Dialogue Management in the Mercury Flight Reservation System

Stephanie Seneff and Joseph Polifroni
Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139
seneff,joe@sls.lcs.mit.edu

Abstract

This paper describes the dialogue module of the Mercury system, which has been under development over the past year or two. Mercury provides telephone access to an on-line flight database, and allows users to plan and price itineraries between major airports worldwide. The main focus of this paper is the dialogue control strategy, which is based on a set of ordered rules as a mechanism to manage complex dialogue interactions. The paper also describes the interactions between the dialogue component and the other servers of the system, mediated via a central hub. We evaluated the system on 49 dialogues from users booking real flights, and report on a number of quantitative measures of the dialogue interaction.

1 Introduction

Dialogue modeling is a critical and challenging aspect of conversational systems, particularly when users are permitted flexibility with regard to defining the constraints of the task. For systems that adopt a strict system-initiated approach, it is feasible to define a set of states and state transitions depending on the usually small number of possible user actions at each state. However, if the user is permitted to say anything within the scope of the recognizer at any time, such a finite-state solution becomes unwieldy. We are interested in the development of mixed-initiative systems, where the system may make specific requests or suggestions, but the user is not required to be compliant. Instead of a finite state dialogue model, we choose to decompose dialogue state into a set of state variables. The activities for a given turn typically involve the sequential execution of a number of specialized routines, each of which performs a specific part of the dialogue requirements and alters the state variables in particular ways. To determine which of the operations should be performed, the system consults a dialogue control table, which is specified in a simple scripting language.

This paper describes experiments with using this approach to dialogue modeling in the context of our Mercury flight reservation system. Mercury allows users to plan air travel between 226 cities worldwide. Following log-on, the user interacts with the system to select the flights of their trip. When the flight plan is completed, the system takes the initiative to offer to price and email the itinerary. Finally, the system asks the user a few questions to help determine user satisfaction.

The overall system makes use of the GALAXY architecture [Seneffet al (1999)], which consists of a number of specialized servers that communicate with one another via a central programmable hub. An audio server captures the user's speech via a Dialogic board, and transmits the waveform to the speech recognizer [Glass *et al* (1996)]. The language understanding component [Seneff (1992)] parses a word graph produced by the recognizer and delivers a semantic frame, encoding the meaning of the utterance, to the discourse component. The output of the discourse component [Seneff (1996)] is the frame-in-context, which is transformed into a flattened E-form (electronic form) by the generation server. This E-form is delivered to the turn manager, and provides the initial settings of the dialogue state.

The turn manager consults the dialogue control table to decide which operations to perform, and typically engages in a module-to-module subdialogue to retrieve tables from the database. It prepares a response frame, which may or may not include tabular entries. The response frame is sent to the generation component [Glass (1994)] which transforms it in parallel into both a text string and an annotated string that specifies the input controls for the speech synthesizer. Finally, the speech synthesizer transmits a waveform to the audio server which then relays the spoken response to the user over the telephone. The entire dialogue is recorded in detail in a log file for later examination.

2 Knowledge Representations

Mercury's dialogue manager manipulates linguistic and world knowledge represented in the form of semantic frames. At each turn it begins with an E-form representing the constraints of the current query as a set of (key: value) pairs. This E-form pro-

U:	John Doe
S:	Please speak the month and day you registered as your password, as in, may first.
U:	april first
S:	Welcome John. You have successfully logged onto Mercury. How may I help you?
U:	i want to travel to washington dulles on november fourth
S:	From Boston to Washington Dulles on November 4. Can you provide the approximate departure time or airline preference?
U:	i need to arrive at around nine a m
I: S:	Okay. Flights from Boston to Washington Dulles arriving at 9:00 am on November 4. One moment please. The best matching flight is a U S Airways flight arriving at 8:59 a.m. I also have a United flight arriving at 8:36 a.m.
U:	what time does the united flight leave
S:	United flight 1819 departs at 7:00 am from Boston. Shall I book this flight for you?
U:	yes please
S:	United flight 1819 has been added to your itinerary. You depart Boston at 7:00 am and arrive in Washington Dulles at 8:36 am. Where would you like to go next?
U:	i would like to return to boston
S:	What date will you be returning on?
U:	the same day
I: S:	Okay. United flights from Washington Dulles to Boston on November 4. One moment please. Can you provide a departure or arrival time? The earliest nonstop leaves at 9:30 a.m. and arrives at 10:57 a.m.
U:	i would like to leave after three thirty
S:	The best matching flight is a United flight departing at 4:30 p.m. I also have a United flight departing at 5:30 p.m.
U:	i will take the first one
S:	United flight 1278 has been added to your itinerary. You depart Washington Dulles at 4:30 pm and arrive in Boston at 6:07 pm. Shall I price your itinerary?
U:	yes please
I: S:	Pricing your itinerary will take a while. In the meantime, I'll summarize your itinerary for you. on November 4, you depart on United flight 1819 from Boston at 7:00 am and arrive in Washington Dulles at 8:36 am. From there, you depart on United flight 1278 at 4:30 pm and arrive in Boston at 6:07 pm. The price for your itinerary is $249.00. Shall I email your itinerary to you?
U:	yes
S:	Your itinerary has been emailed. Is there something else I can do for you?
U:	no thank you
S:	Thank you for calling! Have a nice day!

Table 1: Example dialogue between a user and the system involving a real trip. "U" denotes a user utterance, "S" a system utterance, and "I" an intermediate system response spoken prior to database retrieval.

vides the initial values in the dialogue state, which evolves over the course of the turn as rules are executed.

In addition to the dialogue state, Mercury also makes use of several other internal representations which retain state. These are packaged up into a turn-manager history frame which is passed to the hub for safe-keeping at the end of each turn, and returned with each subsequent turn. The user model contains a record of the system's current belief about the state of the user's flight plan. For example, if the user's first query were "I want to go to Denver next Friday morning, returning the following Wednesday," the system would record that this is a round trip flight and would save the return date (unresolved, in case there was a recognition error on the forward leg date) in the user model. It would retrieve from the user model, as the forward leg source, the default value provided by the user at enrollment. Thus the user model retains both the initial user preferences and all of the details of the current flight plan as they become available. The system also retains internally a list of all the flights that matched the user's constraint specifications, as well as a list of the subset of the flights that it has spoken about (e.g., the nonstop flights or the earliest

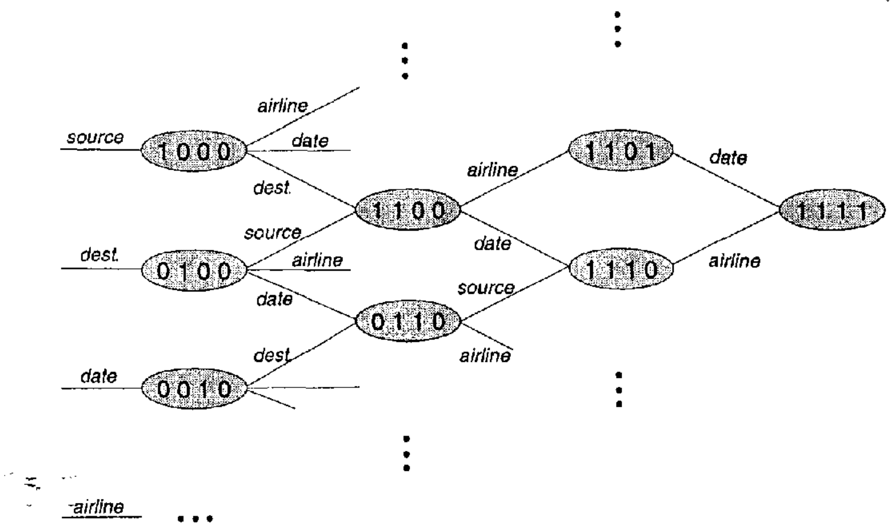

Figure 1: An (incomplete) state diagram for the example system described in the text.

flight). These flights become available for reference in subsequent queries such as "book the third one," or "where does the united flight stop?".

Whenever the system needs to go to the database, it provides an intermediate reply frame which serves both to ask the user for patience and to confirm a consistent shared model of the dialogue history. The system's final response is also a frame, and it often contains a list of flights to be read back to the user. In other cases, it may be a yes-no question as in "Shall I email your itinerary to you?" The latter case also necessitates sending both to the discourse component a system-initiative context for yes/no responses, and to the language understanding component a dialogue context filter that favors confirmations and denials in the N-best selection. A similar process takes place when the system prompts for information such as a date or a departure city.

Table 1 shows a dialogue between a user and the system involving a real trip between Boston and Washington D.C. The name and password information have been omitted for privacy reasons. In the figure, "U" stands for "user", "S" for system, and "I" for intermediate response.

3 Dialogue Management Strategies

A finite state network is a convenient way to formulate dialogue states, as long as the system is able to maintain strong control over the order in which slots are filled, and especially when slots are filled one at a time. However, if the user is permitted the flexibility to fill slots in any order, and to fill more than one slot in a single turn, then it becomes difficult and inefficient to define a finite state network that covers the entire space of possibilities.

A finite state solution for a simple dialogue example is given in Figure 1. Here, we assume that the user can fill four slots (source, destination, date, and airline) and that all four must be filled before the system can retrieve flights. The slots can be filled in any order, and the user can fill any number of slots in each turn. For notational convenience, we represent the states filled/unfilled by 1/0, and we order the fields as [source, destination, date, airline], such that state "1101" says that only date is missing. At each state the system prompts the user for all missing slots. Thus, state 1000 would speak, "Can you provide one or more of destination, date, and airline?" If the user provides more than a single field in a given turn, the system advances by more than one state in the state diagram. To fully specify all possibilities, even for this simple example, requires a large number of states in the state diagram.

!source	-->	prompt_source
!destination	-->	prompt_destination
!airline	-->	prompt_airline
!date	-->	prompt_date
nprompts > 1	-->	mark_multiple
nprompts = 0	-->	retrieve_flights

Figure 2: A dialogue control table corresponding to the state diagram in Figure 2.

Figure 2 shows a set of ordered rules which achieve the same goals as the state diagram but with a greatly simplified dialogue control specification. In this case, state is encoded implicitly as a bit map corresponding to the filled/unfilled values of the four variables. The notation "!" stands for "NOT," meaning that this variable is not yet filled. Following the arrow is the name of the operation to be

```
clause request_keypad                                ⟶  keypad_input
week | day | rel_date                                ⟶  resolve_relative_date
flight_number & !airline                             ⟶  need_airline
clause price & topic itinerary & truth_value deny    ⟶  dont_price_itinerary
clause book & num_found =1                           ⟶  add_flight_to_itinerary
num_nonstops > 1 & num_nonstops < 4 & arrival_time   ⟶  speak_three_arrivals
```

Figure 3: Selected rules from the Mercury dialogue control table. "&" stands for "AND" and "|" stands for "OR".

performed when the conditions are met. Each operation is usually a specific function, and is free to augment the variable space by either altering the values of pre-existing variables or introducing new variables. In our simple example the four prompt operations simply add their respective variables to the response frame and increment the variable "nprompt." The final function "mark_multiple" fires only if more than one of the preceding functions have fired. Its action is to insert "one or more of" into the response. The final "retrieve" operation only fires if none of the preceding operations fired. It is significant that state is dispersed not only over the variables but also over the operations. Thus, any single prompt state of the state transition matrix would involve up to two operations in the dialogue control table, and, conversely, the "mark_multiple" operation is associated with several states.

In addition to possibly altering the values of variables in the dialogue state, operations also have the responsibility to return, upon completion, one of three "move" states. The majority of the operations return the state "CONTINUE," which means simply to continue execution just beyond the rule that just fired. The two other possibilities are "STOP," i.e., exit from the dialogue control table, and "RESTART" which means to return to the top of the set of dialogue rules and start over. An (optional) distinguished operation, which handles activities that should occur routinely at every turn, is executed upon exiting the dialogue control process. The tests on variables can be binary, arithmetic, or string matching on values.

Our Mercury system makes use of this "ordered rules" strategy for dialogue control. However, the rules are dealing with a great deal more complexity than simply deciding what prompts to issue when certain slots are unfilled. There are currently a total of over 200 rules in Mercury's dialogue control table. These rules can be grouped into several distinct categories. Only nine of the rules involve prompts for missing information. Another nine are involved with logging into the system, i.e., acquiring the name and the password, which may have to be entered using the telephone keypad, either upon user request or as a consequence of recognition failure. Eleven rules are concerned with meta-level interactions such as apologies for missing services (no flight status information available), and requests for help or repetition. Several rules have to do with determining whether the current request plausibly involves a reference to a flight in a pre-existing flight list. This could be an obvious reference to the *nth* flight, or it could be an inquiry about the "united flight" or the "nine a.m. flight." Several more rules involve interpreting various references to relative dates and/or times such as "the following Thursday," or "that same afternoon."

The largest single category concerns preparing the reply frame, after the database retrieval has already taken place. Twenty six rules are concerned with this task, and they are keyed on a combination of the number and type of flights retrieved and the specific request of the user (e.g., "where do they connect?"). The purpose of these functions is to reduce the number of database tuples returned for a given query to a manageable set for a spoken response. This reduction is based on a hierarchy of quality measures for each flight, beginning with any stated or inferred preference (e.g., a particular airport in the case of cities with multiple airports, or a particular airline in the case of a multi-leg booking where one leg has already been established) and including number of stops and length of flight. These functions also consolidate information about the flights to be spoken, combining shared attributes. Our ultimate goal is to produce a response such as "I have found three nonstop United flights. Flight 100 leaves at 8:45, flight 200 leaves at 9:10, and flight 300 leaves at 9:30." The dialogue control table facilitates this interaction by enabling the system developer to encode the constraints of the hierarchy in the rules.

Finally, there are a miscellaneous set of rules that have to do with updating the user model, preparing the intermediate reply, pricing or emailing the itinerary, preparing the database query, filtering flights, or updating the itinerary.

Since the operations are general functions, it is up to the system developer to decide how to parcel up the computational requirements into the individual operations. With experience, one acquires a set of guidelines to help formalize this process. As a general rule, it is preferrable to limit the use of nested function calls. Instead, an operation can set a vari-

WER	words/turn	total turns	total time
11.5%	7	11	229 sec.

Figure 4: Some easily computed statistics on the 36 successful bookings.

able to indicate that another operation needs to be called, and then the intended subroutine gets promoted to the status of a dialogue control operation. This has the effect of exposing more of the internal activities of the system to the dialogue control table, which serves as a very useful outline of system control flow. Another general policy is that each operation should be concerned with a single well-defined task, again in order not to conceal complexity.

Figure 3 shows several examples of actual rules in Mercury's dialogue control table.

4 Data Collection and Evaluation

Mercury first became available for data collection in October '99. Prospective users must first enroll by filling in a simple form on a Web page, where they enter, minimally, their name, email address, and password (a date). Once the user's name has been added to the recognizer and language understanding components, they receive an email message informing them of the telephone number. Users are encouraged to attempt to book real trips. From late October to early December, we collected 49 dialogues involving real flight bookings, and these form the basis for our evaluation studies.

Overall, 73% of the bookings were successful (36/49). We used a very strict requirement for success. For example, in one dialogue considered unsuccessful the system did not know the airline that the user requested, and so the user compromised and booked the trip on a different airline. Three of the failures are due to the user simply hanging up in frustration, and three others are due to the system hanging up due to a misrecognized "good-bye." Two failures were due to user inattentiveness. The user believed that the trip was correctly booked, but a misrecognition produced a different itinerary than the one they were specifying. Finally, four of the failures involved completely correct bookings, but the system was unable to follow through with the pricing and/or emailing of the itinerary. Some of these involved inadequacies in the dialogue module, once the user did not provide the expected response to a system request. There was a striking difference in recognition error between the successful and the incomplete bookings (11.5% vs 26% WER). A heavy foreign accent accounted for some of the recognition problems.

Some easily measurable statistics for the successes are given in Figure 4. These numbers were computed on the "core dialogue," defined as the interval subsequent to logging on and up until the itinerary is fully specified, but has not yet been priced. On average users required less than four minutes to complete the core dialogue, although three outliers took more than seven minutes.

IBR:	0	1	2	3	4	total
Nutts:	41	90	55	31	9	226

Figure 5: Distribution of evaluable user utterances in terms of number of new attributes introduced with each dialogue turn. IBR = Information Bit Rate.

4.1 Log File Evaluation

We have long been interested in seeking evaluation metrics that are automatic and that can apply on a per-utterance basis but evaluate a significant portion of the system beyond the recognizer. In [Polifroni *et al.* (1998)] we proposed an E-form evaluation metric, which compares an E-form obtained by parsing the original orthography against that obtained by parsing the selected recognizer hypothesis. We believe this is a good metric for evaluating how well the recognizer and parser are doing, but it says nothing about the discourse and dialogue components.

We recently devised two new evaluation metrics, which we believe are useful measures for assessing the performance of the recognizer, parser, discourse, and dialogue components, collectively. To compute the measures, we must reprocess the log file after the orthographic transcription has been provided for the user queries. Basically, both the recognizer hypothesis and the original orthography are run through the system utterance by utterance, with the discourse and dialogue states being maintained exclusively by the recognizer branch. For both branches, the E-form that is produced after the turn manager has finished processing the query is sent to a special evaluation server. This server maintains a running record of all the attributes that appear in the orthography path, comparing them against their counterparts in the recognizer path.

The two parameters that emerge from comparing these E-forms we refer to as information bit rate (IBR) and user frustration (UF). IBR measures the average number of new attributes introduced per user query. A subsequent query that reiterates the same attribute is excluded since it did not introduce any new information. Thus if the user said, "I want to go from Seattle to Chicago on December 27," and the system misrecognized the date as "December 22," then a subsequent query, "I said December 27" would be registered as contributing a 0 count to the IBR parameter. The UF parameter tabulates how many turns it took, on average, for an intended attribute to be transmitted successfully to the system. Thus, in the example above, the source and destination each took one turn, but the date took two.

There are some difficulties with rerunning the dialogue at a later time. Both the system and the database are in a state of flux, and so the dialogue can become incoherent. For example, in one case the user said, "Book it," in response to a single flight being proposed, but due to changes in the flight schedule, the system proposed three flights in the rerun and the dialogue became incoherent from that point on. To help alleviate incoherence, we provide a mechanism to artificially offset the date, at least to assure that the dates they have selected haven't already passed. In spite of the above problems, we feel that these evaluation metrics show considerable promise.

In a pilot study, we processed a subset of our data through this evaluation configuration. We identified a set of 17 attributes that could be monitored. Five percent of the utterances had orthographies that failed to parse. These are unevaluable without human reannotation, and are hence eliminated from the pool in the discussion below, although they clearly are likely to be very problematic. Figure 5 summarizes the results for information bit rate for the remainder of the utterances. A surprisingly large percentage of the utterances introduce no new concepts. Some, but not all, of these are similar to the date misrecognition example given above. Others are cases where the user was confused about the state of the system's knowledge, and decided to simply repeat all the preceding constraints just to make sure. Some are also misfirings of the endpoint detector producing content-free utterances such as "okay." In other cases the user intended an action, but the system's understanding mechanism was not sophisticated enough. For example "That's good" meaning "book it." We were pleased with the percentage of sentences that contained more than one attribute. We believe that a typical directed dialogue would have far fewer utterances with more than one attribute.

Excluding the 5% of utterances whose orthography failed to parse, our system achieved a 1.05% user frustration rate. This means that, on average, one out of every 20 attributes had to be entered twice. We were very pleased with this number.

5 Summary and Future Work

This paper described our strategy for dialogue management in the Mercury system. Overall, we have found it to be extremely powerful. While the Mercury system is still under active development, we feel that the anticipated extensions of capability will require a straightforward process of expansions in the dialogue control table; i.e., the system has not attained an unmanageable degree of complexity in organization. We believe that this is a direct consequence of the use of the dialogue control table.

Mercury's turn manager deals with some issues, such as the interpretation of dates and times, that are of relevance to many other potential domains. We envision that at some future time we will have available a large library of operations of general utility that can be inserted into a new system to greatly accelerate the time required to bring the new domain up to full operation.

References

Seneff, S., R. Lau, and J. Polifroni. (1999) "Organization, Communication, and Control in the GALAXY-II Conversational System," *Proc. Eurospeech '99*, Budapest, Hungary, pp. 1271–1274.

Glass, J., J. Chang, and M. McCandless. (1996) "A Probabilistic Framework for Feature-based Speech Recognition," *Proc. ICSLP '96*, Philadelphia, PA, pp. 2277–2280.

Seneff, S. (1992) "TINA: a Natural Language System for Spoken Language Applications," *Computational Linguistics*, 18/1, pp. 61–86.

Seneff, S, D. Goddeau, C. Pao, and J. Polifroni. (1996) "Multimodal Discourse Modelling in a Multi-user Multi-domain Environment," *Proceedings, International Conference on Spoken Language Processing '96*, pp 192-195, Oct. 3-6.

Glass, J., J. Polifroni, & S. Seneff. (1994). "Multilingual Language Generation across Multiple Domains." *Proc. International Conference on Spoken Language Processing* (pp. 983–986). Yokohama.

Polifroni, J., S. Seneff, J. Glass, and T.J. Hazen. (1998) "Evaluation Methodology for a Telephone-based Conversational System." *Proc. LREC '98*, pp. 43–50, Granada, Spain.