Language-Processing Strategies and Mixed-Initiative Dialogues

Johan Boye[2], Mats Wirén[2], Manny Rayner[1], Ian Lewin[1], David Carter[1] and Ralph Becket[1]

[1]SRI International
Suite 23, Millers Yard
Cambridge CB2 1RQ, UK

[2]Telia Research
Spoken Language Processing
S-123 86 Farsta, Sweden

Abstract

We describe an implemented spoken-language dialogue system for a travel-planning domain, which accesses a commercially available travel-information web-server and supports a flexible mixed-initiative dialogue strategy. We argue, based on data from initial Wizard-of-Oz experiments, that mixed-initiative strategies are appropriate for many types of user, but require more sophisticated architectures for processing of language and dialogue; we then use these observations to motivate an architecture which combines parallel deep and shallow natural language analysis engines and an agenda-driven dialogue manager. We outline the top-level processing strategy used by the dialogue manager, and also a novel formalism, which we call Flat Utterance Description, that allows us to reduce the output of the deep and shallow language-processing engines to a common representation.

1 Introduction

Travel-planning domains have been a common application area for spoken-language dialogue systems almost from their inception, both as pure research vehicles and now, with maturing speech technology, as fielded prototypes. Fielded systems naturally tend to employ simpler linguistic and dialogue processing. Domain-specific keyword/phrase spotting and slot-filling techniques are preferred for utterance interpretation. At the dialogue level, systems tend to keep the dialogue initiative to themselves by treating the user simply as an answer-supplier. Particular systems may also implement particular instances of more sophisticated processing. However, the simple methods do dovetail simply because the more expectations that a system can impose on a dialogue, then the more those expectations can be used to aid interpretation of user utterances. (For a range of recent work, see [Aust and Oerder, 1995], [Allen *et al.*, 1996], [Lamel *et al.*, 1998], [Litman *et al.*, 1998] and [Bos *et al.*, 1999].)

In the work described here, we are primarily interested in exploring relaxation of the constraint that dialogues be system-driven *together* with the use of both sophisticated (but sometimes brittle) and simple (but generally robust) linguistic processing. We hypothesize that different techniques may be applicable at different points in a dialogue. The specific scenario used was that of booking a business trip within Sweden, using air travel or train, and accessing information about times, destinations and fares. Communication in both directions was entirely in spoken Swedish. The underlying database was the Travellink^{TM} system, accessible at `http://www.travellink.se`.[1]

Prior to designing the system, we collected a corpus of data through a Wizard-of-Oz experiment, obtaining altogether 131 dialogues from 47 subjects (31 male and 16 female); the Wizard's conversational style was purposely chosen so as to permit mixed-initiative user strategies. Analysis of the data showed that it displayed significant variation. For example, with respect to verboseness, there is a range of behaviour stretching from consistent use of short, telegraphic-style utterances to very long, disfluent utterances. Furthermore, there are both inactive users who refrain completely from taking the initiative (in effect leaving it open to the system to cross-examine them) and active users who quickly take the initiative by means of counter-questions, keeping it more or less throughout the dialogue. There is also a range of users whose behaviours fall between these extremes. One of our immediate conclusions was that if mixed-initiative dialogues were supported, then a large proportion of the people interacting with the system would make use of this capability.

Typically, we found that the structure of a dialogue about (a leg of) a trip could be subdivided into two phases. First, there is a *specification* phase, in which the user, possibly in response to system prompting, gave the basic constraints on the trip they were looking for: where they were going to, where they were coming from, the date, and some information about the desired departure or arrival time. We regarded the specification phase as terminated when the system had collected enough information that it could access the database and suggest a possible specific trip. After this, there is a second

[1]We would like to thank SMART for help in making the Travellink^{TM} system available to us.

negotiation phase, in which the user may request additional information about the initially suggested trip, ask for alternative trips, and eventually make a booking. The balance between the two phases displayed considerable variation. For the most active users, the negotiation phase dominated: it sometimes started even before the system had suggested any alternative and could persist more or less throughout the dialogue. In contrast, the negotiation phase could be non-existent in the case of the least active users.

In general, we found that analysis of utterances during the negotiation phase required a higher degree of linguistic sophistication than during the specification phase. For example, it was often necessary to be able to understand expressions referring to objects previously mentioned in the dialogue ("that flight", "the first flight"), or distinguish between questions expecting a yes/no response ("Is that a direct flight?") and questions expecting a new object response ("Is there a direct flight?").[2]

The above characteristics of the data and domain prompted us to focus on the following aspects in the design of the system:

- Ability to handle context-dependent, mixed-initiative dialogues in order to cover both kinds of phases in the dialogue as well as the range of active/inactive users.
- Ability to do linguistic analysis deeper than surface slot-filling, so as to be able to distinguish between different forms of utterances critical to the domain.
- Robustness to be able to advance the dialogue even in the case of complex, disfluent utterances and errors likely to be introduced by the speech recognizer.

To meet these desiderata, we have taken an approach with the following distinguishing characteristics:

- Linguistic analysis is factored into context-independent and context-dependent processing phases. The initial context-independent phase produces a set of descriptions based on the explicit form of the input utterance; the descriptions are then interpreted in the relevant context by the dialogue manager.
- The local exchange of initiatives and responses is guided by domain-dependent moves and games, whereas the global goals are handled using an agenda.
- To tackle deep linguistic analysis as well as robustness successfully, and to try to cover different phases of the dialogue equally well, we augment the slot-filling processing method with a more sophisticated grammar-based method. The two parsing engines are run in parallel, and feed independently into the dialogue manager.

[2]Since the focus of the paper is on discourse-level phenomena, we have throughout translated surface linguistic expressions from Swedish to English as a concession to non-Swedish readers.

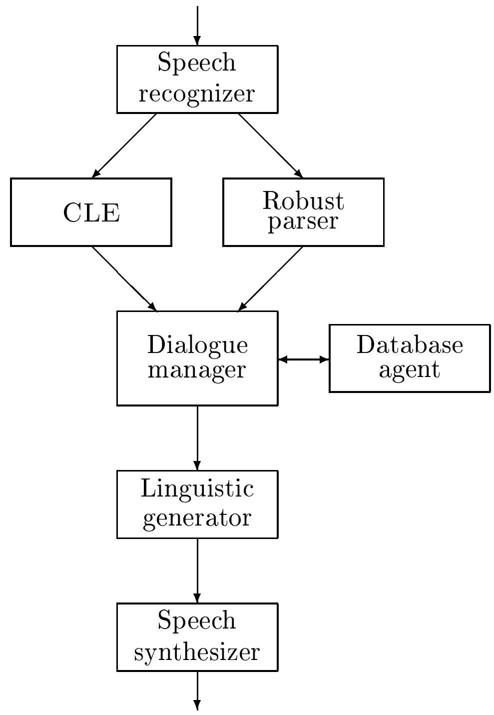

Figure 1: Architecture of the system.

2 System Overview

The architecture of the system is shown in Figure 1. The modules communicate asynchronously by message passing; hence, in principle all of them could run in parallel in different processes. In the current implementation, there are four processes, which handle speech recognition, speech synthesis, database access and everything else, respectively.

The speech recognizer is a Swedish-language version of the SRI Decipher system [Murveit *et al.*, 1993], developed by SRI International and Telia Research. It sends an N-best speech hypothesis list to the two language processors: the Core Language Engine (deep analysis) and the Robust Parser (shallow analysis), further described in Section 3. The language processors each send their analyses to the dialogue manager (DM). After each system turn, the DM updates the language processors with limited information about the state of the discourse: the most recent question (if any) posed by the system, and the types of objects that are salient at the current point in the dialogue.

The DM uses a two-stage heuristic selection process to advance the dialogue. First, each input analysis is categorized as a move of a certain type, and an appropriate response to that move is selected. References are resolved and contextual information is also added, resulting in a further multiplication of possible moves and responses. Secondly, the relative utility of the various responses is judged, and the most productive response move is chosen. The dialogue manager is further described in Section 4.

The generator produces the surface string representing the actual utterance, using a simple template-based approach. The surface string is then turned into speech by Telia Research's synthesizer LIPHON.

In the current system, the database agent contains a web client in order to retrieve data from the Travellink database. All query results are cached in order to shorten the response times as much as possible. However, the response times for most queries would clearly not be acceptable in a commercial system. That inspired us to develop a version that is able to continue the dialogue while database access is in progress (that is, the system might ask about the return leg of a trip, while the database agent is searching for possible trains or flights for the outbound leg).

The system described here is fully implemented and has been permanently installed at the Telia Vision Center in Farsta/Stockholm since November 1998.

3 Language Analysis

3.1 Flat Utterance Descriptions

As previously noted, the system combines two different language processing architectures. Shallow processing is performed by the slot-filling Robust Parser described in Section 3.2 below; deep processing by the SRI Core Language Engine (CLE; [Alshawi, 1992]). Linguistic output can be either *propositional* or *non-propositional*. Non-propositional output consists of markers which are directly linked to dialogue moves; the most important examples are confirmations ("yes", "sure", "that's fine"), rejections ("no", "I'd rather not") and topic shifts ("then..."). Propositional output consists of structured expressions which make reference to world objects like flights, trains, dates, times and costs.

The propositional representations produced by the Robust Parser are lists of slot–filler pairs; those produced by the CLE are expressions in a conservatively extended first-order logic. To allow the DM easily to compare the results produced by the two language processors, it is highly desirable that they be mapped into a common form: the challenge is to find a level of representation which represents an adequate compromise between them. With regard to the CLE, the important point is that most logical forms in practice consist of one or two existentially quantified conjunctions, wrapped up inside one of a small number of fixed quantificational patterns. By defining these patterns explicitly, we can "flatten" our logical forms into a format, which we call a Flat Utterance Description or FUD, that is compatible with a slot–filler list.

The different quantificational wrappers were suggested by our Wizard-of-Oz data; it proved meaningful to distinguish between four kinds of FUDs:

`yn` Are there objects with property P?

`wh` Find X with property P

`wh_agg` Find the maximal/minimal X with property P

`yn_agg` Does the maximal/minimal X with property P also have property P'?

The body of the FUD may contain items of three different kinds. *Slot–filler items* are of the form

$$slot(\langle frame\ name\rangle, \langle slot\ name\rangle, \langle filler\ value\rangle)$$

This is to be interpreted as saying that the slot $\langle slot\ name\rangle$ of the predicate $\langle frame\ name\rangle$ is filled with the value $\langle filler\ value\rangle$.

Constraint items are of the form

$$exec(\langle goal\rangle)$$

and express numerical relations obtaining between slot-fillers and other values. Finally, *referential* items are of the form

$$ref(\langle filler\ value\rangle, \langle ref\ \ info\rangle)$$

and indicate that the object $\langle filler\ value\rangle$ is linguistically associated with referential information encoded as $\langle ref\ info\rangle$.

For instance, the utterance "I want to arrive in Stockholm before 6 pm" is interpreted as "Find flights arriving Stockholm before 6 pm", and is represented by the following FUD:[3]

```
wh(X,[  slot(trip,trip_id,X),
        slot(trip,trip_mode,plane),
        slot(trip,to_city,stockholm)
        slot(trip,arr_time,T)
        exec(before(T,1800))])
```

The utterance "Is that a direct flight?" is represented by:

```
yn([  slot(trip,trip_mode,plane),
      slot(trip,stops,0),
      slot(trip,trip_id,X),
      ref(X,det(def,sing))])
```

where the `ref` expression represents the referential expression ("that") in the utterance, and signals to the dialogue manager that a reference resolution has to be made.

Utterances like "I want the first flight to Stockholm" and "Which is the cheapest ticket?" translate into `wh_agg` expressions, while utterances like "Is that the first flight?" translate into `yn_agg` utterances. In our Wizard-of-Oz data, the vast majority of user utterances translate into `wh` FUDs (including some utterances that superficially are yes/no-questions, like "Are there any flights to Stockholm on Monday morning?").

When producing the FUD, the Robust Parser does a simple pass over *the top hypothesis from the speech recognizer*, in a manner described in the next section. In contrast, the CLE attempts to extract the "best" grammatical fragment from the lattice of words representing *the top five hypotheses* of the recognizer. Currently, the

[3] The notation used here is simplified; for example, in our implementation each filler value is typed.

Repeat until no words remain:

Read the next word.

If a matching pattern is found (possibly by looking ahead), *then* fill the corresponding slot and throw away the words corresponding to the pattern
else throw away the word.

Figure 2: Basic algorithm of the Robust Parser.

longest grammatical fragment is considered to be the best fragment, a strategy that can sometimes lead to trouble (see Section 5).

It is important to understand that the CLE may fail to translate its analyses into FUDs, because the user's utterance is not possible to capture using one of the FUD forms. In these cases, the CLE does not give any output at all. The Robust Parser, on the other hand, will always produce something; if the input is completely unintelligible it will at least give the minimal output `wh(X,[])`. This robustness is usually an advantage, but sometimes it can lead the system down the wrong path (see Section 5).

3.2 The Robust Parser

The main purpose of the Robust Parser is to rapidly produce *some* useful output even if parts of the input are unintelligible or garbled. We have deliberately aimed for a simplistic approach to be able to compare an atheoretical, shallow method with the high-precision but more resource-demanding and fragile processing carried out by the CLE. Also, experiences from multi-engine systems show that approaches such as these may complement each other well [Frederking and Nirenburg, 1994]. Given these objectives, a straightforward pattern-matching, slot-filling approach seemed most suitable.

A first version of the parser with reasonable coverage was developed in about two person-weeks. Briefly, the parser works as follows: First, it looks for domain-dependent keywords and phrases and produces a list of filled slots as well as information about the utterance type (for example, a `wh` or `yn` question). The rules that guide this process are straightforwardly encoded in a Definite Clause Grammar. The result is then converted into a well-formed FUD. The parser is deterministic in the sense that only the first matching pattern is chosen; hence, only a single analysis is produced. (Interestingly, the fastest parsers reported in the literature are all deterministic, rule-based partial parsers [Abney, 1997, page 128].) The basic algorithm is shown in Figure 2.

4 Dialogue Management

The dialogue manager (DM) is responsible for interpreting each user utterance in its appropriate context, issuing database queries, and formulating responses to the user. The DM maintains a *dialogue state*, which is transformed as a reaction to each incoming message (from the language processors and the database agent). The dialogue state consists of three data structures:

- a list of *objects* that have been introduced in the course of the dialogue. An object may be a concrete train or flight alternative proposed by the system, or a set of constraints given by the user;
- the *dialogue history*, that is, the utterances up to the current point in the dialogue;
- the *agenda*, that is, a stack of goals that the dialogue manager is seeking to meet. The agenda encodes the long-term objectives of the system.

The use of an agenda makes the system flexible, and it is easy to quickly reconfigure the DM to try out different dialogue strategies.

User: I want to go from Gothenburg to Stockholm on Friday. **user:constraint**

System: At what time do you want to leave? **system:ask-for-constraint**

U: In the morning. **user:constraint**

S: There is a train at 5.30 am arriving at 9.45 am. **system:suggestion**

U: Is that a direct train? **user:ask-for-info**

S: Yes. **system:answer-with-info**

U: Is there a later train? **user:ask-for-suggestion**

S: There is a train at 6.06 arriving at 9.15. **system:suggestion**

U: Fine, I'll take that one. **user:accept**

Figure 3: A dialogue fragment annotated with move labels.

4.1 Dialogue Moves

One of the most important tasks of the DM is to categorize each user utterance as a *move* of a certain type. The move categories were again determined based on an analysis of our Wizard-of-Oz data. Figure 3 shows an annotated dialogue fragment including most of the important move categories.

For example, in the **user:constraint** move, the user delimits the range of possible trips he is interested in. By contrast, in the **user:ask-for-info** move the user asks for information about possible trips, but the queried information does not count as content to be added to the current constraints on possible trips. The query is a "side question" not contributing directly to the current set of mutually understood constraints (but may, depending on the answer, lead to a new constraint). In the **user:ask-for-suggestion** move, the user asks for an alternative suggestion without rejecting the previous suggestions from the system (the user might very well go back and accept a previous suggestion).

We distinguish between twelve different user moves and roughly the same number of system moves. The DM catgorizes an utterance as a certain move by computing a heuristic likelihood score for each move type, based on the following factors:

- the existence of suitable contexts. For example, an utterance cannot be classified as a **user:accept** unless the system has proposed some train(s) and/or flight(s) that the user can accept.
- the difference between the propositional contents of the utterance and that of the context. For instance, if these two are inconsistent, the utterance cannot be classified as a **user:accept**; if they are consistent, it is unlikely that the utterance should be classified as a **user:ask-for-suggestion**.
- the precence of keywords in the utterance. For example, if the utterance contains "accept words" like "yes", "ok", etc., the **user:accept** score is increased.

The DM has a set of *game rules* [Power, 1979] that constrains the set of possible response moves, given a particular user move. For instance, currently a **user:ask-for-suggestion** is always followed by a **system:suggestion** or a **system:no-suggestion**. The game rules can also easily be changed to redesign the dialogue structure.

We conjecture that this set of move labels is reusable for a large set of applications; basically any application where the user gradually specifies what she wants, the system presents the user with alternative suggestions, and the user accepts some suggestions and rejects others. The implementation of the Dialogue Manager is divided into domain-independent code and domain-dependent code (i.e. code that directly refers to flights, trains, etc.), and is thus largely reusable. However, we do not have a separate domain description language; to modify the Dialogue Manager to work with a new domain, one has to rewrite the domain-dependent Prolog code.

4.2 The Dialogue Management Cycle

In every turn, the DM receives a number of FUDs. No attempt is made to select the "best" FUD at this stage, but *each* FUD is processed in a number of steps. First, references are resolved and contextual information is added. Since there may be several possible antecedents for each reference, and several possible contexts, this leads to a multiplication of the FUD (typically a FUD gives rise to five to ten "resolved" FUDs).

As the next step, each resolved FUD is classified as a move (of one of the categories mentioned in the previous section), and a likelihood score is computed for each pair of resolved FUD and move. The winning pair is sent on to the next stage; all other candidates stemming from the same original FUD are discarded.

Once the DM knows what move the FUD represents, the dialogue state can be updated. For instance, if the user has made a **user:accept** move, the DM will look for the appropriate object in its list of objects under discussion, and mark that object as 'accepted'. Furthermore, it will add actions to the agenda; in the case of an acceptance move from the user, a move for confirming the booking will be added.

For each FUD:

1. Resolve references
2. Add contextual information
3. Classify the FUD as a certain move
4. Update dialogue state
5. Choose a response action (system utterance or database call)
6. Calculate preference score

Figure 4: Basic working cycle of the dialogue manager.

Next, the DM will choose the system's response by looking at its agenda. The agenda is organised as a stack of items of the form

$$\langle Cond, Action \rangle$$

where *Cond* can be any predicate that can be true or false of a dialogue state. If the top item's *Cond* is true for the current dialogue state, the corresponding *Action* is performed. Typically, *Action* is a response move (for example, a reply from the system to the user), or a database lookup. (It may also be an instruction to reorganize some internal data structure.) If *Cond* is false, the whole item is popped off the agenda, and the DM proceeds to the next item.

Finally, the chosen response action is given a score by a heuristic function. For example, prompting the user to rephrase his last utterance is judged as being less productive than asking "When do you want to travel" or performing a database lookup. Furthermore, the response may receive extra scores based on the input FUD (e.g. for the number of previously unknown filler values the FUD determined).

The winning response action is then carried out, which amounts to sending a message to the linguistic generator (in case of a system utterance), or to the database agent. The working cycle of the DM is summarized in Figure 4.

5 A Preliminary Evaluation

This section reports the results of a preliminary evaluation, aimed particularly at testing the relative utility of the Robust Parser and the CLE, respectively. To this end, we used two configurations of the system: One of them (RP–CLE) corresponds to the architecture shown in Figure 1, in which the CLE and the Robust Parser work in parallel. In the other (RP-only), the CLE was disabled, thus only containing the shallow processing path.

Two similar tasks, A and B, were created, each involving a trip with at least three legs during two consecutive days, suitable for both train and air travel. Two subjects were used. Each of them was given the opportunity to try out the RP–CLE version of the system. More specifically, what they used was the demo version of the system, in which system components get highlighted as they engage in processing, and in which the recognized utterance as well as the system's responses are successively written into a window. The purpose of this was to give the subjects a better sense of what was going on, since otherwise the system could remain silent for typically 30–60 seconds on Internet database queries. When the subjects felt that they were able to handle the system, they were presented with tasks A and B in different orders.

The experiment resulted in four dialogues, each consisting of between 22 and 28 user–system turns. Each turn was tagged with "OK" or "failure", depending on whether the system had managed to move the dialogue forward or not in response to the user's utterance (provided that the utterance was reasonable given the context). "Failure" thus consists of cases where the system responded that it did not understand the last utterance or where its response constituted a misunderstanding. Furthermore, each turn was tagged with "user" or "system", depending on whether the subject's utterance was a response to a system initiative or whether the utterance constituted a user initiative (for example, a spontaneous request for information or a counter-question). The tasks were designed so as to encourage mixed initiative, and both subjects displayed a majority of user initiatives in their dialogues.

Because of the small size of the experiment, the results at this point can only be taken as suggestive. Nevertheless, to provide a rough idea of where we stand in our on-going work, we shall briefly present some figures that we obtained.

To begin with, the RP–CLE configuration appeared slightly more efficient in terms of moving the dialogue forward than the RP-only one: The RP–CLE and RP-only dialogues used on average 22 and 27 moves, and out of these had 15 and 14 "OK" turns, respectively. However, in terms of providing successful analyses (in the cases when at least one fragment of the output from the speech recognizer was reasonable), the RP was the slightly more successful one in the RP–CLE configuration: It succeeded on average on 16 turns, whereas the CLE succeeded on 13. Surprisingly, the RP also turned out to be a bit more successful on those turns where the user had taken the initiative: it was successful on almost 2/3 of those cases, whereas the CLE was successful on about half of them.

A closer analysis revealed that on five times in each of the RP–CLE dialogues, failure of the CLE to deliver a correct analysis was due to the fact that it had chosen a wrong fragment (usually too long). The reason for this is that the CLE attempts to analyse the longest grammatical fragment on the path chosen from the N-best list, something which may lead to strange results (compare the example further below).[4]

[4] A previous study using our Wizard-of-Oz data came to a similar result; see [Lewin *et al.*, 1999].

In terms of which component causes the most turn failures, the picture was unclear. In the RP–CLE case, only a single "failure" turn in each dialogue was actually due to language analysis (in which case both the RP and the CLE failed, though the CLE had the better analyses). In the RP-only case, the RP caused none at all of 11 failures in one of the dialogues, whereas in the other, it caused 5 of 15 failures. The figures also indicate that language analysis was not the main bottleneck of the system (both speech recognition and dialogue management were the sources of more failed turns). This might have played a role when none of the subjects said that they had noted any difference in terms of overall performance between the RP–CLE and RP-only configurations of the system. But the relatively small difference in terms of overall turn efficiency, as indicated above, might also have contributed to this.

Our analysis also indicates that the Dialogue Manager is quite good at choosing between analyses from the RP and CLE: In the two RP–CLE dialogues, there is only a single case of the Dialogue Manager choosing the wrong alternative. (In this case, it chooses a CLE analysis which lacks some information but the rest of whose contents are correct, thereby still managing to move the dialogue forward.)

We now turn to some qualitative differences between the RP and CLE that we have observed in our analysis above.[5] To begin with, the obvious advantage of the Robust Parser (RP) is that it is rather undisturbed by ungrammaticalities, disfluences and (to some extent) recognition errors in the input. For example, the utterance

> Hej jag beställer en flygbiljett den åttonde i sjätte tisdag från Stockholm till Sundsvall. (*Hi I'm ordering a flight ticket on June eighth from Stockholm to Sundsvall*).
>
> recognized as VAD HEJ JAG BESTÄLLER JAG VILL JAG DEN ÅTTONDE I SJÄTTE I JAG MMM DÅ STOCKHOLM TILL SUNDSVALL. (roughly *What hi I'm ordering I want I on June eighth in I mmm then Stockholm to Sundsvall.*)

is analysed perfectly by the RP. The CLE locates the longest grammatical fragment "den åttonde i sjätte", and produces an analysis that includes the date but not the destination and origin cities of the trip.

As pointed out above, the strategy of choosing the longest grammatical fragment can sometimes lead the CLE completely astray. The utterance

> Jag bokar det tåget. (*I book that train.*)

was misrecognized as

> JAG BOKAR DET DET TÅGET

whose longest grammatical fragment is "bokar det det tåget" (*"does that book that train"*), which is something completely different from what the user actually said.

[5] All example utterances below come from the experiment described above, using the system.

The CLE failed to produce any FUD, while the RP got it right.

On the other hand, the RP can produce erroneous results because it is analysing unconnected bits and pieces of sentences. For instance, the RP analysed "Klockan nitton eller senare" (*"at seven pm or later"*) as "*at seven pm, and later than some previously mentioned trip*", because it triggered on the two separate patterns "klockan nitton" and "senare" without considering the relation between them.

Actually, the very robustness of the RP can sometimes prove to be a disadvantage. In one case, the test subject meant to say "Jag har företagsrabatt på flyget" (*"I have a corporate discount on air travelling"*), but the input became totally garbled: "JA DÅ HAR FÖRETAG FYRA VAD FÖR ATT FLYGA" (roughly *"Yes then has company four what for to fly"*). The CLE did not produce any FUD. The RP reacted on "to fly", and its analysis together with the keyword "Ja" (*"Yes"*) in the utterance made the system book a previously mentioned flight alternative. If the RP had been disconnected, the system's reply would instead have been to ask the user to rephrase her utterance; certainly a more sensible reaction.

A considerable advantage of the CLE is its ability to look at, and possibly combine, the top N hypotheses from the recognizer. At several occasions this proved to be important, for example, in correcting the top hypothesis "hur och retur" into "tur och retur" (*"return trip"*). The RP, which only has access to the top hypothesis, could only produce the "null" result `wh(X, [])` in this case.

6 Conclusion

We have described an implemented spoken-language dialogue system, which combines deep and shallow language-processing engines and an agenda-driven dialogue manager. We have also described a preliminary evaluation of the system. Although the limited set of data prevents us from drawing any firm conclusions, we feel encouraged in further exploring parallel shallow and deep language processing in the context of spoken-dialogue systems. There are principled cases that the shallow processor has problems dealing with, and in our limited experiment the configuration that combined deep and shallow processing was at a slight advantage relative to the one that only used shallow processing. On the other hand, our hypothesis that deep processing was more advantageous in situations where the user takes the initiative did not receive support by this experiment. Actually, the RP appeared slightly more successful both in general and on user initiatives. However, in our experiment this difference could be largely explained by the extent to which the CLE chose the wrong (grammatical) fragment from the N-best list. It thus seems that we had underestimated the degree to which output from the speech recognizer would require fragment analysis whose results might require careful selection. To be able to deal with this, we would have to improve fragmentary-analysis selection in the CLE so that decisions can be made statistically from the results of supervised training over already parsed corpora. Some work has been done to integrate this technique into our general tool for customizing the disambiguation component of a language processor [Carter, 1997].

References

[Abney, 1997] Steven Abney. Part-of-Speech Tagging and Partial Parsing. In Steve Young and Gerrit Bloothooft, editors, *Corpus-based Methods in Language and Speech Processing*, pages 118–136. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

[Allen *et al.*, 1996] James F. Allen, Bradford W. Miller, Eric K. Ringger and Teresa Sikorski. A Robust System for Natural Spoken Dialogue. In *Proc. 34st Annual Meeting of the Association for Computational Linguistics*, pages 62–70, Santa Cruz, California, USA, 1996.

[Alshawi, 1992] Hiyan Alshawi, editor. *The Core Language Engine*. MIT Press, Cambridge, Massachusetts, 1992.

[Aust and Oerder, 1995] Harald Aust and Martin Oerder. Dialogue Control in Automatic Inquiry Systems. In *Proc. ESCA Workshop on Spoken Dialogue Systems: Theories and Applications*, Vigsø, Denmark, 1995.

[Bos *et al.*, 1999] J. Bos, S. Larsson, I. Lewin, C. Matheson and D. Milward. Survey of Existing Interactive Systems. Trindi (Task Oriented Instructional Dialogue) report number D1.3, 1999. [Trindi, Telematics Application Programme, Language Engineering LE4-8314, funded by the European Commission.]

[Carter, 1997] David Carter. The Treebanker: A Tool for Supervised Training of Parsed Corpora. In *ACL Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, Madrid, 1997. See also: SRI Technical Report CRC-068 available from http:/www.cam.sri.com.

[Frederking and Nirenburg, 1994] Robert Frederking and Sergei Nirenburg. Three heads are better than one. In *Proc. 4th Conference on Applied Natural Language Processing*, pages 95–100, Stuttgart, Germany, 1994.

[Lamel *et al.*, 1998] L. Lamel, S. Rosset, J. L. Gauvin, S. Bennacef, M. Garnier-Rizet and B. Prouts. The LIMSI ARISE System. In *Proc. IEEE 4th Workshop Interactive Voice Technology for Telecommunications Applications*, pages 209–214, Torino, Italy, 1998.

[Lewin *et al.*, 1999] I. Lewin, R. Becket, J. Boye, D. Carter, M. Rayner, and M. Wirén. Language processing for spoken dialogue systems: is shallow parsing enough? In *Accessing Information in Spoken Audio: Proceedings of ESCA ETRW Workship, Cambridge, 19 & 20th April 1999*, pages 37–42, 1999.

[Litman *et al.*, 1998] Diane J. Litman, Shimei Pan and Marilyn A. Walker. Evaluating Response Strategies in a Web-based Spoken Dialogue Agent. In *Proc. ACL/COLING 98: 36th Annual Meeting of the Association of Computational Linguistics*, pages 780–787,

[Murveit *et al.*, 1993] H. Murveit, J. Butzberger, V. Digalakis and M. Weintraub. Large Vocabulary Dictation using SRI's DECIPHER(TM) Speech Recognition System: Progressive Search Techniques. In *Proc. International Conference on Acoustical, Speech and Signal Processing*, Minneapolis, Minnesota, 1993.

[Power, 1979] Richard Power. The Organization of Purposeful Dialogues. *Linguistics*, 17:107–152, 1979.