A DIALOGUE MANAGER USING INITIATIVE-RESPONSE UNITS AND DISTRIBUTED CONTROL

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

Abstract

This paper describes a system for managing dialogue in a natural language interface. The proposed approach uses a dialogue manager as the overall control mechanism. The dialogue manager accesses domain independent resources for interpretation, generation and background system access. It also uses information from domain dependent knowledge sources, which are customized for various applications.

Instead of using complex plan-based reasoning, the dialogue manager uses information about possible interaction structures and information from the specific dialogue situation to manage the dialogue. This is motivated from the analysis of a series of experiments where users interacted with a simulated natural language interface. The dialogue manager integrates information about segment types and moves into a hierarchical dialogue tree. The dialogue tree is accessed through a scoreboard which uses exchangeable access functions. The control is distributed and the dialogue is directed from action plans in the nodes in the dialogue tree.

1 Introduction

To achieve true cooperation a natural language interface must be able to participate in a coherent dialogue with the user. A common, generally applicable approach is to use plan-inference as a basis for reasoning about intentions of the user as proposed by, for instance, Allen & Perrault (1980), Litman (1986), Carberry (1989) and Pollack (1986). However, computationally these approaches are not so efficient.

Reichman (1985) describes a discourse grammar based on the assumption that a conversation can be described using conventionalized discourse rules. Gilbert, Buckland, Frolich, Jirotka & Luff (1990) uses interaction rules in their menu-based advisory system. Our approach is similar to Reichman and Gilbert *et al.* In a series of experiments (Dahlbäck & Jönsson, 1989, Jönsson & Dahlbäck, 1988) we studied dialogue behaviour in an information-seeking interaction between a human and a computer using a simulated natural language interface (NLI). One important result was that the users followed a rather straightforward information searching strategy which could be well described using conventionalized rules.

Reichman uses surface linguistic phenomena for recognizing how the speaker's structure the discourse. We found, however, very little use of surface linguistic cues in our dialogues. In our corpus users normally initiate a request for information, which is followed by an answer from the system. Sometimes the request needs clarification before the answer can be given as a response to the initial question (this is illustrated in section 4 and 5). Optionally the user can interrupt the original question and start a new initiative–response unit, but this also follows the goals of information-seeking. Thus, we adopt a strategy in which we employ the notion of adjacency pairs (Schegloff & Sacks, 1973, see also Levinson, 1983: 303f). In our approach the dialogue is planned and utterances are interpreted in terms of speech acts. The speech acts are determined on the basis of structural information in the utterance and in the immediate context.

Further, we found, in our experiments, that different configurations of the background system (e.g. data base, consultation) and task to solve (e.g. information retrieval, configuration) require different mechanisms for handling dialogue in an NLI (Jönsson, 1990). Therefore, one major design criterion is that the system should be easy to adapt (customize) to a new application.

The natural language interface described in this paper is constructed on the assumption that different applications have different sublanguages (Grishman & Kittredge, 1987), i.e. subsets of a natural language. A sublanguage is not only defined by a grammar and lexicon, but also by interaction behaviour, i.e factors such as how the user and system handle clarifications, who takes the initiative, what is cooperative in a certain application, what are the user categories and so on.

The dialogue manager operates as the central controller in the NLI (Ahrenberg, Dahlbäck & Jönsson, 1990). It passes information encoded in directed acyclic graphs (dags) between different modules for parsing, generation, etc. This paper, however, only describes the dialogue manager's role in the control of the dialogue. I assume that the dag's correctly describe the full meaning of the user's input. For a discussion of interpretation of user input in this system see Ahrenberg (1988). The dialogue manager is implemented in CommonLisp but is currently not completely integrated with the other modules of the system.

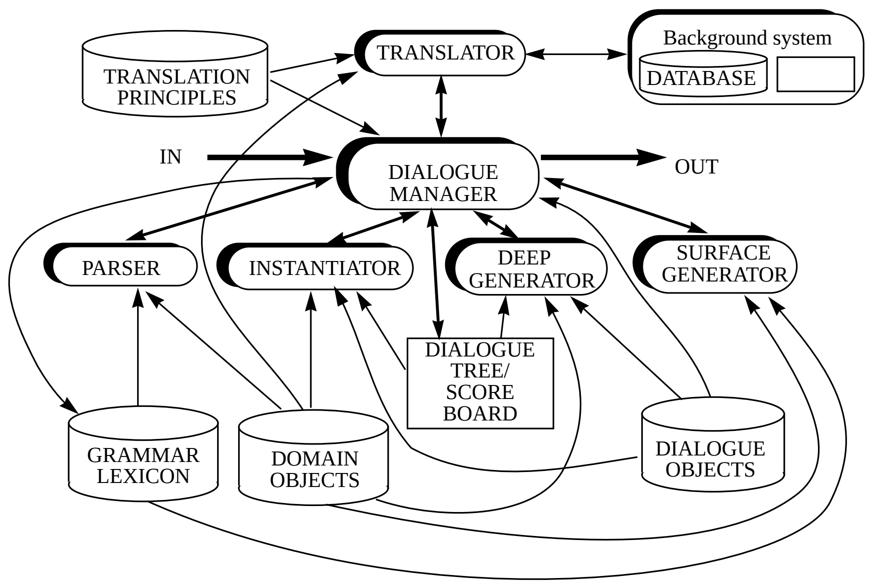

Figure 1. Overview of the architecture

2 The dialogue manager

The dialogue manager (DM) is the kernel in the natural language interface, see figure 1. It directs the dialogue, assists the instantiator and deep generator and communicates with the background system. DM can be viewed as a controller of *resources* and *knowledge sources*.

The resources in our system are a chart parser (Wirén, 1988), an instantiator which links the linguistic object descriptions to objects in the universe of discourse (Ahrenberg, 1989), a translator which translates the instantiated structures into a form suitable for accessing the background system[1] and finally a deep and a surface generator for generating a system utterance. These resources are domain independent processes accessing various knowledge sources.

The knowledge sources are domain dependent and implemented in the same knowledge base system and can be modified for each new application. We use a lexicon for general and domain-specific vocabulary and a grammar with knowledge of syntactic constructions and their semantic impact. Furthermore, we use descriptions of **dialogue objects**, i.e. segments and moves and their associated information (section 3) and domain object descriptions which contain relations between the concepts used to describe objects in the background system and constraints on them.

The need for domain object information in a natural language database interface has been argued for by for instance Copestake & Sparck Jones (1990) and McCoy & Cheng (1988). The domain objects are primarily used by the instantiator and deep generator, but the translator, parser and surface generator can also use this information. For a discussion on domain objects in this system see Ahrenberg, Jönsson & Dahlbäck (1990).

Each input or output from the resources passes via the dialogue manager (DM). A typical segment begins with an input from the user that is sent to the DM which passes it to the parser. The parser sends its result to the DM which passes it to the instantiator where it is enhanced with referential information. This is sent to the translator which accesses the background system and if the access succeeds, informs the DM. The DM forwards the information to the deep generator where an enhanced description is created which is sent to the surface generator and finally a response is given from the DM to the user. This has the advantage that the DM always has control over what happens in the system. Thus, if one module does not succeed with its task, the DM directs the recovery. For instance, if the translator cannot access the data base due to lack of information from the user, the DM receives information from the translator that there is information missing and then in turn calls the deep and surface generators to produce a suitable message to the user. The DM then waits for input to provide to the parser and instantiator. Finally, the DM tries to integrate the new information with the previous information.

Internally the dialogue manager maintains three dynamic structures for monitoring the dialogue: the **dialogue tree** (section 4) where the dialogue history is kept, **action plans** (section 5) for controlling the dialogue and finally a **scoreboard** (section 6) which constitutes the interface between the dialogue tree and other modules of the system.

3 Dialogue objects

Dialogue objects play a central role in this architecture. The dialogue objects consist of two components, one is a process description of a prototypical use of the dialogue object. This is described below. The other contains static information about speaker, hearer, type, topic, context and different types of descriptors to describe salient objects, e.g. the focused object, potential focused objects and optionally the current set. The current set records which subset of the data base that is currently used. We found in our data base dialogues that the user often restricts the possible candidates in a database

1. Initially we use only a relational database system.

search. For an example consider the dialogue fragment in example 1[1].

S7> Most of the cars consume less than 1 litre/10km, be more specific
U8> 0.7 litre/10km
S9> Daihatsu Charade   Fiat Uno
Ford Fiesta   Honda Civic
Lancia Y10/A112   Mazda 323
Mitsubishi Colt   Nissan Micra
Nissan Cherry   Opel Corsa
Peugeot 205   Opel Kadett
Renault 5   Toyota Starlet
Toyota Corolla   Volkswagen Polo
U10>Which five consume least? Rank them
S11>Daihatsu Charade
Fiat Uno
Nissan Micra
Toyota Starlet
Honda Civic
U12>Which of these is safest
S13>They are all equal except Nissan Micra which is the worst
U14>Which one is easiest to handle?
S15>No information

Example 1

First the user specifies a set of cars in utterance U8>, presented by the system in S9>. This set is however, too large, therefore in utterance U10> it is reduced. In the sequence of utterances U12> to S15>, Current set consists of the cars presented in U11>. Current set does not have to be explicit as in example 1, instead it can be described by constraints. For instance, in a travel data base the user may be interested in a trip to Greek islands which restricts the search in the database to Greek islands for a large part of the ensuing dialogue.

The communication is hierarchically structured using three different categories of dialogue objects. There are various proposals as to the number of levels needed. The system developed by Polanyi & Scha (1984) uses five different levels to hierarchically structure a dialogue and LOKI (Wachtel, 1986) uses four. In LOKI the levels are: conversation, dialogue, exchange and move. When analysing our dialogues we found no certain criteria concerning how to divide a dialogue into a set of exchanges. Therefore we only use three different dialogue object types: dialogue, initiative-response-unit (IR) and move. Dialogue, in our notation, is similar to conversation in LOKI, while IR-units resemble exchanges. IR-units are recursive and, unlike LOKI, we allow arbitrary embedding of IR-units.

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 is used for describing information about a move. Moves are categorized according to the type of illocutionary act and topic. Some typical move types are: Question (Q), Assertion (AS), Answer (A) and Directive (DI). Topic describes which knowledge source to consult: the background system, i.e. solving a task (T), the ongoing dialogue (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 an exchange of information begins 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 database 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 a clarification request from the user, Q_T/AS_S when the requested information is not in the database, Q_D/A_D for questions about the ongoing dialogue.

The dialogue manager uses a dialogue tree (section 4) as control structure. The root node is of type Dialogue (the D-node) and controls the overall interaction. When an IR-unit is finished it returns control to the D-node. The D-node creates an instance of a new IR-unit with information about initiator and responder. It also copies relevant information about salient objects and attributes from the previous IR-unit to the new one. Our simulations show that users prefer coherence in the dialogue. Thus, we use the heuristic that no information explicitly changed is duplicated from one IR-unit to the next.

As stated above, an instance of a dialogue object has one component describing static information about initiator, responder, salient objects etc., and another describing the process, i.e. the actions performed when executing the object. We call this a plan, although if we were to follow Pollack (1990) we could call it recipe-for-actions. Figure 2 shows a template description for an IR-unit used in a database information-seeking application.

```
Class: IR
Topic: T ∨ D ∨ S
Context: link to father
Initiator: System ∨ User
Responder: System ∨ User
Type: type of IR-unit e.g. Q/A
Initiative type: e.g. Q
Response type: e.g. A
Turns: list of daughter nodes
CurrentObject:
CurrentAttribute:
CurrentSet:
CurrentRequest:
[Initiator = User]   →  [Process: ((create-move user)
                                   (access)
                                   (create-move system)
                                   (up))                ]
[Initiator = System] →  [Process: ((create-move system)
                                   (create-move user)
                                   (up))                ]
```

Figure 2. A template description for IR-units

The static component forms the context in which the processes are executed. The attributes are updated with new values during the execution of the action plan. For instance, a user IR-unit, i.e. an IR-unit which waits for a user initiative to be interpreted, has no value for the Initiative and Response slots until the initiative has been interpreted. This is discussed further in section 4.

The process component of the IR-unit is divided into two different plan descriptions, one if the system initiated the segment and another for a user-initiated segment.

1. The dialogue is an English translation of a dialogue from our corpus of Swedish dialogues collected in Wizard-of-Oz simulations. It is continued in section 4.

However, as can be seen in figure 2, they use the same general actions for creating moves, acting and traversing the tree (up). The actions behave differently depending on the static description, for instance the action (access) uses the value of the slot Topic to determine which knowledge source to consult. Information about values of attributes describing the request for information is found in the dag structure delivered by the instantiator which is passed to the translator by the dialogue manager. The slot CurrentRequest contains the request formed by the translator and is used for clarifications.

In database applications the system behaves as a user-directed interface. It initiates an IR-unit only for clarification requests, either because 1) difficulties arise when interpreting the utterance, or 2) difficulties arise when accessing the data base, e.g. when the user needs to provide a parameter for correct access, see S17> in example 2 below, or finally 3) if difficulties arise in the presentation of the result from the data base access. The action to take after a clarification request is first to check the validity of the response and then to propagate the information to the node which initiated the clarification.

In other applications, e.g. tutoring or consultation systems, the behaviour need not be user-directed. Instead it may be system-directed or mixed initiative. In our approach this is achieved by customizing the dialogue objects, section 7.

For move-units there are two different process descriptions, one for user moves and one for system moves. The user move has the plan ((parse) (instantiate) (up)) and the system move has the plan ((deep-generate) (surface-generate) (up)).

4 The dialogue tree

The dialogue tree represents the dialogue as it develops in the interaction. Information about salient objects is represented in the dialogue tree and is used by the instantiator and deep generator. The dialogue manager updates the dialogue tree for each new move.

An important feature of the dialogue manager is distributed control. Every node in the tree is responsible for its own correctness. For instance, the plan for a task related question-answer, Q_T/A_T, contains no reparation strategies for missing information to the background system. If the interpreter fails to access the data base due to lack of information, the translator signals this to the DM which creates an instance of an IR-unit for a clarification request and inserts it into the Q_T/A_T. The plan for clarification request then generates a move explaining the missing information and creates a user move waiting for the user input. This has the advantage that the plans are very simple, as they only have local scope, cf. sections 3 and 6. Furthermore, the plans are more generally applicable.

U16>	I would like a car with a large boot
S17>	How big (litres)?
U18>	I don't know
S19>	They vary in size from about 200-350 litres
U20>	I want at least 300 litres.
S21>	BMW 318/320

Example 2

The tree is built bottom up but with a top down prediction from the context. This is illustrated in the dialogue in example 2, which will generate a dialogue tree with clarifications on two levels. Initially the D-node creates an instance of an IR-node and inserts it into the tree, i.e. creates links between the IR-node and the D-node. The IR-node creates an instance of a user move. The move node parses and instantiates U16> successfully as an AS_T and then integrates it into the tree. Information from the move-node is then available also at the IR-node whose type can be determined as AS_T/A_T. When the database is accessed from this node, the translator finds that there is a need for clarification, in this case concerning the use of the word *large* in connection with a boot. This creates a plan which first prompts the user with a question, S17>, and then waits for the user to give an answer. Here the user does not answer but instead expresses a request for clarification, U18>. This is shown in part 1) of figure 3 as the clarification IR-unit, QS_S/A_S. The fact that U18> constitutes a clarification request and not an answer to S17> is decided after the creation of the user move from U18>. When the DM receives the interpretation from the instantiator, it does not satisfy the expectation for an answer, and so it has to instantiate a new IR-unit for clarification request which is connected to the previously created IR-clarification request (Q_T/A_T).

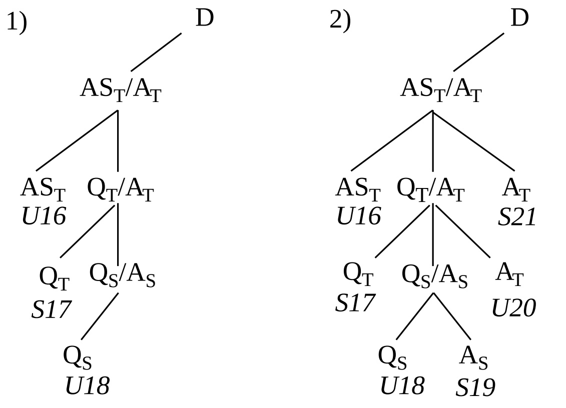

Figure 3. A dialogue tree

Utterance U18> in the context of the Q_T/A_T IR-unit indicates that the user needs some information about the background system and it is thus interpreted as Q_S. This information is supplied in S19>. For the next utterance, U20>, a new user move is created which is integrated into the tree as an answer to the original clarification request. This information is propagated up to the first node AS_T/A_T which now can form an answer to the first question S21>, part 2) in figure 3. The next step (not shown in figure 3) is to generate a new IR-unit under D which will generate a new user move and the system is ready for further user input.

5 The action plan

The plan describing a prototypical use of an object is pushed onto a stack called the action plan. In accordance with our distributed design, each node maintains its own stack, see figure 5. The overall control strategy is that the stack top is popped and executed. Complex plans, as when the query to the data base needs clarification, are handled with the same control mechanism. The dialogue manager then updates the action plan of the current node with an action for creating an instance of a clarification request dialogue object and another action to integrate new information. The DM pops the stack of the current node and executes that action. When this new exchange is completed the result is integrated into the node which initiated the clarification.

Again, consider the dialogue tree in figure 3. Part 1) in figure 4 shows the stack for the node AS_T/A_T before processing U16>, i.e. before the move node is created which parses and instantiates the move. At this time the node type is not known.

1)
```
(create-move user)
(access)
(create-move system)
(up)
```

2)
```
(create-IR QT/AT)
(integrate-new-info)
(access)
(create-move system)
(up)
```

Figure 4. The action plan for an IR-node

Popping the action (create-move user) results in the creation of a move node which is ready to interpret a user input. The move node has a plan of its own: ((parse) (instantiate) (up)). When U16> is interpreted in the move node, AS_T in figure 3, the move node ends with the action (up) which tries to find a corresponding father. In this case it succeeds with the IR-unit from which the move node was created and the dialogue is controlled from this node, now AS_T/A_T. The stack top is now (access) which in this case uses the topic T, i.e. a data base access. However, the data base access does not succeed. Therefore a call for clarification, an action for later integrating the new information into the old request and a new call to (access) is placed on the stack. This is seen in part 2) of figure 4. The action (access) has different repair strategies for the different clarification request types described above. Similar repair strategies apply to all actions.

The stack top is an action which creates a known IR-unit asking for a data base access parameter. This action then creates the Q_T/A_T-node in figure 3. Now this node will have its own action plan stack from which processing is controlled. This node is also responsible for the correctness of the answer given from the user, which in this case results in a new clarification request. This does not affect the node AS_T/A_T instead the clarifications are processed and eventually control is returned to the node AS_T/A_T and the new information is integrated into its old request, stored in CurrentRequest.

The two clarification nodes, Q_T/A_T, Q_S/A_S, in figure 3 behave in a similar fashion.

6 Scoreboard

Controlling the dialogue is only one of the responsibilities of the dialogue manager. It is also responsible for monitoring the dialogue. Information about salient objects is represented in the dialogue tree and is accessed through a scoreboard, figure 5. The scoreboard is the interface between the dialogue manager and the other modules in the NLI.

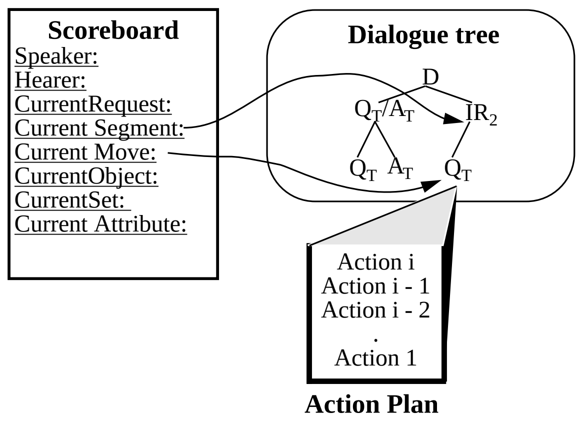

Figure 5. The internal structures used by DM

The attributes of the scoreboard take their values from the tree via pointers or via retrieve functions which search the dialogue tree. The lexicon and grammar are written with references to the attributes on the scoreboard and therefore are not involved in traversing the dialogue tree.

Furthermore, the retrieve functions can be altered, allowing the search for a referent to an anaphoric expression to be application dependent. This means that we need only update the retrieve function connected to an element on the scoreboard, not the grammar or lexicon, when an application requires a change in dialogue style.

7 Customization

One objective of this project is to develop a natural language interface that can be customized to different applications, i.e. a natural language interface shell to be used by a language engineer when creating an NLI for a specific application.

Customization is achieved by using different exchangeable/modifiable knowledge sources. Our intention is to build a library of prototypical knowledge sources and re-use much of the knowledge between different applications. For instance the lexicon for an SQL data base interface needs to be updated with data base content-specific terms but large parts of it are re-usable. Furthermore, we believe this to be possible not only for the lexicon and grammar, but also for the dialogue objects. The plans for a data base system will be much the same regardless of the domain. Customization, however, is not the topic of this paper. For more on this see Jönsson (1991).

8 Summary

I have presented an architecture for dialogue management for natural language interfaces to various applications. The dialogue manager operates as a controller of resources for parsing, instantiation, generation and database access.

The design of the dialogue manager is based on the analysis of a corpus of simulated human-computer interactions. Unlike plan-based proposals which employ user intentions to guide the interaction, the dialogue manager described here uses plans with information about prototypical interaction patterns. The plans are modelled in dialogue objects which also contain static information for representing the dialogue.

The dialogue objects are hierarchically structured in three categories: dialogue, initiative-response and move. The initiative-response category is recursive. Use of an initiative-response structure can be criticised in the same way as adjacency pairs for not adequately describing a naturally occurring discourse. However, for a restricted sublanguage, such as natural language communication with computers, we believe that this is a very efficient way of managing the dialogue (cf. Levinson 1981: 114).

The dialogue history is represented in a dialogue tree consisting of instantiated dialogue objects. The resources access the dialogue tree through a scoreboard and thus need no mechanisms for traversing the tree.

We have conducted experiments which show that in an information-seeking human-computer dialogue the proposed mechanisms can correctly handle the dialogue. Empirical tests will show how many different interaction settings we can handle.

Acknowledgements

This work is much inspired by the work that I have done with Nils Dahlbäck and Lars Ahrenberg. Åke Thurée did most of the coding for the DM in Xerox Common Lisp on a Sun Sparc Station and many ideas were formed during discussions with him. Lars Ahrenberg, Ivan Rankin, Mats Wirén and Richard Hirsch have read previous versions of the paper and provided many valuable comments.

References

Ahrenberg, Lars (1988) An Object-Oriented Dialogue System for Swedish, *Nordic Journal of Linguistics*, Vol. 11, Nos 1-2, pp 3-16

Ahrenberg, Lars (1989) A Constraint-Based Model for Natural-Language Understanding and a Pilot Implementation. Research Report LiTH-IDA-R-89-22, Department of Computer and Information Science, Linköping University.

Ahrenberg, Lars, Arne Jönsson & Nils Dahlbäck (1990) Discourse Representation and Discourse Management for Natural Language Interfaces, *To appear in Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine*, Täby, Stockholm.

Allen, James. F. & C. Raymond Perrault (1980) Analysing Intention in Utterances, *Artificial Intelligence,* 15, pp 143-178.

Carberry, Sandra (1989) A Pragmatics-Based Approach to Ellipsis Resolution, *Computational Linguistics*, Vol. 15, No 2. pp 75-96.

Copestake, Ann & Karen Sparck Jones (1990) Natural Language Interfaces to Databases, Technical Report No. 187, University of Cambridge, UK

Dahlbäck, Nils & Arne Jönsson (1989) Empirical Studies of Discourse Representations for Natural Language Interfaces, *Proceedings of the Fourth Conference of the European Chapter of the ACL,* Manchester. 1989.

Gilbert, Nigel, Sarah Buckland, David Frolich, Marina Jirotka & Paul Luff, Providing Advice Through Dialogue, (1990) *Proceedings of ECAI-90,* Stockholm.

Grishman, R. & Kittredge, R. (Eds.) 1986. *Analysing language in restricted domains*. Lawrence Erlbaum.

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

Jönsson, Arne (1991) A Natural Language Shell and Tools for Customizing the Dialogue in Natural Language Interfaces. Internal Report, LiTH-IDA-R-91-10.

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

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

Levinson, Stephen C. (1983) *Pragmatics*. Cambridge University Press.

Linell, Per, Lennart Gustavsson & Päivi Juvonen (1988) Interactional Dominance in Dyadic Communication. A presentation of the Initiative-Response Analysis. *Linguistics,* 26(3).

Litman, Diane J. (1986) Understanding Plan Ellipsis, *Proceedings of AAAI-86*.

McCoy, Kathleen F. & Jeannette Cheng (1988) Focus of Attention: Constraining What Can Be Said Next, *Presented at the 4th International Workshop on Natural Language Generation.*Buffalo.

Polanyi, Livia & Remko Scha (1984) A Syntactic Approach to Discourse Semantics, *Proceedings of COLING'84,* Stanford.

Pollack, Martha E. (1986) A Model of Plan Inference that Distinguishes between the Beliefs of Actors and Observers, *Proceedings of the 24th Annual Meeting of the ACL*, New York.

Pollack, Martha E. (1990) Plans as Complex Mental Attitudes, *Intentions in Communication,* MITPress, 1990.

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

Schegloff, Emanuel, A. & Harvey Sacks (1973) Opening up closings, *Semiotica,* **7**, pp 289-327.

Wachtel, Tom (1986) Pragmatic sensitivity in NL interfaces and the structure of conversations, *Proceedings of COLING'86*. Bonn.

Wirén, Mats (1988) On Control Strategies and Incrementality in Unification-Based Chart Parsing, Licentiate thesis, Thesis No 140, Department of Computer and Information Science, Linköping University