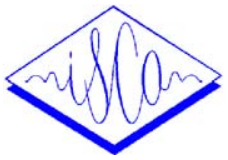

6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

FREE-FLOW DIALOG MANAGEMENT USING FORMS

K. A. Papineni *S. Roukos* *R. T. Ward*
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA

10.21437/Eurospeech.1999-364

ABSTRACT

This paper describes a task oriented, mixed initiative dialog manager. To perform a task according to a user's request, the dialog manager needs many pieces of information. It may have to query the user for missing information, clarify ambiguous information, inherit information from context, and confirm information before performing critical tasks. We describe a form-based framework for free-flow dialog management that deals with these issues. Here, a form corresponds to a specific task in the domain. To allow users to address any task any time, the dialog manager determines the best form that corresponds to the user's intention on every turn. Our dialog manager associates a score to each form as a measure of matching between the form and the user's input. Then the best scoring form corresponds to user's intention.

1. INTRODUCTION

Task-oriented dialog managers perform tasks according to a user's request. In performing a task, the computer usually needs many pieces of information. Often users do not supply all these pieces in a single turn of the conversation or may supply ambiguous information. They may refer to information latent in the context. They may want explicit confirmation before the computer performs critical tasks. There are many common domain-independent issues such as querying for missing information, clarifying ambiguity, inheriting information from context, and confirmation of critical tasks. Further common issues include giving context-sensitive help to the user, handling cancellation requests, handling such events as silence and speech recognition errors.

We describe a form-based framework for free-flow dialog management that deals with all the above issues. Here, a form corresponds to a specific task in the domain; a collection of forms correspond to the entire application. This framework allows users to address any task any time. This means the dialog manager does not know a priori which form corresponds to the user's turn (utterance). Thus, the principal job of the dialog manager is to determine the best form that corresponds to the user's intention.

We model user intention at every turn. That is, the dialog manager does not assume the user continues to address a single task until it is finished or canceled explicitly, unlike machine-initiative dialog managers. This implies the dialog manager always evaluates the suitability of each task to user's input. We do this by associating a score to each form as a measure of matching between the form and the user's input. Then the best scoring form corresponds to user's intention.

Our approach also allows an application developer to prescribe the behavior of the dialog manager and hence the application itself in a simple declarative manner in a text file ("script"). The script, read by the dialog manager at run time, contains the specification of all the forms in the domain. This file determines the functionality of the entire application.

Since the dialog manager itself is application-blind, the application-specific knowledge is assumed to be in the backend. The script specifies the interaction between the dialog manager and the backend in terms of messages to be sent to the backend and actions to be performed depending on the backend response. In this way, we sepa-

rate the process of managing the dialog from the backend transactions themselves.

A powerful feature of our dialog manager is its ability to dynamically switch between a traditional machine-initiative mode and a free-flow mixed-initiative mode. Our dialog manager can be started in either mode. Whenever the user appears to need more hand-holding, the dialog manager can switch to machine-initiative mode. It reverts to the previous free-flow mode when hand-holding is no longer necessary. Maintaining and modifying a list of currently admissible forms is the principal mechanism by which we adapt the dialog mode dynamically.

In our framework, the state of the dialog is not modeled explicitly. For a state-based approach, see [1]. For a form-based approach to a single-task application, see [2]. One of the first single-task systems based on frames is [3]. For an information-based approach using typed feature structures, see [4].

In this paper, we illustrate the dialog manager in a telephony application. We discuss the dialog manager, form structure, and a heuristic scoring metric in this context.

In Section 2, we briefly illustrate the architecture. In Section 3, we discuss the structure of the forms. In Section 4, we discuss the scoring measure.

2. ARCHITECTURE

While the dialog manager of this paper does not assume the specific form of human input (handwriting, speech, typed text) or the computer output (graphical display, synthesized speech), we describe it in the context of a telephony conversational system. Figure 1. describes the architecture of the system.

The square boxes in the figure represent application-specific models or files that their corresponding engines need. The overall system is described in more detail in [5].

3. STRUCTURE OF FORMS

The dialog manager's role is to assist users in performing tasks in an application domain. Buying a stock, reserving a flight, dialing a person's pager are examples of tasks. To perform a task, the dialog manager usually requires many pieces of information. For example, the dialog manager needs to know the name of the stock and dollar amount or number of shares to purchase a stock. A task corresponds to a form. Forms contain slots that correspond to the required pieces of information. For example, a BUY form contains at least two slots: STOCKNAME and AMOUNT.

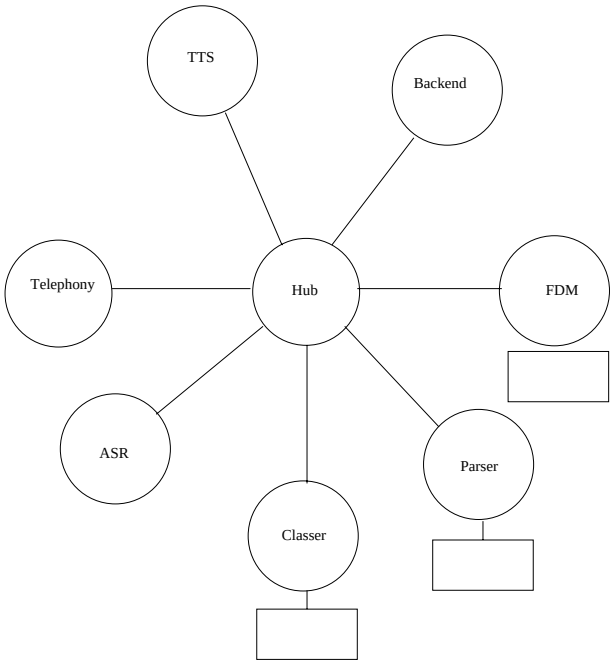

Figure 1. Architecture

The dialog manager is turn-based. Users may not supply all the information needed to perform a task in one turn. In such cases, the dialog manager has to prompt for the missing information. Therefore, a slot contains prompt messages that should be sent to users when the slot is empty.

Similarly, users may supply information that is ambiguous. The dialog manager first has to recognize that the information is ambiguous and then send a message to the user to clarify the information. Whether a particular piece of information is ambiguous or not depends on the application dynamically. For instance, "sell my growth fund" is ambiguous if the user has more than one growth fund, but is clear otherwise. A user's portfolio of funds can change dynamically between turns. Therefore filling some slots may involve consultation with a backend application that connects to a database.

Those slots which require backend consultation for filling also contain messages that should be sent to the backend for disambiguation. They also have clarification messages to be sent to the user. In summary, a slot is associated with several messages.

Once all the necessary slots are filled, the dialog manager can attempt to perform the task. The dialog manager does this by sending a message to the backend with appropriate information from the slots of the form. The dialog manager then sends status report corresponding to the return status from the backend. Therefore, forms contain, apart from slots, a number of messages that are sent either to the backend or to the user depending on the context.

A form represents a task that the user intends to perform. The user's intention is encoded in the semantic representation of the utterance. Clearly there must be a correspondence between the semantic representation and the forms. The semantic representation of the utterance from the parser is simplified to make a list of attribute-value pairs. These attribute-value pairs become the input the dialog manager. The attributes correspond to slots in a form, and their values correspond to fillers of the slots. That is, a slot is filled with a value of an attribute that is declared to match the slot. The application developer prescribes what attributes are allowed to match a slot of a form. So, the minimal specification of a form contains member slots of the form, matching attributes for each slot and slot-level and form-level messages as described above.

The following is a simple example of a form:

```
\begin{form} BUY
 \slot AMT ^MatchedBy: amount
  \begin{messages}
   \msg Prompt: "buy how much?"
  \end{messages}

 \slot FUND ^MatchedBy: fund-buy
  \begin{messages}
   \msg Prompt: "buy which fund?"
   \msg BEMsg: "BEProcess DISAMB $FUND "
    \begin{rclist}
     \rc NONE \msg Prompt: "no fund.."
     \rc MANY \msg Prompt: "many funds..."
    \end{rclist}
  \end{messages}

\begin{messages}
 \msg Help: "a fund name and an amount..."
 \msg Cancel: "canceling purchase..''
 \msg Confirm: "buy $AMT of $BUY. okay?"
 \msg BEMsg: "BEProcess BUY $FUND $AMT"
  \begin{rclist}
   \rc OK \msg Prompt:"completed buying.."
   \rc LC \msg Prompt:"low cash balance.."
   \rc ERROR \msg Prompt:"unable to buy.."
  \end{rclist}
\end{messages}
```

Notice that the messages are simply templates into which variables or functions can be interpolated. The functions may create only side effects or generate text or do both. By writing appropriate messages, the application programmer can not only prescribe what will be presented to the user or another component of the conversational system, but also control what forms will be enabled or disabled or cleared upon sending the message.

The messages are labeled messages. The dialog manager selects a message depending on its label and the context. For instance, a form-level message labeled "Help" will be selected by the dialog manager when the user requests for help on the task corresponding to that form. The specification of the task in a help request may be implicit or explicit. Similarly, a slot-level message labeled "Prompt" will be selected by the dialog manager when the dialog manager decides that the slot's value must be elicited from the user.

Backend messages (slot-level or form-level messages whose target is the backend) encode the task to be performed by the backend. The backend tries to perform the task described in the message, and returns with a return code. Each backend message is therefore associated with a set of return-codes. Each return code is associated with an optional list of forms to be enabled (made admissible), an optional list of forms to be disabled (made inadmissible), an optional list of forms/slots to be cleared, and a message-to-user reporting the status back to the user. In essence,

when the dialog manager generates the rc message, the state of the dialog manager can change in a way the application developer prescribes.

Each form described in the script is a template used by the dialog manager to dynamically create (instantiate) objects storing relevant information about the task that the form corresponds to. At the time of initialization, the dialog manager creates one object per form in the script. The collection of forms thus created becomes the "current set of forms".

The dialog manager also maintains a list of the forms that are currently admissible. The initial list of admissible forms is specified in the script. For example, users cannot transfer money between their accounts before they log in. The script corresponding to any banking application is expected to have only one form, namely the LOGIN form, that is initially admissible. Other forms in the application such as BALANCE-QUERY, WITHDRAW, TRANSFER forms are initially inadmissible. The application developer has control over not only what is initially admissible, but also over what tasks are admissible at various stages of dialog: for example, the application developer can specify that BALANCE-QUERY, WITHDRAW, TRANSFER become admissible after LOGIN form is successfully completed.

The list of admissible forms also plays a central role in dynamically adapting the mode of dialog (machine-initiative/directed-dialog versus mixed-initiative/free-flow dialog) to the flow of conversation. By suitably expanding or contracting the list of admissible forms, the dialog manager can operate in machine-initiative mode or in mixed-initiative mode.

4. SCORE OF A FORM

The input to the dialog manager is a list of attribute-value pairs. Given these pairs, we must determine the most suitable form among the set of admissible forms. Given a partially filled form, our task is to associate a score to the form with respect to the list of attribute-value pairs. We first try to match the attributes to the slots in the form. Let m and n be the number of attributes that do and do not match a slots of the form, respectively. After matching, let u (f) be the number of slots that are unfilled (filled) in the form. A heuristic score is a function of m, n, f, u. We offer one example below:

$$m - n + f - u$$

REFERENCES

[1] E. Levin, R. Pieraccini, and W. Eckert, "Using Markov Decision Process for Learning Dialogue Strategies", Proceedings of the International Conference on Acoustics, Speech, Signal Processing, Seattle, May 1998.

[2] D. Goddeau et al, "A Form-based Dialog Manager for Spoken Language Applications," Proceedings of the International Conference on Spoken Language Processing, 1995, pp. 701-704.

[3] Bobrow et al, "GUS: A Frame-driven Dialog Manager," Artificial Intelligence, vol 8 (1977), pp 155-173.

[4] Matthias Denecke and Alex Waibel, "Dialogue Strategies Guiding Users to their Communicative Goals", Proceedings of Eurospeech-97, Rhodes, Greece, 1998, pp. 1339-1342.

[5] K. Davies et al, "The IBM conversational telephony system for financial applications", this proceedings.