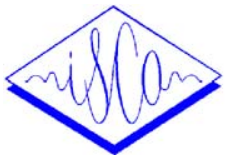

6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

USER MODELLING IN ADAPTIVE DIALOGUE MANAGEMENT

Gert Veldhuijzen van Zanten
IPO, Center for Research on User-System Interaction,
P.O. Box 213, 5600 MB Eindhoven, the Netherlands
G.E.Veldhuijzenv.Z@tue.nl

10.21437/Eurospeech.1999-276

ABSTRACT

This paper describes an adaptive approach to dialogue management in spoken dialogue systems. The system maintains a user model, in which assumptions about the user's expectations of the system are recorded. Whenever recognition errors occur, the dialogue manager drops assumptions from the user model, and adapts its behaviour accordingly. The system uses a hierarchical slot structure that allows generation and interpretation of utterances at various levels of generality. This is essential for exploiting mixed-initiative to optimise effectiveness. The hierarchical slot structure is also a source of variation. This is important because it is ineffective to repeat prompts in cases of speech recognition errors, for in such cases users are likely to repeat their responses also, and thus the same speech recognition problems reoccur.

1. INTRODUCTION

The effectiveness of dialogue-management strategies heavily depends on expectations that the user has of the system. Any reasonable strategy survives when users know what is expected of them. Getting a user to know this, however, is a far from trivial matter. It is very hard to explain the system behaviour in a concise and comprehensive introduction. It is better to give guidance during the dialogue. However, such guidance can be time-consuming and irritating to the more confident (or experienced) user. Therefore, we developped an adaptive strategy that gives guidance only when necessary, i.e. when not giving guidance causes errors. Note that even a simple system prompt asking a question, like *"What is your destination city?"* already gives a kind of guidance. Under certain circumstances this inherent guidance can be misleading to the user. The question does not suggest that the system is capable of grasping a reply like *"I need to be in Amsterdam tomorrow morning at 10 o'clock."* This is unfortunate since it means that users hardly ever exploit mixed-initiative capabilities while that could potentially lead to much shorter dialogues. Dialogue designers should therefore take into account not only whether the selected prompt conveys the intended effects of the designer, but should also take into account which (possibly unintended) effects a prompt may have on a user.

Our approach rests on two pillars. One is a hierarchical slot structure that allows questions to be asked on different levels of granularity. This provides for alternatives when certain prompts are too specific and would be misguiding. The second pillar is a user model in which assumptions about the expectations of the user are recorded. This is essential for adaptive system behaviour.

We give an account of the hierarchical slot structure and the user model, and pose a number of rules that describe when prompts are appropriate, depending on the user model. A question about some higher level slot is appropriate only when the user knows which lower level slots the system regards the higher level slot to be constructed of. Eg. a question like *"In which train connection are you interested?"* is appropriate only when the user knows that a reply can consist of an origin and destination station and some reference to the date and time of travel. Asking for each of these lower level items in turn is a way to give this information to the user.

2. DIALOGUE MANAGEMENT

Dialogue management can be seen as a combination of various strategies that operate on different communication layers.

- The top layer is concerned with the goals that system and caller try to achieve.
- A second layer deals with the distribution of attention. This layer is concerned with coherence of discourse.
- A third layer is concerned with giving guidance to the caller.
- A fourth layer contains strategies for grounding the information that is being conveyed. This layer is concerned with verification and acknowledgement and such. For a theory concerned with this layer, see Traum [2].

- The next layers deal with the actual utterances, at speech act, linguistic, word and speech levels.

Grosz and Sidner [1] describe a theory that deals with the intentional and attentional layers (layers 1 and 2) and describe some relations with of the lower layers. In this paper, we will focus on the third layer, where an important role is played by the user model. We use a hierarchical slot structure as a framework to which we can attach the different dialogue management layers. The hierarchy supplies anchoring pionts for each of the layers; it does not only represent the structure of the data in the domain, it can also be viewed as a representation of the task that the system must perform and the goals that are associated with it. For each layer, we can associate with the slots some information items that can be used to drive the corresponding strategies. First we will describe the hierarchical slot structure, and then we will show how it is used for dialogue management.

3. HIERARCHICAL SLOT STRUCTURE

Many spoken dialogue systems employ some kind of slot filling. In these systems the information that is needed from the user is modelled by a list of slots. Slots are associated with prompts in such a way that the prompts invite the user to supply values for the corresponding slots. A drawback of the slot-filling paradigm as it stands, is that it results in rather rigid dialogues. Users of automatic systems tend to follow the structure imposed by the system, rather than taking the initative and give over-informative answers. It is our claim that this is at least partly due to the way that prompts are directed at specific slots. By asking specific questions like *"On which date do you want to travel?"* a system does not give a clue that it might be able to process over-informative answers, which include for instance the time of travel. As a result users hardly ever give over-informative answers to such systems, even though this could make the dialogue considerably more efficient.

Other drawbacks of using a fixed set of slots occur in different dialogue management layers. For instance, problems that arise in the understanding of parts of slot values cannot be discussed, because there are no slots associated with the parts.

To alleviate some of the problems mentioned above, we will use a hierarchically structured set of slots. The hierarchy contains slots at various levels of abstraction. These slots give rise to questions from relatively general ones like *"When do you want to travel?"* to very specific questions like *"Do you want to travel on September 30th?"* and several levels in between.

To demonstrate the usefulness of representing different levels of generality, we give an example. Many systems contain a `date` and a `time` slot. Given these slots, we can design prompts like *"On which date?"* and *"At what time do you want to travel?"* Thus, we need two questions and verifications to fill both slots. In [3], Sanderman et al show that using today as default date, can sometimes help. However, what if it happens to be a few minutes before midnight? Is today then still a good default?

If these slots are modelled as sub-slots of a more general `moment` slot, then the dialogue becomes much simpler. The system can simply ask for a value for the moment slot, for instance with *"When do you want to travel?"*. If the user answers with just a time, then we can, in many domains, assume that that time will (by default) refer to the nearest moment in the future. Either today or tomorrow, depending on whether it is now earlier or later that the given time. This is in accordance with the Gricean maxim of quantity [4]. Thus, by adding a little more structure, we can deal with dates and times with just one question instead of two (possibly followed by verification), invite users to supply a date and a time in one utterance, and get the right defaults at the same time. Compare the two dialogues

S: Do you want to travel today? U: Yes S: At what time? U: At two o'clock in the afternoon S: At two pm? U: Yes

and

S: When do you want to travel? U: at two o'clock S: Two o'clock this afternoon? U: Yes

Having a hierarchical slot structure is essential in the design of flexible dialogues. In [5], we described an adaptive mixed-initiative dialogue management strategy. Depending on the success or failure of certain questions, the system zooms-in to more detailed questions, or zooms-out to higher-level questions. Below, we will give a more formal description of the hierarchical slot structure and the focus on the implications on an associated user model.

3.1. Formal definition

A slot is a container for a value that the system needs to know in order to perform an action. The

user may chose one value from a set of values, called the range of the slot.

Definition: A hierarchical slot structure consists of a set of slots S, a set of values V, a non-cyclic, non-reflexive feature relation $F \subseteq S \times S$ and a function ρ: $(S \rightarrow \mathbf{2}^V) \cup (F \rightarrow V \rightarrow V)$ for which the following holds:

$$f=(s_1,s_2) \Leftrightarrow (\forall v \in \rho\, s_1)(\rho\, f\, v \in \rho\, s_2)$$

❑

The function ρ determines which values can be stored in a slot. And it also determines the semantics of the features that constitute the hierarchy. For instance, given a time slot t, an hour slot h, and a minute slot m that are connected by features hour=(t,h) and minute=(t,m), the semantics of this structure can be given by a function ρ for which

$\rho\, t$ = {0..1439}
$\rho\, h$ = {0..23}
$\rho\, m$ = {0..59}
ρ hour = $\lambda t.\ t$ **div** 60
ρ minute = $\lambda t.\ t$ **mod** 60

During the dialogue, we must remember which values are stored in which slots, and which values have been denied. For this purpose, an information state is maintained.

Definition: An information state I is a set of propositions of the form (s=v) or of the form (s#v), where s is a slot and v is a value from the range of s ($v \in \rho\, s$). An information state I is logically consistent iff it satisfies the predicate *consistent* defined by

consistent(I) iff for all $s \in S$ and $v,v' \in V$
(s=v) $\notin I$ or (s#v) $\notin I$, and
((s=v) $\in I \wedge$ (s=v') $\in$ I) $\Rightarrow v=v'$

We define the closure I^* of an information state I as the smallest set for which we have that for all features f =(s_0,s_1) $\in F$ and values v_0, v_1 such that $v_1=\rho\, f\, v_0$ the following holds

$I \subseteq I^*$,
(s_0=v_0) $\in I^* \Rightarrow$ (s_1=v_1) $\in I$,
(s_1#v_1) $\in I^* \Rightarrow$ (s_0#v_0) $\in I$ and
if *consistent* ($I^* \cup \{s=v\}$) and
$\forall v'$(*consistent* ($I^* \cup \{s=v'\}$) $\Rightarrow v=v'$)
then
(s=v)$\in I^*$

❑

A proposition (s=v) states that slot s has value v. A proposition (s#v) states v is not the value of slot s. Consistency of information states corresponds to the requirement that a slot can only have one value and that value cannot be both the same as and different from itself. The closure operation models the reasoning that is needed to relate values of slots at different levels of generality.

The hierarchical slot structure serves as the skeleton of the dialogue manager: Slots can represent goals. Subslots can naturally represent sub-goals. The concept of attentional focus is greatly supported by having concepts at different levels of generality; Adding a grounding status to each slot supports grounding strategies; and underspecification in the caller's utterances can be resolved using the hierarchy (see [6]). More importantly, the strategies that operate on various layers of dialogue management can be described without referring to the specifics of the task domain. We have implemented a dialogue manager that can be parameterized with a domain specification that is basically a hierarchical slot structure.

In the next section, we will describe how a user model can be associated with the hierarchical slot structure that supports the generation of prompts at the right level of generality.

4. USER MODELING

One of the main concerns in dialogue design is to deal with misunderstandings. At first thought, it may seem best to prevent them altogether, but that does not seem to be possible and even if it would be, it may be more effective to take a short-cut that involves the risk of a possible misunderstanding and patch things when it actually occurs. It is important, then, to prevent reoccurence of the same misunderstandings. For that we need ways to vary system prompts in ways that address the possible causes of the misunderstanding. Such causes may have to do with the mental state of the user. For instance, an open question like *"How may I help you?"*, can be useful for users who know what kind of services the system can provide. However, users who don't know this may actually say anything, most likely something that the speech recognizer cannot handle. Therefore, we have set up a number of rules that determine when to ask which kind of prompt depending on assumptions about the user. Speech recognition errors can then be used to drop certain assumptions and thus the system will select a prompt that better matches the updated user model.

The user model consists of a number of flags that are maintained for each slot in the hierarchical slot structure. These flags are $WantsToKnow_U$, $KnowsStruct_U$, $KnowsVals_U$ and $CanExpress_U$. The flag $WantsToKnow_U$ determines whether the user wants to get information about a slot. This flag is used to determine in which part of the hierarchical slot structure the user is interested. The $KnowsStruct_U$

flag determines whether the user knows which subslots are relevant. The $KnowsVals_U$ flag determines whether the user knows which values can be assigned to a slot, and the $CanExpress_U$ flag determines whether the user knows how to express the values of the slot in a way that the speech recognizer can handle.

We will now describe the rules that govern the questioning behaviour of the system. Questions are only asked for slots whose value is unknown to the system. Therefore, we define a predicate *known* as follows.

$$known(s) = card(s) = 1$$
$$card(s) = \| \{v \mid possible(s,v)\} \|$$
$$possible(s,v) = consistent\ (I^* \cup \{s=v\})$$

Thus, a slot is known if there is only one possible value for it. An open question for slot *s* will only be asked if we can assume that the user is able to formulate an answer. Since the open question itself does not give any information about which subslots or values the system recognizes, we must assume that the user already knows them. Furthermore, we don't want to ask a question about a slot if one of its subslots is already known. The precondition for open questions is therefore defined as follows.

Preconditions $AskOpen(s)$:
$\neg known(s) \wedge$
$(knowsVals_U(s) \vee knowsStruct_U(s)) \wedge$
$\forall(s' \in subslots(s))\ \neg known(s')$

Alternatives questions supply information about the values that can be recognized by the system. Therefore, the assumption that the user knows these values need not hold. Alternatives questions can only be asked for slots for which the number of values is rather small. The preconditions for alternatives questions are defined as follows.

Preconditions $AskAlts(s, V)$:
$\forall(v \in V)\ possible(s,v) \wedge$
$1 < ||V|| <$ max_alts $\wedge$
$\neg knowsVals_U(s)$

If open questions and alternatives questions fail, we can ask yes/no questions.

Preconditions $AskYN(s,v)$:
$\neg known(s) \wedge$
$possible(s,v) \wedge$
$1 < card(s) <$ max_alts $\wedge$
$\neg canExpress_U(s)$

The user model is updated when the system provides information and when errors occur. Whenever a question is asked about a slot, then the *knowsStruct* flag of the superslot is set. And when an alternatives question is asked, the *knowsVals* flag is set. When errors occur after an open question, then the *knowsStruct* and *knowsVals* flags are reset. When errors occur after an alternatives question, the *canExpress* flag is reset.

5. CONCLUSIONS

The hierarchical slot structure allows a clear separation of concerns, which is essential in dialogue design. It allows for the reuse of dialogue strategies in different domains. It adds onto traditional slot filling dialogues a dimension that can be exploited to regulate initiative, and to adapt system behavior in cases of difficulties. The variation in system prompts is important to adapt to different epistemic states that a user can be in.

ACKNOWLEDGEMENT

This research is carried out within the framework of the Priority Programme Language and Speech Technology (TST). The TST-Programme is sponsored by NWO (Dutch Organisation for Scientific Research).

REFERENCES

[1] B.J. Grosz and C.J. Sidner (1986), "Attention, Intentions, and the Structure of Discourse", in Computational Linguistics, Vol. 12, nr. 3, July-September, pp. 175–204.

[2] David R. Traum (1994), "A computational Theory of Grounding in Natural Language Conversation", Ph.D. Thesis, Univ. of Rochester, Rochester, New York, pp. 1–195.

[3] A. Sanderman, J. Sturm, E. den Os, L. Boves and A. Cremers (1998), "Evaluation of the Dutch Train Timetable Information System developed in the ARISE project. in: *Proceedings of IVTTA'98: IEEE 4th Workshop Interactive Voice Technology for Telecommunications Applications*, Turin, Italy, September 29–30, pp.91-96, ISBN 0-7803-5028-6.

[4] H. Grice (1975), "Logic and Conversation", in: P. Cole and J.L. Morgan (eds), *Syntax and Semantics 3: Speech Acts*, New York: Academic Press, pp. 41–58.

[5] G.E. Veldhuijzen van Zanten (1998), "Adaptive Mixed-Initiative Dialogue Management", in: *Proceedings of IVTTA'98: IEEE 4th Workshop Interactive Voice Technology for Telecommunications Applications*, Turin, Italy, September 29–30, pp. 65–70, ISBN 0-7803-5028-6

[6] G.E. Veldhuijzen van Zanten (1996), "Pragmatic Interpretation and Dialogue Management in Spoken-Language Systems", In: S. LuperFoy, A. Nijholt and G.E. Veldhuijzen van Zanten (eds), *TWLT11: Dialogue Management in Natural Language Systems*, Proceedings of the Twente Workshop on Language Technology 11, pp. 81–88.