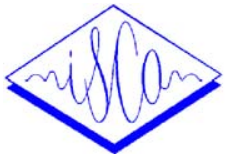

EUROSPEECH '97
5th European Conference on Speech
Communication and Technology
Rhodes, Greece, September 22-25, 1997

GENERATING SEMANTICALLY CONSISTENT INPUTS TO A DIALOG MANAGER

Alicia Abella and Allen L. Gorin

AT&T Labs Research
180 Park Ave. Florham Park, NJ 07932
{abella,algor}@research.att.com

ABSTRACT

Spoken dialog systems interpret a user's request and engage in conversation if the need arises. It is the responsibility of the dialog manager to determine if this need is present and how to proceed. Our spoken dialog system is constructed to sufficiently understand a user's response to the open-ended prompt *'How may I help you?'* in order to route a caller to an appropriate destination, with subsequent processing for information retrieval or call completion. In this paper we describe how to structure the relationships among the call types into an inheritance hierarchy. We then describe an algorithm which exploits this hierarchy and the output of a spoken language understanding module to generate a set of semantically consistent inputs.

1. INTRODUCTION

There exists a large number of spoken dialog systems whose dialog managers differ in the strategies they employ, in the way they represent and manipulate task knowledge and in how much initiative they take. Examples of such systems include [5], [2], [10] and [7]. Our dialog manager employs general dialog strategies to engage a user in a dialog [1], utilizes object-oriented paradigms for representing task knowledge and applies an optimization algorithm for solving the problem of determining a semantically consistent set of inputs. It is often the case that a user's utterance needs clarification either because the user has supplied too much information or not enough. In order to exhibit an appropriate behavior it is important for the dialog manager to infer the user's intention, detect the presence of ambiguities based on the task knowledge and construct a semantically consistent representation of the user's intention. We offer an exact algorithm for solving this problem. A similar problem is addressed by [7] but with a heuristic solution.

Our spoken dialog system is constructed to understand the response to the open-ended prompt *'How may I help you?'*. An example dialog exchange with the system is

User: I seemed to have dialed a wrong number can you give me credit?
System: What was the number that you dialed?
User: 555 4456
System: Was the call billed to the phone you're calling from now?
User: No.
System: How was the call billed?
User: To my credit card.
System: May I have your card number, please?
User: 0000 1234 5678 0000
System: I've given you credit for that call.

Many user interface issues arise when designing such unconstrained systems, a discussion of this can be found in [3].

In the next section we introduce an inheritance hierarchy of call-types (services). This hierarchy and the output of the Spoken Language Understanding (SLU) module [4][9] is used by the dialog manager to generate a semantically consistent combination of call types in the form of a Boolean formula. The Boolean formula is then used by the dialog manager to ask clarifying questions, ask for further information required to complete the transaction and determine in what order to ask such questions. This formula is attained through a Boolean formula minimization process and always yields a minimal and consistent combination of call types. Furthermore, a change in the hierarchy does not require changing the dialog manager. This algorithm together with a set of general purpose dialog strategies constitutes our dialog manager.

2. TASK REPRESENTATION

Our task requires that the system classify a user's call into one (or more) call types out of 15 (a subset is illustrated in figure 1) or send the call to a human agent (OTHER).

Each call type is represented as a class. Each class may have attributes associated with it that describe the class. These attributes may be specific to the class or may be inherited from other classes. Fig-

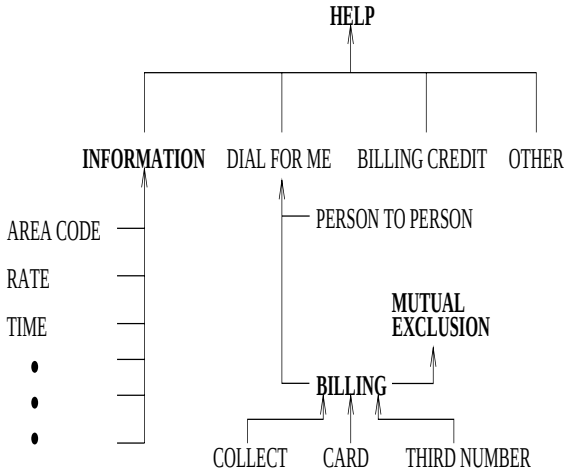

Figure 1: Inheritance hierarchy for a subset of call types.

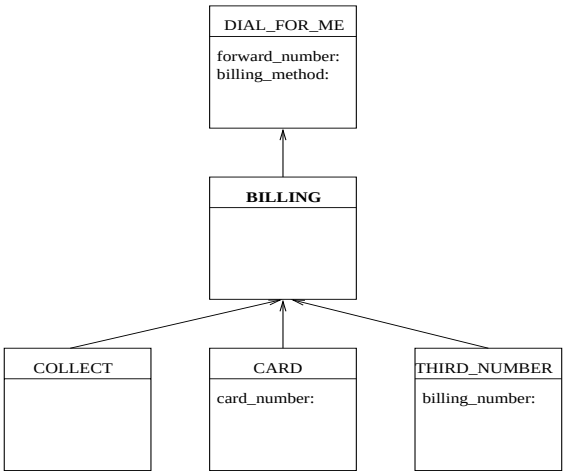

Figure 2: Examples of call types and their associated attributes.

ure 2 illustrates an example of several call types DIAL FOR ME, BILLING, CARD, COLLECT, and THIRD NUMBER. Since DIAL FOR ME is an ancestor class of BILLING and BILLING is in turn an ancestor class of CARD, COLLECT, and THIRD NUMBER they each inherit the *forward_number* attribute from DIAL FOR ME. In addition, CARD has *card_number* and THIRD NUMBER has *billing_number*. Since COLLECT only requires the *forward_number* it needs no additional attributes.

The relationship among the call types and auxiliary concepts (shown in boldface) is illustrated in figure 1. The 15 call types are subdivided into three categories, INFORMATION, DIAL FOR ME, BILLING CREDIT and the additional OTHER. The system described in [4] does not take advantage of the relationships among call types when engaging the user in a dialog. This new approach utilizes this hierarchy to generate the semantically consistent combination of

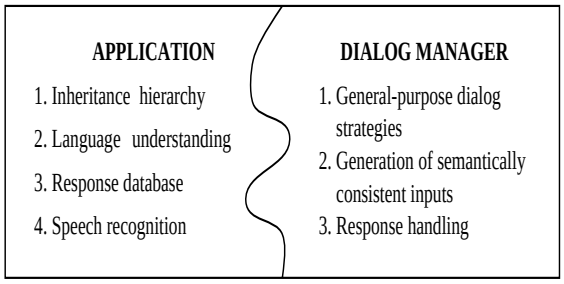

Figure 3: Separation of application specific components from the generic components.

call types that are representative of the caller's request.

Figure 3 illustrates the dialog manager architecture. The dialog manager itself consists of general purpose dialog strategies as described in [1], the algorithm for generating the set of semantically consistent inputs, and response handling. Response handling entails accessing the proper prompts to play back to the user. The application specific components utilized by the dialog manager include the inheritance hierarchy, the spoken language understanding module ([9]), the response database, which contains the prompts to play and the speech recognition module ([6]).

In the next section we will describe how the dialog manager exploits the inheritance hierarchy and task knowledge using several examples.

3. EXAMPLES OF DIALOG BEHAVIOR

Currently a user can speak a request fluently and the recognizer passes its result to the SLU module which produces a set of call types with the ranked probabilities that a caller wanted each of the various services. The following examples illustrate the dialog manager's utilization of the inheritance hierarchy.

Conjunction of call types

User: I would like to make a person to person call to Tokyo but can you tell me what time it is there now?
SLU output: {(PERSON TO PERSON,0.97),
(TIME,1.00)}
Boolean formula: TIME **AND** PERSON TO PERSON
System: It is currently 11AM. How would you like to bill this call?

In this example the dialog manager examines the inheritance hierarchy to find that the two call types returned by the SLU module belong to two unrelated branches of the inheritance hierarchy, which implies that the caller wishes the system to fulfill both of its requests. Each call type has associated with it a priority. Informational requests have the highest pri-

ority and are therefore satisfied first. Because of this, TIME, which is a descendant of INFORMATION is processed first. The dialog manager then proceeds to process the PERSON TO PERSON request which first requires eliciting the billing method.

Mutually exclusive call types

User: I would like to charge this call.
SLU output: {(THIRD NUMBER,0.75),
(CARD,0.24),}[1]
Boolean formula: THIRD NUMBER **OR** CARD
System: How would you like to bill this call?

In this example it is necessary for the dialog manager to initiate a clarification dialog to determine which billing method to apply. Again the dialog manager examines the inheritance hierarchy to find that these call types are mutually exclusive since there can only be one billing method. The relation of mutual exclusion is implemented by introducing the concept of MUTUAL EXCLUSION as an ancestor of BILLING.

Superseding call types

User: I'm trying to make a calling card call can you help me dial it?
SLU output: {(DIAL FOR ME,0.78), (CARD,0.96)}
Boolean formula: CARD
System: May I have your card number, please?
User: 000-123-4567-000
System: What number would you like to call?

In this example, it was not necessary for the dialog manager to ask for the billing method, which would be the case if only the call type DIAL FOR ME had been returned. Instead it can go ahead and ask for the calling card number and phone number because DIAL FOR ME is an ancestor of CARD which implies that CARD supersedes DIAL FOR ME. Each attribute of a call type has an associated priority. This priority is used to determine which attribute to ask about first. For the CARD call type the attribute card number has a higher priority than phone number and therefore is asked about first.

Unsolicited information

User: Yes operator I'm trying to dial 123 4567 but I can't get through.
SLU Output: {(DIAL FOR ME, 0.84)}
Boolean formula: (DIAL FOR ME)
System: Would you like to make a call?
User: Yes.
System: May I have that number again?
User: 555 4567

In this example, it was necessary to verify that the user wanted to make a call since the confidence was not high enough. However, it was not necessary to explicitly ask the user *What number would you like to call?* but rather only to verify the phone number that was provided in the initial utterance.

Context Switch

User: Um uh can I make a collect call?
SLU Output: {(COLLECT, 0.80)}
Boolean formula: (COLLECT)
System: Would you like to make a collect call?
User: No no charge it to my calling card.
SLU Output: {(CARD, 0.85)}
Boolean formula: (CARD)
System: Would you like to make a card call?
User: Yes.

It is very useful for the system to be able to handle a context switch especially if the user has made a mistake and wishes to repair it, which is the case in this example.

In the next section we will describe the algorithm for generating the Boolean formula.

4. THE ALGORITHM

The Boolean formula minimization algorithm for generating the minimal and consistent combination of call types eliminates the often complex coding of *if ... then* rules that often accompany dialog managers. Our dialog manager is data driven, we provide it with the inheritance hierarchy and it applies the Boolean formula minimization algorithm. If we change the inheritance hierarchy we do not have to create a new dialog manager. Another advantage of this data driven approach is in its ability to facilitate portability to other applications. To use the dialog manager for a different application requires creating a new inheritance hierarchy but not a new dialog manager.

We will describe the algorithm for generating the Boolean formula via an example:

User: I would like to charge a person to person call.
SLU Output: {(THIRD NUMBER,0.67),
(CARD,0.25), (PERSON TO PERSON,0.97)}[2]
Boolean formula: (PERSON TO PERSON **AND** THIRD NUMBER) **OR** (PERSON TO PERSON **AND** CARD)
System: How would you like to bill this call?

From the inheritance hierarchy, the system knows that CARD and THIRD NUMBER are mutually exclusive and that PERSON TO PERSON can be performed in conjunction with either of these two call types. As a result, the semantically consistent combination of call types is (PERSON TO PERSON **AND** THIRD NUMBER) **OR** (PERSON TO PERSON **AND** CARD). The dialog manager maps the initial interpretation to an interpretation based on the inheritance hierarchy and uses this to determine which of its dialog strategies to apply. The initial interpre-

[1] For this particular request the SLU module does not return the COLLECT concept because it has a very low probability.

A \ BC	00	01	11	10
0	000	001	**111**	110
	000	001	011	010
1	110	**111**	**111**	110
	100	101	111	110

Figure 4: Initial bit strings representing the values of the variables for each cell (shaded) and the result of applying the transformation $A = B = A \wedge B$ (unshaded).

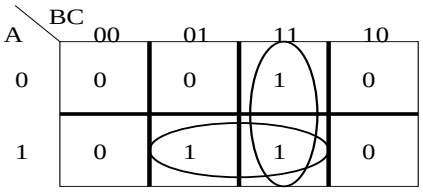

Figure 5: Result of applying Boolean formula minimization. (A **AND** C) **OR** (B **AND** C)

tation, which is PERSON TO PERSON **AND** THIRD NUMBER **AND** CARD is mapped into an ambiguous interpretation based on the inheritance hierarchy since THIRD NUMBER and CARD can not coincide. The dialog manager then knows to apply a disambiguation strategy to prompt the user to specify the billing method. Unlike an example that would require eliciting missing information this scenario is an ambiguous one because the SLU module has provided two billing methods, as opposed to none.

In general, the semantically consistent combination of call types can be found using Boolean formula minimization. We will illustrate the algorithm using the Karnaugh map method for Boolean formula minimization [8]. We will begin by abstracting the call types from this example to A, B and C, where A and B are mutually exclusive. Because A and B are mutually exclusive the resulting formula should not contain A **AND** B. Utilizing this fact, we fill up the Karnaugh map in such a way that the minimization yields a formula that does not contain A **AND** B. This is done using the following algorithm:

Input: SLU output, inheritance hierarchy
Output: Boolean formula

Step–1 Create a Karnaugh Map whose cells each contains a bit string representing the values of the variables for that cell (shown in the shaded region of figure 4).

Step–2 For all variables that are mutually exclusive (A and B in this example) apply the transformation $A = B = A \vee B$. The result of this transformation is shown in the unshaded region of figure 4.

Step–3 For all unshaded cells of figure 4 evaluate $X = A \wedge B \wedge C$ to yield the 1s and 0s used in the standard Karnaugh Map.

Step–4 Perform Boolean formula minimization on X via the Karnaugh Map method [8] as illustrated in figure 5.

If we apply this algorithm on our example the output is the Boolean formula (A **AND** C) **OR** (B **AND** C). The dialog manager then uses this formula to determine that it needs to ask a clarifying question.

5. CONCLUSION

This paper presents a method for determining a set of semantically consistent call types. Task knowledge is represented in an inheritance hierarchy and relations (superseding and mutual exclusion) are defined between call types. These relationships govern the generation of the final set of semantically consistent call types. This set is generated using a Boolean formula minimization algorithm. It is this final set of call types that governs the dialog behavior.

6. REFERENCES

[1] A. Abella, M. K. Brown, and B. Buntschuh. Development principles for dialog-based interfaces. *European Conference on Artificial Intelligence*, 1996.

[2] S. Bennacef, L. Devillers, S. Rosset, and L. Lamel. Dialog in the railtel telephone-based system. *International Conference on Spoken Language Processing*, 1996.

[3] Susan Boyce and Allen L. Gorin. Designing user interfaces for spoken dialog systems. *Proc. Intl. Symposium Spoken Dialog (ISSD)*, Oct. 1996.

[4] A.L. Gorin, G. Riccardi, and J.H. Wright. How may I help you? *Speech Communciation*, to appear.

[5] Helen Meng and Senis Busayapongchai et. al. Wheels: A conversational system in the automobile classifieds domain. *International Conference on Spoken Language Processing*, 1996.

[6] G. Riccardi, A. L. Gorin, A. Ljolje, and M. Riley. A spoken language system for aumated call routing. *Proc. ICASP Munich, Germany*, pages 1143–1146, 1997.

[7] M.D. Sadek, A. Ferrieux, A. Cozannet, P. Bretier, F. Panaget, and J. Simonin. Effective human-computer cooperative spoken dialogue: the ags demonstrator. *International Conference on Spoken Language Processing*, 1996.

[8] Stephen H. Unger. *The essence of logic circuits.* Prentice Hall, 1989.

[9] J.H. Wright, A. L. Gorin, and G. Riccardi. Automatic acquisition of salient grammar fragments for call-type classification. *Eurospeech*, 1997.

[10] S.J. Young and C.E. Proctor. The design and implementation of dialogue control in voice-operated database inquiry systems. *Computer Speech and Language*, pages 329–353, 1989.