menu2dialog

Staffan Larsson, Robin Cooper, Stina Ericsson
{sl, cooper, stinae}@ling.gu.se
Department of linguistics, Göteborg University
Box 200, Humanisten, SE-405 30 Göteborg, Sweden

Abstract

While menu interfaces are ubiquitous in modern technology they are often tedious and frustrating. Current dialogue technology can straightforwardly implement menu structures but the effect can be even more tedious since the user is forced to use the natural and efficient modality of speech in a very unnatural way descending the menu structure one node at a time. GoDiS, an experimental dialogue system implemented using the TRINDIKIT, offers the possibility of allowing the user to present several pieces of relevant information at one time or to present information in the order in which the user finds most natural.

1 Introduction

Much of our interaction with automated task oriented systems is currently menu driven[1]. For example, an automated system to book a cinema ticket can involve first choosing the city, then the film, the cinema and the time. Current technology using a telephone interface involves listening to a number options and responding by pressing a number. An example of a more complex menu structure is present in the programming facilities of mobile phones where the menu can be viewed as a tree structure which the user has to descend starting at the root node.

While menu interfaces are ubiquitous in modern technology they are often tedious and frustrating. Current dialogue technology can straightforwardly implement menu structures but the effect can be even more tedious since the user is forced to use the natural and efficient modality of speech in a very unnatural way descending the menu structure one node at a time. GoDiS, an experimental dialogue system implemented using the TRINDIKIT, offers the possibility of allowing the user to present several pieces of relevant information at one time or to present information in the order in which the user finds most natural. This means that users can use their own conception of the knowledge space and not be locked to that of the designer of the system.

We first introduce GoDiS and the type of information state it uses. We then introduce the concepts of question and task accommodation, and define the kind of dialogue plans used by GoDiS. The relation between menus and dialogue plans is then examined, and some dialogue plans used by GoDiS in the mobile phone interface domain are presented. Finally, some sample dialogues demonstrate the use of question and task accommodation in menu-based dialogue in GoDiS.

In this paper we present the ideas behind the adaption of GoDiS to the mobile phone interface domain, and more generally to domains involving menu-based interfaces. The paper is thus of a theoretical rather than practical nature, and we have not yet evaluated the GoDiS system in this domain. Also, since the topic of the paper is how menus can be used in dialogue management, we do not discuss general issues concerning dialogue systems, such as speech.

2 Information state in GoDiS

The techniques described in this paper have been implemented in GoDiS[Bohlin *et al.*, 1999], an experimental dialogue system initially adapted for the travel agency domain but later adapted for several other domains, including mobile phone interface. GoDiS is implemented using the TRINDIKIT[Larsson and Traum, 2000; Larsson *et al.*, 2000], a toolkit for experimenting with information states and dialogue move engines and for building dialogue systems.

The notion of information state we are putting forward here is basically a version of the dialogue game board which has been proposed by Ginzburg [Ginzburg, 1998]. We represent information states of a dialogue participant as a record of the type shown in figure 1.

The main division in the information state is between information which is private to the agent and that which is shared between the dialogue participants. The private part of the information state contains a PLAN field holding a dialogue plan, i.e. is a list of dialogue actions that the agent wishes to carry out. The AGENDA field, on the other hand, contains the short term goals or obligations that the agent has, i.e. what the

[1] Work on this paper was supported by SIRIDUS (Specification, Interaction and Reconfiguration in Dialogue Understanding Systems), EC Project IST-1999-10516, and D'Homme (Dialogues in the Home Machine Environment), EC Project IST-2000-26280. The first author also wishes to thank STINT (The Swedish Foundation for International Cooperation in Research and Higher Education). The authors would also like to thank the students of the Dialogue Systems course held at Göteborg University in the spring of 2000 for help with the implementation of the telephone interface adaption of GoDiS.

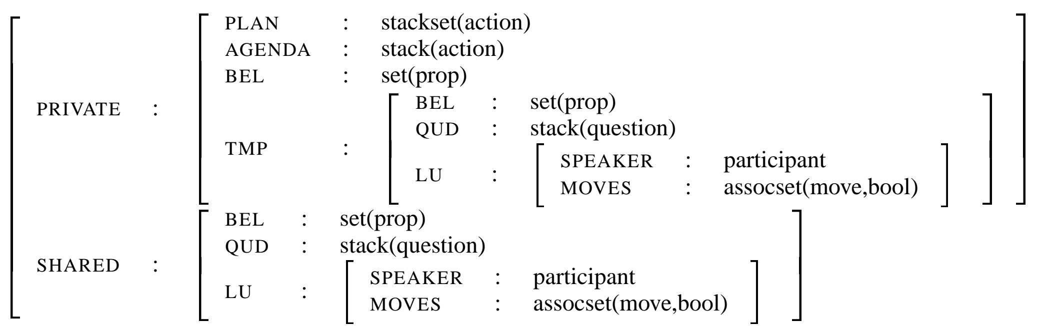

Figure 1: The type of information state we are assuming

Question	Relevant answers
$?\lambda x.P(x)$	a or $P(a)$ provided $P(a)$ is a proposition
$?P$	yes, no, P, or $\neg P$
$\{?L_1, ?L_2, \ldots, ?L_n\}$	$L_1, L_2, ...,$ or L_n

Table 1: Relevant answers to questions

agent is going to do next. We have included a field TMP that mirrors the shared fields. This field keeps track of shared information that has not yet been grounded, i.e. confirmed as having been understood by the other dialogue participant. The SHARED field is divided into three. One subfield is a set of propositions (commitments) which the agent assumes for the sake of the conversation (COM). The other subfield is for a stack of questions under discussion (QUD). These are questions that have been raised and are currently under discussion in the dialogue. The LU field contains information about the latest utterance.

We define *update rules* for updating the information state based on the recognised move(s). The rules are defined in terms of preconditions and effects on the information state; the effects are a list of operations to be executed if the preconditions are true. As an example, the update rule for integrating an answer (a simplified version of Ginzburg's QUD downdate rule) is shown below.

RULE: **QUD-downdate**

PRE: {
- member(SHARED.LU.MOVES, answer(A))
- fst(SHARED.QUD, Q),
- domain : answer_to(Q, A)

EFF: {
- pop(SHARED.QUD)
- reduce(Q, A, P)
- add(SHARED.COM, P)

Informally, if the user just performed an "answer" move with content A, and there is a question Q topmost on QUD such that A is an answer to Q, pop Q off the QUD and add the proposition P (obtained by β-reduction) to the shared beliefs. There is a corresponding rule **QUD-update** for updating the QUD when a question is asked, by pushing it on top of QUD.

3 Question and task accommodation

On of the techniques we have investigated in GoDiS is a notion of accommodation based on an information update perspective on Lewis' notion of accommodation [Lewis, 1979].

Dialogue participants can address questions that have not been explicitly raised in the dialogue. In such cases, coherence is preserved if the agent is able to find a question which is relevant at that point in the dialogue which can then be accommodated onto the QUD.

The update rule for question accommodation, QuAcc, is shown below. When interpreting the latest utterance by the other participant, the system makes the assumption that it was an **answer** move with content A. This assumption requires accommodating some question Q such that A is a relevant answer to Q. The check operator "answer-to(A, Q)" is true if A is a relevant answer to Q given the current information state, according to some (possibly domain-dependent) definition of question-answer relevance.

RULE: **QuAcc**

PRE: {
- member(SHARED.LU.MOVES, answer(A)),
- val(SHARED.LU.SPEAKER, usr),
- member(PRIVATE.PLAN, raise(Q))
- domain : answer_to(A, Q)

EFF: {
- del(PRIVATE.PLAN, raise(Q))
- push(SHARED.QUD, Q)

Since the question-answer relation is crucial to the concept of question and task accommodation, we provide table 1 indicating the relevant answers to different types of questions.

Question accommodation presupposes that the system has a plan where it can search for questions to accommodate. However, in some cases (e.g. the beginning of a dialogue) the system does not have a plan containing a suitable question. In this case, if the user makes an utterance containing information which can be interpreted as an answer in the domain, the system has to find a plan containing matching questions. The domain knowledge contains plans for various tasks that the system can perform. What it needs to do is take the user's utterance and try to match it against questions in the plan types in its domain knowledge. When it finds a suitable match it will accommodate the task corresponding to the matching plan by adding a proposition indicating the

Menu	Plan
multi-choice list $\langle L_1, L_2, \ldots, L_n \rangle$	action to resolve alternative question plus case-statement `findout`$(\{?L_1, ?L_2, \ldots, ?L_n\})$, `case`$(\langle\langle L_1,$ `exec`$(T_1)\rangle, \ldots, \langle L_n,$ `exec`$(T_n)\rangle\rangle)$ where T_i is the task corresponding to L_i
tick-box or equivalent +/- P	action to resolve y/n-question `findout`$(?P)$
dialogue window parameter=_	action to resolve wh-question findout$(?\lambda x.$parameter$(x))$
pop-up message	action to inform

Table 2: Conversion of menus into dialogue plans

task to the shared beliefs. This done by the **TaskAcc** rule. This rule also puts the inferred plan in the PLAN field.

RULE: **TaskAcc**
PRE: { member(SHARED.LU.MOVES, $Move$)
domain : match_task($Move, Task, Plan$) }
EFF: { add(SHARED.COM, task($Task$))
set(PRIVATE.PLAN, $Plan$) }

Once the system has accommodated the task and found the plan it can accommodate the QUD with the relevant question and proceed with the interpretation of ellipsis in the normal fashion. Accommodating the dialogue plan in this way actually serves to drive the dialogue forward. That is, the mechanism by which the agent interprets this ellipsis, gives him a plan for a substantial part of the rest of the dialogue.

When the system is given one or several answer-moves matching more than one plan, the system will ask the user to clarify what the user wants. For example, in the phone interface domain several plans contain the question "What name?" ($?\lambda x.$name(x)), so if the user just provides an answer (e.g. by saying "Jim"), the system will ask "What do you mean by that? Do you want to search the phonebook, add a new number, change a name, delete a name, or assign a ringing tone?".

4 Dialogue plans

In this section, we introduce a simple formalism for representing dialogue plans. The following constructs are used:

- A, where A is an action to be executed
- `exec`(S), where S is a task to be executed
- $\langle B_1, B_2, \ldots, B_n \rangle$, where $B_1, B_2, \ldots, B_n$ is a list of plan constructs to be executed in sequence
- `if_then`(P, A), where P is a proposition and A is a plan construct; if P is in SHARED.COM, execute A. (Free variables in the proposition may become bound when checking SHARED.COM; these binding are preserved in A but not outside the `if_then` construct.)
- `case`$(\langle (P_1, B_1), (P_2, B_2), \ldots, (P_n, B_n), C \rangle)$, where P_x are propositions, and B_x and C are plan constructs; if P_1 holds, execute B_1; else, if P_2 holds, execute B_2, etc. ; else, execute C.

The available actions related to dialogue moves are the following:

- `findout`(Q): find the answer to Q. This is usually done by asking a question to the user, i.e. by performing an `ask` move. The proposition resulting from answering the question is stored in `shared.com`. The `findout` action is not removed until the question has been answered.
- `raise`(Q): raise the question Q. This is always done by asking a question to the user, i.e. by performing an `ask` move. The proposition resulting from answering the question is stored in `shared.com`. The `raise` action is removed as soon as it has been integrated.
- `respond`(Q): respond to question Q. If is an answer to Q in the PRIVATE.BEL field, perform an `answer` move. If there is no such answer, the system will respond by saying that no answer is available.
- `inform`(P): perform an `inform` move with content P.

There are also actions for accessing resources such as databases and external devices. These actions may be domain-specific. In the mobile phone system the resource-related actions include:

- `lookup_database`(Q): using the information currently in SHARED.COM, try to find the answer to question Q in the database and store it in PRIVATE.BEL. (If there is no answer to Q, the database query will return **failed**.)
- `call`(N): call telephone number N.
- `assign_tone`$(Name, Tone)$: assign ringing tone $Tone$ to name $Name$. (This causes $Tone$ to be used as a ringing tone whenever $Name$ is calling from the number specified in the phonebook)

5 From menu to dialogue

Menus can be translated into dialogue plans using a fairly simple schema. We assume menu interfaces consist of (at least) the following elements:

- *multi-choice lists*, where the user specifies one of several choices
- *dialogue windows*, where the user enters requested information using the keyboard
- *tick-box*, which the user can select or de-select

```
⟨ findout({?task(search_phonebook), ?task(service_number),
          ?task(add_entry), ?task(erase), ?task(edit),
          ?task(assign_tone)}),
  case(⟨ ( task(search_phonebook), exec(search_phonebook) ),
          ( task(service_number), exec(service_number) ),
          ( task(add_entry), exec(add_entry) ),
          ( task(erase), exec(erase) ),
          ( task(edit), exec(edit) ),
          ( task(assign_tone), exec(ringtone) ) ⟩
```

Figure 2: Toplevel phonebook plan

```
⟨ findout( ?λx.name(x) ),                              "What name?"
  lookup_database(?λx.number(x) ),
  respond( number(Num) ) ),                            "Then number is Num"
  if_then( number(Num),
          ⟨    findout( ?call ),                       "Do you want to make a call?"
               if_then( call & name(N),
               ⟨ call(Num),
                 inform(call_name(N))⟩⟩⟩               "Calling N"
```

Figure 3: Plan for searching the phonebook

S: What can I do for you?
U:(silence)
S: Do you want to go to the phonebook, to messages, to clock, to calculator or to games?
U:Phonebook
S: Do you want to search, add a new entry, erase an entry, change an
entry, or assign a ringing tone?
U:Search
S: What name?
U:Jim
S: The number is 548435. Do you want to make a call?
U:Yes
S: Calling Jim.

Figure 4: Sample dialogue with menu traversal

S: What can I do for you?	[ask(?λx.task(x))]
U:Jim	[answer(jim)]
S: OK. Do you want to search, add a new entry, delete an entry, edit an entry, or give a ringing tone ?	[ask(⟨ ?task(search_phonebook), ?task(add_entry), ?task(erase), ?task(edit), ?task(assign_tone) ⟩)
U:Assign tone	[answer(assign_tone)]
S: Which tone?	[ask(?λx.tone(x))]
U:Bossanova	[answer(name(jim))]
S: The ringing tone for Jim is now bossanova.	[inform(ringing_tone(jim, bossanova))]

Figure 5: Sample dialogue with task and question accommodation

S: What can I do for you?	`[ask(?λx.task(x))]`
U:Call Jim	`[answer(call), answer(name(jim))]`
S: The number is 548435. Calling Jim.	`[answer(number(548435)), inform(call(jim))]`

Figure 6: Sample dialogue with task and question accommodation

- *pop-up messages* containing information from the system

The correspondence between menu elements and plan constructs is shown in table 2.

(For alternative questions, it is sometimes a good idea to first raise a wh-question (e.g. "What can I do for you?") to allow the user to answer this question without having to hear all the alternatives. Only if this does not succeed will it be necessary to ask the alternative-question.)

For example, the phonebook menu consists of a multiple-choice list ⟨ **`Search, Service Nos., Add entry, Erase, Edit, Assign tone`** ⟩. This corresponds to the plan in figure 2.

The menu for searching the phonebook consists of

1. a dialogue window asking for a name
2. a pop-up message giving the number
3. a choice to call or not call the person
4. if a call is made, a pop-up message

This menu can be translated to the plan in figure 3, using the translation schema and some additional plan components, including a database search. We aim at eventually being able to translate menus into dialogue plans automatically, possibly using XML markup of menus and dialogue plans.

6 Sample dialogues

In the simplest type of dialogue, the user traverses the menu step by step by answering system questions, as in figure 4.

In figures 5 and 6, we see two sample dialogues where the user provides information forcing the system to do task and question accommodation. The dialogue in 5 also contains a clarification question from the system regarding what task the user wants the system to perform. The tasks asked about are all tasks whose plan contain the question $?\lambda x.\text{name}(x)$ ("What name?").

7 Conclusion

Information-seeking systems are similar in many ways to menu-driven systems; basically, the user needs to give the system certain information which enables to system to perform its task - booking a ticket, providing price information, making a call etc. Therefore, the techniques developed for information-seeking dialogue apply also to menu-driven systems.

GoDiS, an experimental dialogue system implemented using the TRINDIKIT, offers the possibility of allowing the user to present several pieces of relevant information at one time or to present information in the order in which the user finds most natural (either in speech or written interfaces). The system will request information that is missing. If information presented by the user is relevant to a specific branch of the menu system then the system will be able to jump directly to that branch without requiring the user to step through all the intervening nodes. If the user does not know what to do the system can present the options. This means that users can use their own conception of the knowledge space and not be locked to that of the designer of the system.

References

[Bohlin *et al.*, 1999] P. Bohlin, R. Cooper, E. Engdahl, and S. Larsson. Information states and dialogue move engines. In J. Alexandersson, editor, *IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 1999.

[Ginzburg, 1998] J. Ginzburg. Clarifying utterances. In J. Hulstijn and A. Niholt, editors, *Proc. of the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues*, pages 11–30, Enschede, 1998. Universiteit Twente, Faculteit Informatica.

[Larsson and Traum, 2000] Staffan Larsson and David Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *NLE Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, 2000.

[Larsson *et al.*, 2000] Staffan Larsson, Alexander Berman, Johan Bos, Leif Grönqvist, Peter Ljunglöf, and David Traum. Trindikit 2.0 manual. Technical Report Deliverable D5.3 - Manual, Trindi, 2000.

[Lewis, 1979] D. K. Lewis. Scorekeeping in a language game. *Journal of Philosophical Logic*, 8:339–359, 1979.