studying human-computer interaction in general. Jönsson (1990) did some initial analysis with the purpose of developing strategies for dialogue management for different natural language interfaces.

1.2 Transportable Natural Language Interfaces

Transportable natural language interfaces do not attempt to deal with a complete natural language but rather a sublanguage as described above.

The problem of adapting a database domain description is studied in systems like TEAM and TELI and is also used in commercial NLI's like IBM's LanguageAccess. TEAM (Martin, Appelt, Grosz & Pereira 1985) is a system designed for customization to different data base applications. TEAM allows the end user to update the interface with new concepts and also to integrate these into the domain knowledge base. One of the requirements was that the information necessary to adapt to a new database should be acquired from the end user. Hence TEAM does not consider information relating to the grammar or dialogue behaviour. The same aim lies behind the TELI system (Ballard & Stumberger, 1986), namely that the end-users do the customization. TELI uses a set of tools to aid the end-user in the customization of domain concepts. The idea is somewhat similar to the ones on user-derived interfaces, (see section 1.3) but it still only allows adding new concepts by an end-user who is not a linguist.

Another problem is to allow for domain dependent lexicon, syntax and semantics customization. The problem of syntactic-semantic role relations is addressed in LUKE (Wroblewski & Rich, 1989), where linguistic knowledge and domain knowledge are integrated in the same knowledge-base, allowing for simultaneous development and interleaving of syntactic and semantic processes. This requires a more sophisticated system builder.

However, a cooperative NLI must be more than a simple question-answering system, it must also participate in a coherent dialogue with the user. Many solutions

to this problem are centered around the idea that the dialogue is planned by recognizing the users' goals and intentions, Carberry (1990) presents an overview. These approaches deal with general discourse and hence the problem of transportability is not addressed.

The HAM-ANS project (Hoeppner, *et al* 1983) took an even broader view. Here the interest was not only in adapting the system to a new database, but instead they realized that there are different interaction situations that differ not only with respect to the background system but also with respect to dialogue type, user type, intended system behaviour and discourse domain. In HAM-ANS a core system was developed which could be adapted to different applications. The idea was that the core system should be the same for every application. This is then enhanced with new information for a new application. HAM-ANS separates out the processes from the different knowledge sources which makes it easier to adapt a knowledge base to a new application, as no processing mechanisms needs updating. They use the core to develop NLI's to a hotel reservation system, a system for analysing traffic scenes and a consultations system on fishery data. There is no tool for customization and the knowledge sources are of different types, which makes it difficult to develop tools for customization.

1.3 A Methodology for Customization

In software engineering the notion of rapid prototyping is a well-known concept meaning that a system is developed in cooperation with the end user. The idea is to build a prototype that is enhanced with new features as the end users use the system. The final system will thus hopefully reflect the users' needs instead of those of the system engineers'.

A similar approach was taken by Kelley (1983) where he used a method for developing a natural language interface in six steps. The first step is to analyse the task and the second to develop semantic primitives for that task. The third step then is called the first Wizard of Oz-step. In this step Kelley lets the subject interact with what they believe is a natural language interface but which in fact is a human simulating such an interface. This provides data that are used to build a first version of the interface, step four. Kelley starts without any grammar or lexicon. The rules and lexical entries are those used by the users during the simulation.

Next, in step 5, Kelley starts to improve his interface by conducting new Wizard of Oz simulations, this time with the interface running. However, when the user/subject enters a query that the system cannot handle, the Wizard takes over and produces an appropriate response. This interaction is logged and later the system is updated to be able to handle the situations where the wizard was responding, step six. The advantage is that the user's interaction is not interrupted and a more realistic dialogue is thus obtained.

The method used by Kelley of running a simulation in parallel with the interface was developed by Chapanis (1982) and used in his experiments on human-computer interaction. It was also used by Good, Whiteside, Wixon & Jones (1984). They developed a command language interface to an e-mail system by this iterative design method which they call UDI (User-derived Interface). Kelley and Good *et al* focus on updating the lexical and grammatical knowledge and are not concerned with dialogue behaviour.

In this paper we go further by using a natural language interface system which is based on modern linguistic theories, which use an object-oriented knowledge representation language throughout and which is capable of participating in a coherent dialogue with the user, LINLIN (LInköping Natural Language INterface) (Ahrenberg, Jönsson & Dahlbäck, 1990). In LINLIN the interface is customized to a certain application using a process inspired by the method of user-derived interfaces. The paper will focus on the dialogue management part of the interface.

2 LINLIN

In LINLIN linguistic knowledge and domain knowledge are integrated in the same knowledge base allowing interleaving of syntactic and semantic processes (Ahrenberg, 1989). Semantic interpretation is object-oriented (Hirst, 1987) and involves the linking of linguistic objects (parts of utterances) to objects (instances, classes, properties) of the universe of discourse, of which the system may have independent knowledge.

In LINLIN we regard a dialogue manager (henceforth: DM) as the central controlling module which directs the dialogue and keeps a record of the dialogue history. It can be viewed as a controller of resources (parser, instantiator, deep generator, surface generator and translator). The resources are regarded as domain independent processes accessing different application and domain dependent knowledge sources (grammar, lexicon, domain objects, dialogue objects and translation principles), see figure 2.

DM uses information from three knowledge sources; domain object descriptions which describe the domain concepts and their relations, dialogue object descriptions, and finally information about the ongoing dialogue modelled in a dialogue tree.

2.1 The dialogue manager

The dialogue manager is presented in Jönsson (1991) but some of its distinguishing features needs to be presented before we can discuss how the DM is customized.

Reichman (1985) describes a discourse grammar based on the assumption that a conversation can be described using conventionalized discourse rules. We argue similarly (Ahrenberg, Jönsson & Dahlbäck, 1990) that a segment structure inferred from the structure and content of the utterances is primary when describing a dialogue. This proposal has been criticized by for instance Levinson (1981) as not accurately describing a naturally occurring discourse, but for a restricted sublanguage in a limited domain even Levinson agrees that a speech act theory of dialogue may have its utility. Reichman uses surface linguistic phenomena for recognizing the speakers structuring of the discourse. However, we found very little use of surface linguistic cues in our dialogues (Jönsson, 1990). Instead the users often follow a local discourse plan with a clear goal such as obtaining information about some items. This does not mean that the users always fulfil one plan or that there are no clarifications, but they can be handled using the same mechanisms, as described below.

We structure the communication hierarchically using three different categories of dialogue objects. Instances of dialogue objects form a dialogue tree which represent the dialogue as it develops in the interaction, see figure 1. The root category is called Dialogue (D), the interme-

diate category Initiative-Response (IR) and the smallest unit handled by our dialogue manager is the move. An utterance can consist of more than one move and is thus regarded as a sequence of moves. A move object contains information about a move. Moves are categorized according to type of illocutionary act and topic. Some typical move types are: Question (Q), Assertion and declaration of intent (AS), Answer (A) and Directive (DI). Topic describes which knowledge source to consult — the background system, i.e. solving a task (T), the ongoing discourse (D) or the organisation of the background system (S). For brevity when we refer to a move with its associated topic, the move type is subscribed with topic, e.g. Q_T.

Normally two moves constitute an exchange of information which begin with an initiative followed by a response (IR). The initiative can come from the system or the user. A typical IR-unit in a question-answer data base application is a task related question followed by a successful answer Q_T/A_T. Other typical IR-units are: Q_S/A_S for clarification request, Q_T/AS_S when the requested information is not in the data base, Q_D/A_D for questions about the ongoing dialogue.

A dialogue object has two parts. One part contains information about static properties like type, topic, initiator, responder, salient objects and attributes and contextual information. Another part of the dialogue object is a process description of the actions performed when executing the object, we call this an action plan. The different action plans are simple sequences of actions, e.g. the action plan for a user initiated IR-unit is:

(create-move user)
(access)
(create-move system)
(up).

The execution of this action plan, however, is context dependent, i.e. the behaviour depends on the information in the static part. For instance the action (access) uses information about topic to access the appropriate knowledge base.

The action plan is pushed on to a stack and execution is controlled from the stack of the current active node. Actually we have one stack for each node in the tree, see figure 1. This reflects our distributed approach and has the advantage that we do not need to manipulate one single global stack with a complex control mechanism. Every action plan terminates with the execution of an action which transfers control to a node above the current node, often the father node, and then execution is resumed from the action plan of that node.

One important feature of the dialogue manager is that every node in the tree is responsible for its own correctness. For instance the plan for a Q_T/A_T, i.e. a normal task related question-answer, contains no reparation strategies for missing information to the background system. If the interpreter fails to access the background system due to lack of information, the translator signals this to the DM which creates an instance of an IR-unit for clarification request (say Q_D/A_D) and inserts it into the context of Q_T/A_T. The action plan for clarification request then generates a move explaining the missing information and creates an instance of a user move waiting for user input. This has the advantage that the plans can be made very simple, as they need only be aware of their own local behaviour and context. Further, the plans are more generally applicable; the plan for a clarification request is similar for any level in the dialogue tree.

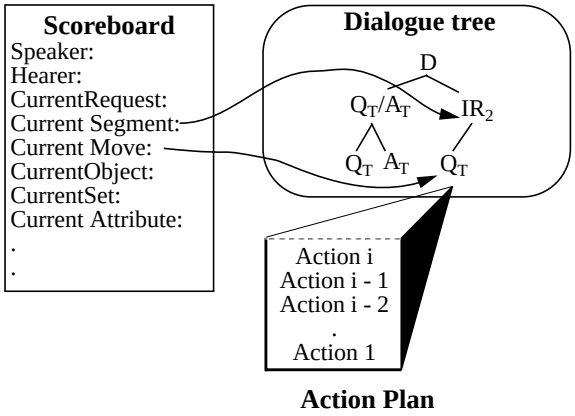

Figure 1. The Internal structures used by the dialogue manager.

3 Customization

Customizing LINLIN means augmenting the various knowledge sources seen to the right in figure 2. LINLIN is not to be regarded as having a completely empty knowledge base. Rather we start with a basic lexicon and grammar and also some dialogue object descriptions that are common to many applications. These knowledge sources constitute a shell which needs to be updated with domain dependent concepts, rules and interaction principles. This is done by a person, or a team of persons, cf. Wroblewski & Rich (1989), with some background in linguistics. I refer to such a person as a *language engineer.*

However, the language engineer is not to be regarded as the only person involved in the customization. The translator, i.e. the module interfacing the background system, needs to be rewritten for each new application. The same translator can, however, be used in different domains using the same application through the means of translation rules for a specific domain, e.g. different SQL databases. There should also be a tool for end user customization, like the ones used by TEAM or LUKE.

One novel aspect of our approach is that the information available to the language engineer does not only consist of the requirements and application descriptions from the customer, but also a corpus collected using Wizard of Oz-experiments with the end-users, see the upper right in figure 2. This information is integrated into the system's knowledge bases using two different tools, a knowledge base tool and a system for acquiring information from a corpus. Adding the information provided by the customer using a knowledge base tool is a well-known, although not in any respect solved, task, and is not elaborated further upon in this paper.

Another aspect of the customization, deals with performance improvement. This can in part be achieved by reducing the size of the knowledge bases. Kelley (1983) developed his interface with no initial knowledge, thus the only concepts were those found during the simulations. This will give a limited, but hopefully sufficient, knowledge base for one application, which was Kelley's goal. However, my purpose is to develop an interface which can be customized to many different applications and thus I would like to reuse as much previous information as possible. One way to achieve this efficiently is to

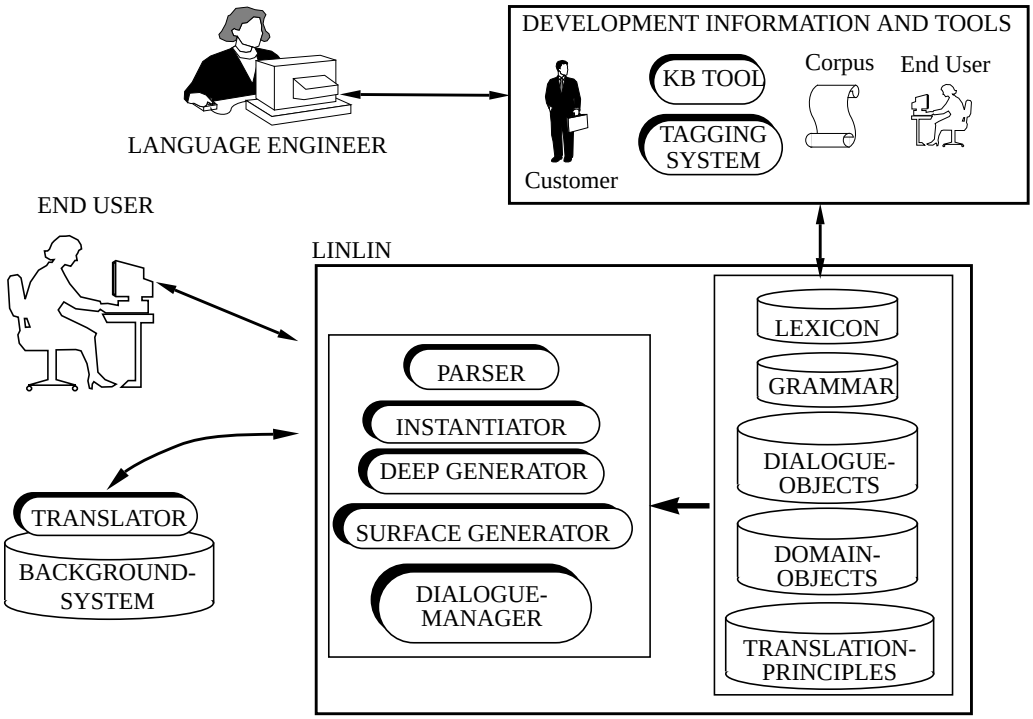

Figure 2. A Natural Language Development Environment

structure the knowledge bases into two levels. One primary level for knowledge that is found to be relevant in the current application, either given by the customer or through the simulations, and another secondary level containing the rest. Thus, the use of simulations can also be used for improving performance. However, we have as yet no knowledge base that is large enough to be considered for such modifications.

3.1 The tagging system

For analysis of the corpus we use an interactive graphical tagging system called DagTag (Ahrenberg & Jönsson, 1989). In DagTag the tag is a directed acyclic graph, a dag, used in the encoding of information. DagTag allows tagging on various levels of description, e.g. lexical, syntactic/semantic and dialogue. Thus, we can use the same tool to encode information for all the different knowledge sources utilized in our interface. First the segment to be tagged, e.g a move, is marked, and then the tagging information is assigned, e.g. Q_T. This process is much speeded up with the use of templates. A template is a ready-made dag of arbitrary complexity and specification. For the most common tags the dag is fully specified, but for the least frequently used it is better to use a general dag which is specified when used. The templates are constructed using a graphical dag-editor.

Further, as the tags have the structure of dags, which is the same format as used in LINLINs knowledge bases, we can extract information from a tagged corpus to be incorporated into LINLINs knowledge bases. We have carried out initial experiments on the acquisition of grammar and lexicon from a tagged corpus (Jönsson & Ahrenberg 1990).

The task at hand now is how to do this for the other knowledge bases, initially the dialogue objects.

3.2 Customizing the dialogue

The use of dialogue objects makes it possible for us to also customize the interaction principles from a tagged corpus. As said above, the dialogue objects consisted of one static part containing contextual information and one part describing a prototypical behaviour. Now, the process part is the same for dialogue objects with the same basic behaviour, c.f. the action plan for a user initiated IR-unit described in section 2.1. The number of such generic dialogue objects is very small. For a database information retrieval application the different types are: D-unit, system initiated IR-unit, user initiated IR-unit, generate move and interpret move. These generic dialogue objects are used to create instances of specific dialogue objects for a certain situation, e.g. a Q_T/A_T. They are constructed as templates in DagTag, i.e. the same tool that is used in the subsequent knowledge acquisition phase.

The dialogue is manually tagged using descriptors for move types and IR-units. Figure 3 shows a simplified dag corresponding to a Q_T/A_T initiated by the system[1]. The information in this dag is used to describe one instance of a dialogue object. The general plan for a system initiated IR-unit is specified as a Q_T/A_T by extracting information from this dag and encode it in the static part of the dialogue object as values to the different attributes. This then provides information to the DM for instance on what type of move that is legal in this situa-

1. This example is meant to illustrate how the information is extracted. The Q/A IR-unit is such a basic IR-unit that it is probably regarded as domain independent.

tion, the value in the Response slot. In this case it says that the user may respond with an A.

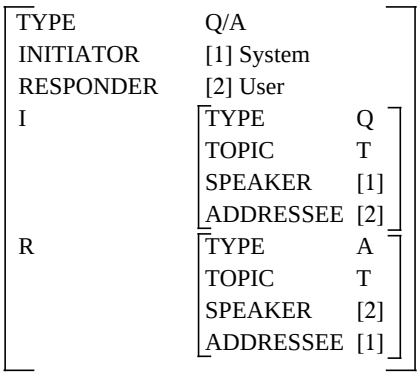

Figure 3. A simplified dag for a Q_T/A_T IR-unit

Suppose we encounter an utterance where the user responds to a Q_T with an AS_T, i.e. we have a Q_T/AS_T as a legal IR-unit. For simplicity we ignore the other properties. This new IR-unit does not constitute a new IR-unit of type Q_T/AS_T instead it corresponds to modifying the previously created Q_T/A_T to something like $Q_T/(A_T \vee AS_T)$. The $(A \vee AS)$ is a value in the Response slot allowing also an AS to be a legal move to a question with topic T.

The dag in figure 3 also contains information about relevant properties for a move-unit, i.e. the values to the attributes I and R. This information can be used for specifying the move dialogue objects by extracting information like type, topic, addressee and speaker.

Further, as the customization is carried out by a language engineer, it is possible to define move and IR units consisting of new action sequences. This can be used for instance to build more complex IR-units like $DI_T/ACK_T/ACK_D$.

It is also possible to customize the referent resolution algorithms used by the NLI. Information about salient objects is represented in the dialogue tree and is used by the instantiator and deep generator through a scoreboard, see figure 1. Associated with the slots on the scoreboard are access functions. The access functions can be altered, allowing the search for a referent to an anaphoric expression to be application dependent.

4 Discussion and current status

We have collected and analysed a number of dialogues. In the corpus most of the utterances are user-initiated task questions provoking a response which is retrieved from the data base. The response is mostly an A_T and in some cases an AS_S (i.e. no information). The number of clarifications is low and, interestingly, when the system has initiated the clarification, the user does not respond to the clarification, instead he starts a new initiative. Further, there are some ending and greeting sequences and one instance where the system/wizard opens a new IR-unit.

I have partially analysed a small part of the corpus consisting of three Wizard of Oz dialogues with the purpose of developing dialogue objects. The application is information retrieval from a data base containing information on properties of used car models. My intention is, however, to investigate the other types of applications that we are dealing with, such as advisory systems and configuration systems.

In the work by Kelley (1983) and Good *et al.* (1984) the customization process was saturated after a certain number of dialogues. We have not analyzed the corpus enough to say whether this will happen to the dialogue objects, too. However, for data base systems there seems to be a limited set of move and IR-unit types, but whether this is true for other applications is an open question.

The current status of LINLIN is that, except for the deep generator, there are pilot versions of the different processes, running with small knowledge bases. There is also a pilot version of the dialogue manager, but it is not yet integrated with the other modules. The tagging system has been used with a very small portion of the corpus for augmenting the lexicon and grammar. Currently we are working on using the tagging system to create the generic dialogue objects and we will also use DagTag to specify the dialogue objects.

5 Acknowledgements

This work is part of a larger project carried out at the Natural Language Processing Laboratory, Linköping, sponsored by the Swedish National Board for Technical Development, STU and Swedish Council for Research in the Humanities and Social Sciences (HSFR). Many of the ideas on dialogue management and customization have evolved during discussions with my colleagues in the lab, especially Lars Ahrenberg and Nils Dahlbäck.

Åke Thurée did most of the coding for the DM in Xerox Common Lisp on a Sun Sparc Station. Ivan Rankin, Mats Wirén and Richard Hirsch have read previous versions of the paper and provided many valuable comments.

References

Ahrenberg, L. (1989). On the integration of linguistic knowledge and world knowledge in natural language understanding. In Ö. Dahl & K. Fraurud (eds.) *Papers from the First Nordic Conference on Text Comprehension in Man and Machine*, Institute of Linguistics, University of Stockholm, pp. 1-11.

Ahrenberg, L. and Jönsson, A. (1989) An interactive system for tagging dialogues. *Literary and Linguistic Computing* **3**(2), 66-70.

Ahrenberg, L., Jönsson, A. & Dahlbäck, N. (1990) Discourse Representation and Discourse Management for a Natural Language Dialogue System, *To appear in Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine*, Täby, Stockholm.

Ballard, B. W. & Stumberger, D. E. (1986) Semantic Acquisition in TELI: A Transportable, User-Customized Natural Language Processor, *Proceedings of the 24th Annual Meeting of the ACL*, New York.

Carberry, S. (1990) *Plan Recognition in Natural Language Dialogue*, MIT Press.

Chapanis, A. (1982) Appendix: Man-Computer Research at Johns Hopkins, *Information Technology and Psychology*, Kasschau, Lachman & Laugherty (Eds), Raeger Publishers.

Dahlbäck, N. & Jönsson, A. (1989) Empirical Studies of Discourse Representations for Natural Language Inter-

faces, *Proceedings of the Fourth Conference of the European Chapter of the ACL*, Manchester. 1989.

Good, M. D., Whiteside, J. A., Wixon, D. R. & Jones, S. J., (1984) Building a User-Derived Interface, *Communications of the ACM*, Vol. 27, No 10, pp 1032-1043.

Grishman, R. & Kittredge, R. (eds.) 1986. *Analysing language in restricted domains*. Hillsdale, N.J.: Erlbaum.

Hirst, G. (1987). *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge University Press.

Hoeppner, W., Christaller, T., Marburger, H., Morik, K., Nebel, B., O'Leary, M. & Wahlster, W. (1983) Beyond Domain Experience: Experience with the Development of a German Language Access System To Highly Diverse Background Systems, Research Report, University of Hamburg, Bericht ANS-16.

Jönsson, A. (1991) A Dialogue Manager Using Initiative-Response Units and Distributed Control, *Proceedings of the 5th Conference of the European Chapter of the ACL*, Berlin, Germany.

Jönsson, A. (1990) Application-Dependent Discourse Management for Natural Language Interfaces: An Empirical Investigation, *Papers from the Seventh Scandinavian Conference of Computational Linguistics*, Reykjavik, Island.

Jönsson, A. & Ahrenberg, L. (1990) Extensions of a descriptor-based tagging system into a tool for the generation of unification-based grammars. To appear in *Research in Humanities Computing* **1**.

Jönsson, A. & Dahlbäck, N. (1988) Talking to a Computer is not Like Talking to Your Best Friend. *Proceedings of The first Scandinivian Conference on Artificial Intelligence*, Tromsø, Norway.

Kelley, J. F. (1983) Natural Language and Computers: Six Empirical Steps for Writing an Easy-to-Use Computer Application, PhD thesis, The Johns Hopkins University.

Levinson, S. C. (1981) Some Pre-Observations on the Modelling of Dialogue, *Discourse Processes*, No 4, pp 93-116.

Martin, P., Appelt, D. E., Grosz, B. J. & Periera, F. (1985) TEAM: An Experimental Transportable Natural-Language Interface, *IEEE quarterly bulletin on Database Engineering*, Vol. 8, No. 3.

Reichman, R. (1985) *Getting Computers to Talk Like You and Me*, MIT Press, Cambridge, MA.

Wroblewski, D. A. and Rich, E. A. (1989) LUKE: An Experiment in the Early Integration of Natural Language Processing. *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas pp. 186-191.

A Natural Language Shell and Tools for Customizing the Dialogue in Natural Language Interfaces

Arne Jönsson

Department of Computer and Information Science
Linköping University
S- 581 83 LINKÖPING, SWEDEN
Phone: +46 13281717
Email: ARJ@IDA.LIU.SE

Abstract

The language used in an interaction between a human and a computer depends on the application as well as the user. This paper describes a method for building domain specific natural language interfaces by updating knowledge bases for a domain independent natural language shell.

The interface uses a uniform knowledge representation for linguistic, dialogue and domain knowledge. Further, it contains a dialogue manager which can be customized for different applications by changing or updating dialogue objects describing different interaction situations. Information about the interactions is gathered from experiments with a simulated human-computer interface, Wizard of Oz experiments. These are incorporated into the interface using a system which extracts information from a tagged corpus and inserts it into the knowledge bases. Information on all the different levels of an NLI can be extracted from the corpus in this way, but in this paper emphasis is laid on extracting and incorporating dialogue management information.

1 Introduction

The building of one single domain independent natural language interface is becoming less interesting today. Instead it is agreed that one needs to build a new system for each new application. This can be done in different ways, either by updating an existing system with domain specific information or by rewriting the whole system. Our approach favours the former.

1.1 Acquisition of Sublanguages

Natural language interfaces can be used in many different applications, e.g. data base systems, consultation systems and configuration systems. Each new application has its own vocabulary, phrase structure and interaction style. This is what is called a sublanguage (Grishman & Kittredge, 1987), i.e. a subset of a natural language. A sublanguage is not only defined by its grammar and lexicon, but also by its form of interaction, i.e factors like how the user and system handle clarifications, who takes the initiative, what is cooperative in a certain application, what are the user categories etc. This means that we are interested in finding which sublanguages that are used in various domains. We do this by simulations where users interact with what they believe is a natural language interface but in fact their utterances are interpreted by a human. These experiments are often called Wizard of Oz-experiments. In Jönsson & Dahlbäck (1988) and Dahlbäck & Jönsson (1989) we have presented the results of our experiments conducted with the purpose of studying human-computer interaction in general. Jönsson (1990) did some initial analysis with the purpose of developing strategies for dialogue management for different natural language interfaces.

1.2 Transportable Natural Language Interfaces

Transportable natural language interfaces do not attempt to deal with a complete natural language but rather a sublanguage as described above.

The problem of adapting a database domain description is studied in systems like TEAM and TELI and is also used in commercial NLI's like IBM's LanguageAccess. TEAM (Martin, Appelt, Grosz & Pereira 1985) is a system designed for customization to different data base applications. TEAM allows the end user to update the interface with new concepts and also to integrate these into the domain knowledge base. One of the requirements was that the information necessary to adapt to a new database should be acquired from the end user. Hence TEAM does not consider information relating to the grammar or dialogue behaviour. The same aim lies behind the TELI system (Ballard & Stumberger, 1986), namely that the end-users do the customization. TELI uses a set of tools to aid the end-user in the customization of domain concepts. The idea is somewhat similar to the ones on user-derived interfaces, (see section 1.3) but it still only allows adding new concepts by an end-user who is not a linguist.

Another problem is to allow for domain dependent lexicon, syntax and semantics customization. The problem of syntactic-semantic role relations is addressed in LUKE (Wroblewski & Rich, 1989), where linguistic knowledge and domain knowledge are integrated in the same knowledge-base, allowing for simultaneous development and interleaving of syntactic and semantic processes. This requires a more sophisticated system builder.

However, a cooperative NLI must be more than a simple question-answering system, it must also participate in a coherent dialogue with the user. Many solutions

to this problem are centered around the idea that the dialogue is planned by recognizing the users' goals and intentions, Carberry (1990) presents an overview. These approaches deal with general discourse and hence the problem of transportability is not addressed.

The HAM-ANS project (Hoeppner, *et al* 1983) took an even broader view. Here the interest was not only in adapting the system to a new database, but instead they realized that there are different interaction situations that differ not only with respect to the background system but also with respect to dialogue type, user type, intended system behaviour and discourse domain. In HAM-ANS a core system was developed which could be adapted to different applications. The idea was that the core system should be the same for every application. This is then enhanced with new information for a new application. HAM-ANS separates out the processes from the different knowledge sources which makes it easier to adapt a knowledge base to a new application, as no processing mechanisms needs updating. They use the core to develop NLI's to a hotel reservation system, a system for analysing traffic scenes and a consultations system on fishery data. There is no tool for customization and the knowledge sources are of different types, which makes it difficult to develop tools for customization.

1.3 A Methodology for Customization

In software engineering the notion of rapid prototyping is a well-known concept meaning that a system is developed in cooperation with the end user. The idea is to build a prototype that is enhanced with new features as the end users use the system. The final system will thus hopefully reflect the users' needs instead of those of the system engineers'.

A similar approach was taken by Kelley (1983) where he used a method for developing a natural language interface in six steps. The first step is to analyse the task and the second to develop semantic primitives for that task. The third step then is called the first Wizard of Oz-step. In this step Kelley lets the subject interact with what they believe is a natural language interface but which in fact is a human simulating such an interface. This provides data that are used to build a first version of the interface, step four. Kelley starts without any grammar or lexicon. The rules and lexical entries are those used by the users during the simulation.

Next, in step 5, Kelley starts to improve his interface by conducting new Wizard of Oz simulations, this time with the interface running. However, when the user/subject enters a query that the system cannot handle, the Wizard takes over and produces an appropriate response. This interaction is logged and later the system is updated to be able to handle the situations where the wizard was responding, step six. The advantage is that the user's interaction is not interrupted and a more realistic dialogue is thus obtained.

The method used by Kelley of running a simulation in parallel with the interface was developed by Chapanis (1982) and used in his experiments on human-computer interaction. It was also used by Good, Whiteside, Wixon & Jones (1984). They developed a command language interface to an e-mail system by this iterative design method which they call UDI (User-derived Interface). Kelley and Good *et al* focus on updating the lexical and grammatical knowledge and are not concerned with dialogue behaviour.

In this paper we go further by using a natural language interface system which is based on modern linguistic theories, which use an object-oriented knowledge representation language throughout and which is capable of participating in a coherent dialogue with the user, LINLIN (LInköping Natural Language INterface) (Ahrenberg, Jönsson & Dahlbäck, 1990). In LINLIN the interface is customized to a certain application using a process inspired by the method of user-derived interfaces. The paper will focus on the dialogue management part of the interface.

2 LINLIN

In LINLIN linguistic knowledge and domain knowledge are integrated in the same knowledge base allowing interleaving of syntactic and semantic processes (Ahrenberg, 1989). Semantic interpretation is object-oriented (Hirst, 1987) and involves the linking of linguistic objects (parts of utterances) to objects (instances, classes, properties) of the universe of discourse, of which the system may have independent knowledge.

In LINLIN we regard a dialogue manager (henceforth: DM) as the central controlling module which directs the dialogue and keeps a record of the dialogue history. It can be viewed as a controller of resources (parser, instantiator, deep generator, surface generator and translator). The resources are regarded as domain independent processes accessing different application and domain dependent knowledge sources (grammar, lexicon, domain objects, dialogue objects and translation principles), see figure 2.

DM uses information from three knowledge sources; domain object descriptions which describe the domain concepts and their relations, dialogue object descriptions, and finally information about the ongoing dialogue modelled in a dialogue tree.

2.1 The dialogue manager

The dialogue manager is presented in Jönsson (1991) but some of its distinguishing features needs to be presented before we can discuss how the DM is customized.

Reichman (1985) describes a discourse grammar based on the assumption that a conversation can be described using conventionalized discourse rules. We argue similarly (Ahrenberg, Jönsson & Dahlbäck, 1990) that a segment structure inferred from the structure and content of the utterances is primary when describing a dialogue. This proposal has been criticized by for instance Levinson (1981) as not accurately describing a naturally occurring discourse, but for a restricted sublanguage in a limited domain even Levinson agrees that a speech act theory of dialogue may have its utility. Reichman uses surface linguistic phenomena for recognizing the speakers structuring of the discourse. However, we found very little use of surface linguistic cues in our dialogues (Jönsson, 1990). Instead the users often follow a local discourse plan with a clear goal such as obtaining information about some items. This does not mean that the users always fulfil one plan or that there are no clarifications, but they can be handled using the same mechanisms, as described below.

We structure the communication hierarchically using three different categories of dialogue objects. Instances of dialogue objects form a dialogue tree which represent the dialogue as it develops in the interaction, see figure 1. The root category is called Dialogue (D), the interme-

diate category Initiative-Response (IR) and the smallest unit handled by our dialogue manager is the move. An utterance can consist of more than one move and is thus regarded as a sequence of moves. A move object contains information about a move. Moves are categorized according to type of illocutionary act and topic. Some typical move types are: Question (Q), Assertion and declaration of intent (AS), Answer (A) and Directive (DI). Topic describes which knowledge source to consult — the background system, i.e. solving a task (T), the ongoing discourse (D) or the organisation of the background system (S). For brevity when we refer to a move with its associated topic, the move type is subscribed with topic, e.g. Q_T.

Normally two moves constitute an exchange of information which begin with an initiative followed by a response (IR). The initiative can come from the system or the user. A typical IR-unit in a question-answer data base application is a task related question followed by a successful answer Q_T/A_T. Other typical IR-units are: Q_S/A_S for clarification request, Q_T/AS_S when the requested information is not in the data base, Q_D/A_D for questions about the ongoing dialogue.

A dialogue object has two parts. One part contains information about static properties like type, topic, initiator, responder, salient objects and attributes and contextual information. Another part of the dialogue object is a process description of the actions performed when executing the object, we call this an action plan. The different action plans are simple sequences of actions, e.g. the action plan for a user initiated IR-unit is:

(create-move user)
(access)
(create-move system)
(up).

The execution of this action plan, however, is context dependent, i.e. the behaviour depends on the information in the static part. For instance the action (access) uses information about topic to access the appropriate knowledge base.

The action plan is pushed on to a stack and execution is controlled from the stack of the current active node. Actually we have one stack for each node in the tree, see figure 1. This reflects our distributed approach and has the advantage that we do not need to manipulate one single global stack with a complex control mechanism. Every action plan terminates with the execution of an action which transfers control to a node above the current node, often the father node, and then execution is resumed from the action plan of that node.

One important feature of the dialogue manager is that every node in the tree is responsible for its own correctness. For instance the plan for a Q_T/A_T, i.e. a normal task related question-answer, contains no reparation strategies for missing information to the background system. If the interpreter fails to access the background system due to lack of information, the translator signals this to the DM which creates an instance of an IR-unit for clarification request (say Q_D/A_D) and inserts it into the context of Q_T/A_T. The action plan for clarification request then generates a move explaining the missing information and creates an instance of a user move waiting for user input. This has the advantage that the plans can be made very simple, as they need only be aware of their own local behaviour and context. Further, the plans are more generally applicable; the plan for a clarification request is similar for any level in the dialogue tree.

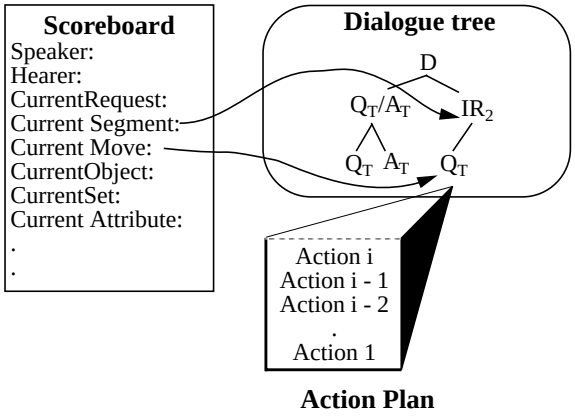

Figure 1. The Internal structures used by the dialogue manager.

3 Customization

Customizing LINLIN means augmenting the various knowledge sources seen to the right in figure 2. LINLIN is not to be regarded as having a completely empty knowledge base. Rather we start with a basic lexicon and grammar and also some dialogue object descriptions that are common to many applications. These knowledge sources constitute a shell which needs to be updated with domain dependent concepts, rules and interaction principles. This is done by a person, or a team of persons, cf. Wroblewski & Rich (1989), with some background in linguistics. I refer to such a person as a *language engineer.*

However, the language engineer is not to be regarded as the only person involved in the customization. The translator, i.e. the module interfacing the background system, needs to be rewritten for each new application. The same translator can, however, be used in different domains using the same application through the means of translation rules for a specific domain, e.g. different SQL databases. There should also be a tool for end user customization, like the ones used by TEAM or LUKE.

One novel aspect of our approach is that the information available to the language engineer does not only consist of the requirements and application descriptions from the customer, but also a corpus collected using Wizard of Oz-experiments with the end-users, see the upper right in figure 2. This information is integrated into the system's knowledge bases using two different tools, a knowledge base tool and a system for acquiring information from a corpus. Adding the information provided by the customer using a knowledge base tool is a well-known, although not in any respect solved, task, and is not elaborated further upon in this paper.

Another aspect of the customization, deals with performance improvement. This can in part be achieved by reducing the size of the knowledge bases. Kelley (1983) developed his interface with no initial knowledge, thus the only concepts were those found during the simulations. This will give a limited, but hopefully sufficient, knowledge base for one application, which was Kelley's goal. However, my purpose is to develop an interface which can be customized to many different applications and thus I would like to reuse as much previous information as possible. One way to achieve this efficiently is to

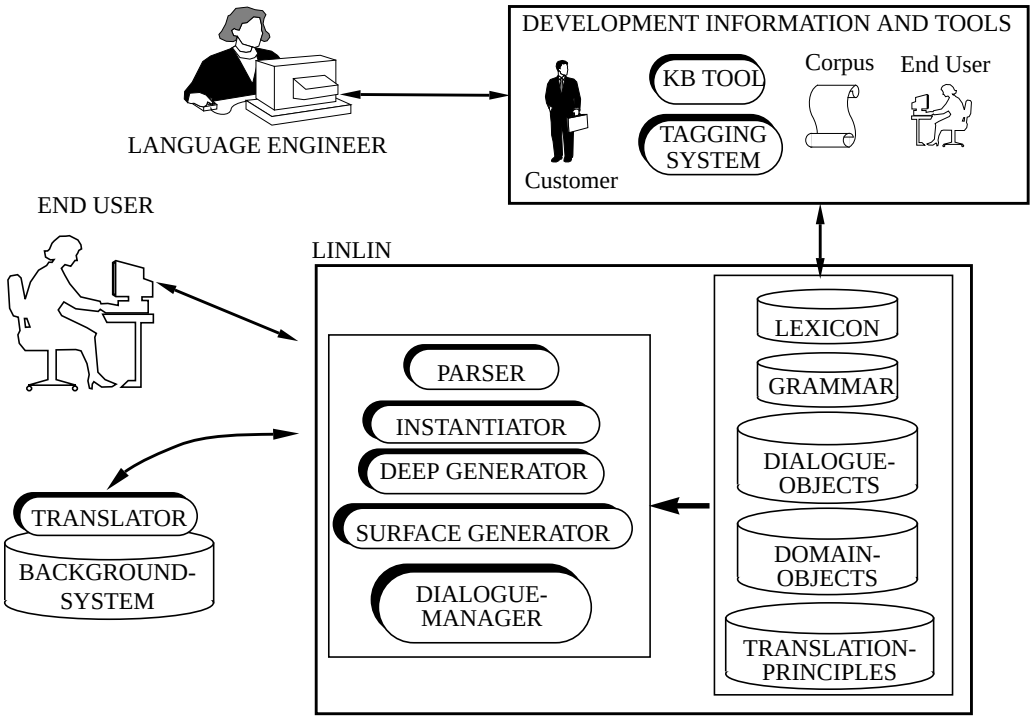

Figure 2. A Natural Language Development Environment

structure the knowledge bases into two levels. One primary level for knowledge that is found to be relevant in the current application, either given by the customer or through the simulations, and another secondary level containing the rest. Thus, the use of simulations can also be used for improving performance. However, we have as yet no knowledge base that is large enough to be considered for such modifications.

3.1 The tagging system

For analysis of the corpus we use an interactive graphical tagging system called DagTag (Ahrenberg & Jönsson, 1989). In DagTag the tag is a directed acyclic graph, a dag, used in the encoding of information. DagTag allows tagging on various levels of description, e.g. lexical, syntactic/semantic and dialogue. Thus, we can use the same tool to encode information for all the different knowledge sources utilized in our interface. First the segment to be tagged, e.g a move, is marked, and then the tagging information is assigned, e.g. Q_T. This process is much speeded up with the use of templates. A template is a ready-made dag of arbitrary complexity and specification. For the most common tags the dag is fully specified, but for the least frequently used it is better to use a general dag which is specified when used. The templates are constructed using a graphical dag-editor.

Further, as the tags have the structure of dags, which is the same format as used in LINLINs knowledge bases, we can extract information from a tagged corpus to be incorporated into LINLINs knowledge bases. We have carried out initial experiments on the acquisition of grammar and lexicon from a tagged corpus (Jönsson & Ahrenberg 1990).

The task at hand now is how to do this for the other knowledge bases, initially the dialogue objects.

3.2 Customizing the dialogue

The use of dialogue objects makes it possible for us to also customize the interaction principles from a tagged corpus. As said above, the dialogue objects consisted of one static part containing contextual information and one part describing a prototypical behaviour. Now, the process part is the same for dialogue objects with the same basic behaviour, c.f. the action plan for a user initiated IR-unit described in section 2.1. The number of such generic dialogue objects is very small. For a database information retrieval application the different types are: D-unit, system initiated IR-unit, user initiated IR-unit, generate move and interpret move. These generic dialogue objects are used to create instances of specific dialogue objects for a certain situation, e.g. a Q_T/A_T. They are constructed as templates in DagTag, i.e. the same tool that is used in the subsequent knowledge acquisition phase.

The dialogue is manually tagged using descriptors for move types and IR-units. Figure 3 shows a simplified dag corresponding to a Q_T/A_T initiated by the system[1]. The information in this dag is used to describe one instance of a dialogue object. The general plan for a system initiated IR-unit is specified as a Q_T/A_T by extracting information from this dag and encode it in the static part of the dialogue object as values to the different attributes. This then provides information to the DM for instance on what type of move that is legal in this situa-

1. This example is meant to illustrate how the information is extracted. The Q/A IR-unit is such a basic IR-unit that it is probably regarded as domain independent.

tion, the value in the Response slot. In this case it says that the user may respond with an A.

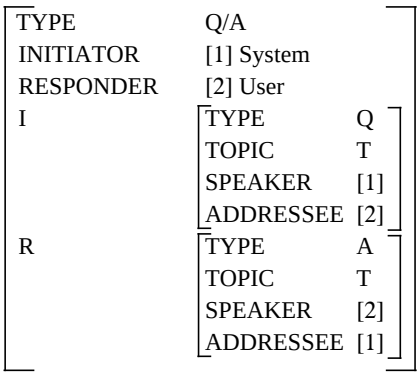

Figure 3. A simplified dag for a Q_T/A_T IR-unit

Suppose we encounter an utterance where the user responds to a Q_T with an AS_T, i.e. we have a Q_T/AS_T as a legal IR-unit. For simplicity we ignore the other properties. This new IR-unit does not constitute a new IR-unit of type Q_T/AS_T instead it corresponds to modifying the previously created Q_T/A_T to something like $Q_T/(A_T \vee AS_T)$. The $(A \vee AS)$ is a value in the Response slot allowing also an AS to be a legal move to a question with topic T.

The dag in figure 3 also contains information about relevant properties for a move-unit, i.e. the values to the attributes I and R. This information can be used for specifying the move dialogue objects by extracting information like type, topic, addressee and speaker.

Further, as the customization is carried out by a language engineer, it is possible to define move and IR units consisting of new action sequences. This can be used for instance to build more complex IR-units like $DI_T/ACK_T/ACK_D$.

It is also possible to customize the referent resolution algorithms used by the NLI. Information about salient objects is represented in the dialogue tree and is used by the instantiator and deep generator through a scoreboard, see figure 1. Associated with the slots on the scoreboard are access functions. The access functions can be altered, allowing the search for a referent to an anaphoric expression to be application dependent.

4 Discussion and current status

We have collected and analysed a number of dialogues. In the corpus most of the utterances are user-initiated task questions provoking a response which is retrieved from the data base. The response is mostly an A_T and in some cases an AS_S (i.e. no information). The number of clarifications is low and, interestingly, when the system has initiated the clarification, the user does not respond to the clarification, instead he starts a new initiative. Further, there are some ending and greeting sequences and one instance where the system/wizard opens a new IR-unit.

I have partially analysed a small part of the corpus consisting of three Wizard of Oz dialogues with the purpose of developing dialogue objects. The application is information retrieval from a data base containing information on properties of used car models. My intention is, however, to investigate the other types of applications that we are dealing with, such as advisory systems and configuration systems.

In the work by Kelley (1983) and Good *et al.* (1984) the customization process was saturated after a certain number of dialogues. We have not analyzed the corpus enough to say whether this will happen to the dialogue objects, too. However, for data base systems there seems to be a limited set of move and IR-unit types, but whether this is true for other applications is an open question.

The current status of LINLIN is that, except for the deep generator, there are pilot versions of the different processes, running with small knowledge bases. There is also a pilot version of the dialogue manager, but it is not yet integrated with the other modules. The tagging system has been used with a very small portion of the corpus for augmenting the lexicon and grammar. Currently we are working on using the tagging system to create the generic dialogue objects and we will also use DagTag to specify the dialogue objects.

5 Acknowledgements

This work is part of a larger project carried out at the Natural Language Processing Laboratory, Linköping, sponsored by the Swedish National Board for Technical Development, STU and Swedish Council for Research in the Humanities and Social Sciences (HSFR). Many of the ideas on dialogue management and customization have evolved during discussions with my colleagues in the lab, especially Lars Ahrenberg and Nils Dahlbäck.

Åke Thurée did most of the coding for the DM in Xerox Common Lisp on a Sun Sparc Station. Ivan Rankin, Mats Wirén and Richard Hirsch have read previous versions of the paper and provided many valuable comments.

References

Ahrenberg, L. (1989). On the integration of linguistic knowledge and world knowledge in natural language understanding. In Ö. Dahl & K. Fraurud (eds.) *Papers from the First Nordic Conference on Text Comprehension in Man and Machine*, Institute of Linguistics, University of Stockholm, pp. 1-11.

Ahrenberg, L. and Jönsson, A. (1989) An interactive system for tagging dialogues. *Literary and Linguistic Computing* **3**(2), 66-70.

Ahrenberg, L., Jönsson, A. & Dahlbäck, N. (1990) Discourse Representation and Discourse Management for a Natural Language Dialogue System, *To appear in Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine*, Täby, Stockholm.

Ballard, B. W. & Stumberger, D. E. (1986) Semantic Acquisition in TELI: A Transportable, User-Customized Natural Language Processor, *Proceedings of the 24th Annual Meeting of the ACL*, New York.

Carberry, S. (1990) *Plan Recognition in Natural Language Dialogue*, MIT Press.

Chapanis, A. (1982) Appendix: Man-Computer Research at Johns Hopkins, *Information Technology and Psychology*, Kasschau, Lachman & Laugherty (Eds), Raeger Publishers.

Dahlbäck, N. & Jönsson, A. (1989) Empirical Studies of Discourse Representations for Natural Language Inter-

faces, *Proceedings of the Fourth Conference of the European Chapter of the ACL*, Manchester. 1989.

Good, M. D., Whiteside, J. A., Wixon, D. R. & Jones, S. J., (1984) Building a User-Derived Interface, *Communications of the ACM*, Vol. 27, No 10, pp 1032-1043.

Grishman, R. & Kittredge, R. (eds.) 1986. *Analysing language in restricted domains*. Hillsdale, N.J.: Erlbaum.

Hirst, G. (1987). *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge University Press.

Hoeppner, W., Christaller, T., Marburger, H., Morik, K., Nebel, B., O'Leary, M. & Wahlster, W. (1983) Beyond Domain Experience: Experience with the Development of a German Language Access System To Highly Diverse Background Systems, Research Report, University of Hamburg, Bericht ANS-16.

Jönsson, A. (1991) A Dialogue Manager Using Initiative-Response Units and Distributed Control, *Proceedings of the 5th Conference of the European Chapter of the ACL*, Berlin, Germany.

Jönsson, A. (1990) Application-Dependent Discourse Management for Natural Language Interfaces: An Empirical Investigation, *Papers from the Seventh Scandinavian Conference of Computational Linguistics*, Reykjavik, Island.

Jönsson, A. & Ahrenberg, L. (1990) Extensions of a descriptor-based tagging system into a tool for the generation of unification-based grammars. To appear in *Research in Humanities Computing* **1**.

Jönsson, A. & Dahlbäck, N. (1988) Talking to a Computer is not Like Talking to Your Best Friend. *Proceedings of The first Scandinivian Conference on Artificial Intelligence*, Tromsø, Norway.

Kelley, J. F. (1983) Natural Language and Computers: Six Empirical Steps for Writing an Easy-to-Use Computer Application, PhD thesis, The Johns Hopkins University.

Levinson, S. C. (1981) Some Pre-Observations on the Modelling of Dialogue, *Discourse Processes*, No 4, pp 93-116.

Martin, P., Appelt, D. E., Grosz, B. J. & Periera, F. (1985) TEAM: An Experimental Transportable Natural-Language Interface, *IEEE quarterly bulletin on Database Engineering*, Vol. 8, No. 3.

Reichman, R. (1985) *Getting Computers to Talk Like You and Me*, MIT Press, Cambridge, MA.

Wroblewski, D. A. and Rich, E. A. (1989) LUKE: An Experiment in the Early Integration of Natural Language Processing. *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas pp. 186-191.