



Response planning and generation in the MERCURY flight reservation system[☆]

S. Seneff[†]

MIT Laboratory of Computer Science, Spoken Languages System Group,
200 Technology Square Rm NE43-643, Cambridge, MA 02139, U.S.A.

Abstract

This paper describes the response planning and generation components of the MERCURY flight reservation system, a mixed-initiative spoken dialogue system that supports both voice-only interaction and multi-modal interaction augmenting spoken inputs with typing or clicking at a displayed Web page. MERCURY is configured using the Galaxy Communicator architecture (Seneff, Hurley, Lau, Schmid, & Zue, 1998), where a suite of servers interact via program control mediated by a central hub. Language generation is performed in two steps: response *planning*, or deep-structure generation, is carried out by the dialogue manager, and is well-integrated with other aspects of dialogue control; control flow is specified by a *dialogue control table* (Seneff & Polifroni, 2000a). Response *generation*, or surface-form generation, is executed by a separate language generation server, under the guidance of a set of recursive generation rules and an associated lexicon (Baptist & Seneff, 2000). The generation of the textual string for the graphical interface and the marked-up synthesis string for spoken outputs are controlled by a shared set of generation rules (Seneff & Polifroni, 2000b). Thus there is a direct meaning-to-speech mapping that eliminates the need to analyze linguistic structure for synthesis. To date, we have collected over 25 000 utterances from users interacting with the MERCURY system. We report here on both the results of user satisfaction studies conducted by the National Institute of Standards and Technology (NIST), and on our own tabulation of a number of different measures of dialogue success.

© 2002 Elsevier Science Ltd. All rights reserved.

1. Introduction

The Spoken Language Systems Group at the MIT Laboratory for Computer Science has been developing spoken dialogue systems for over a decade. Through the years, these systems have grown in complexity, but we have also learned to streamline many aspects of system development to help manage complexity. Our systems have been

[☆] The research was supported by DARPA under contract N66001-99-8904 monitored through Naval, Command, Control, and Ocean Surveillance Center.

[†] E-mail: seneff@sls.lcs.mit.edu

developed within the Galaxy Communicator framework (Seneff *et al.*, 1998; Seneff *et al.*, 1999), where a programmable hub coordinates interactions among a suite of servers, each of which specializes in a particular aspect of the problem, such as speech recognition or database access. Wherever possible, we have attempted to equip each server with generic library code, with domain and language dependencies maintained in external tables, rules, and models.

The MERCURY system (Seneff & Polifroni, 2000a), which provides information about flights available for over 500 cities worldwide, is our most sophisticated system to date in terms of its dialogue model. We have invested considerable effort into making MERCURY intuitive to use and robust in handling a wide range of different ways users might express their flight constraints and select the flights of the itinerary. Our goal in doing this research was to demonstrate that, given the current constraints on the state of the art in telephone-based speech understanding, it is possible to design a telephone-access spoken dialogue system that people would be willing to use to plan their air travel. We believe we have reached this goal, in that there now exists a community who make their flight reservations using MERCURY. A user begins by logging on, providing both their name and a password, which allows the system to look up some personalized information such as the e-mail address and the preferred originating city. MERCURY's dialogue plan involves arranging a trip one leg at a time. Once the itinerary is fully specified, MERCURY offers to price the itinerary, and subsequently to send a detailed record of the itinerary to the user via e-mail, which they can then forward to their travel agent for the actual booking.

Users can initiate an interaction with MERCURY by simply calling the system on the telephone.¹ MERCURY is also accessible from a Web page, in which case the spoken interaction is augmented with a tabular display of the set of retrieved flights. The user can type "Call me at (phone number)" in the type-in window, to initiate a multi-modal (voice/GUI) interaction. The interface supports clicking on a displayed flight and referring to it verbally: "Book this flight," as illustrated in Figure 1.

To demonstrate some of the features of the MERCURY system, an actual voice-only dialogue between MERCURY and a user is shown in Figures 2 and 3. After the user had successfully completed the first leg of her trip, the system asked for a return date, assuming that this was a simple round-trip itinerary. The user offered instead the next leg of a multi-leg trip. After the second leg had been booked, the system changed its prompt to, "Where would you like to go next?" At this point, the system mis-recognized the destination as "Denver" instead of "Boston," but the user was able to recover from the system's mistake in the immediate subsequent query. Later on, another mis-recognition caused it to redundantly inform the user of the connection city. At the conclusion of the travel planning process, the system offered to price the itinerary. Again the user chose not to comply, but instead asked about rental cars. The system correctly informed the user that it was unable to deal with hotels and rental cars, and then repeated the pricing request. This time the user agreed, and the dialogue was brought to a close.

The above example shows that, in addition to correctly understanding the user's queries in the proper context, the system must be able to plan and generate the appropriate responses to provide its contributions to the conversation. The response

¹MERCURY's toll-free number is 877-628-8255.

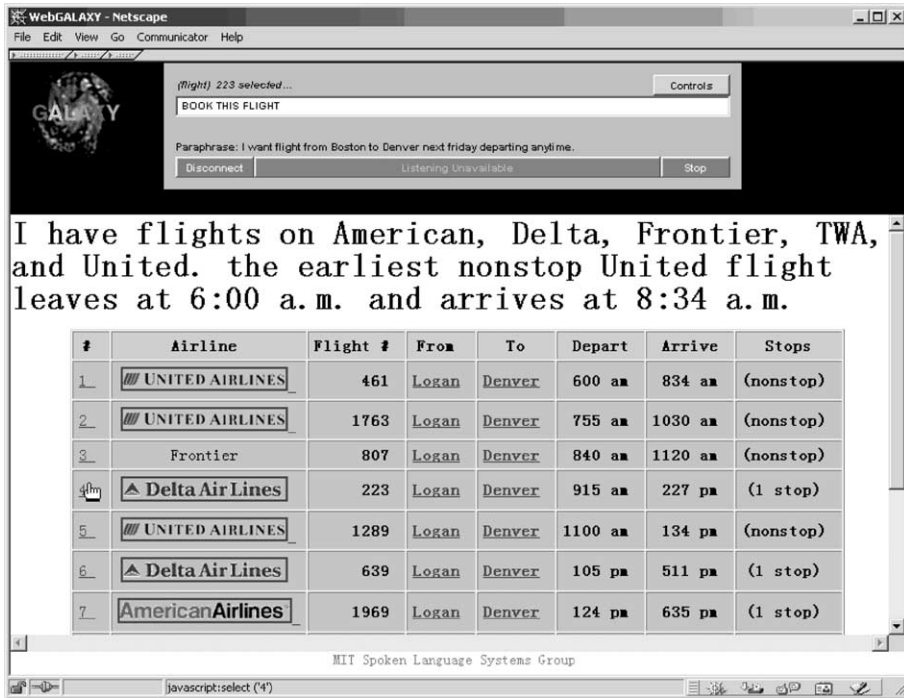


Figure 1. An illustration of the graphical interface to MERCURY. Airline names and source and destination are linked to Web pages for the corresponding airlines and airports. The user has clicked on the fourth flight, and is asking to add it to the itinerary.

might be a simple paraphrase of the input, providing feedback concerning its understanding of the query (see Section 6.5), along with a prompt soliciting further information (see Section 6.1). At other times, the language generation component is tasked with presenting the information it has retrieved in a manner that can easily be absorbed. For applications in which display is not available, for example, the dialogue manager must reduce the information to a summary of the salient facts describing a large list of retrieved database items, in order not to bore or overload the user (see Section 6.2). The system must also offer helpful suggestions at strategic points, to move the conversation towards a productive conclusion (as described in Sections 6.3 and 6.4).

The generation requirements of spoken dialogue systems are quite different from those of traditional text-planning and paragraph-level generation tasks. For instance, generation in dialogue systems must be computationally very efficient, requiring ideally only a small fraction of real-time to completely generate and synthesize the response utterances. Furthermore, in a dialogue involving interaction over the telephone, the generated responses cannot be overly detailed in the information they present, because the user's memory may be inadequate for the task. Finally, part of the responsibility of the generation component is to inform the user of the system's interpretation of their query; it is not sufficient to just provide the data retrieved from the database, when there may have been a major recognition error that has led the system to answer the wrong question.

SYSTEM: Welcome Lindsey. How may I help you?

USER: I would like to leave Boston on November second in the late morning so I can arrive in Detroit Michigan around 2 p.m.

SYSTEM: Okay, flights from Boston to Detroit departing between 10:00 am and noon arriving at 2:00 pm on Friday November second. One moment please... I have 2 nonstop Northwest flights: a flight arriving at 12:52 p.m., and a flight arriving at 2:07 p.m. Would one of these work?

USER: Yes the one that arrives at 2:07 p.m.

SYSTEM: Northwest flight 389 has been added to your itinerary. You depart Boston at 11:55 am and arrive in Detroit at 2:07 pm. What date will you be returning on?

USER: From Detroit I want to go to San Francisco on November sixth.

SYSTEM: Okay, Northwest flights from Detroit to San Francisco on Tuesday November sixth. One moment please... Can you provide a departure or arrival time? The earliest nonstop flight leaves at 9:10 a.m. and arrives at 11:16 a.m.

USER: That would be fine.

SYSTEM: Northwest flight 343 has been added to your itinerary. You depart Detroit at 9:10 am and arrive in San Francisco at 11:16 am. Where would you like to go next?

USER: Then on November eleventh I would go from San Francisco to Boston. [misrecognized]

SYSTEM: Okay, Northwest flights from San Francisco to Denver on Sunday November 11th. One moment please... I couldn't find any flights on Northwest. sorry I could not find flight departing between 5:00 am and 11:00 pm.

USER: I need a flight going to Boston Massachusetts.

SYSTEM: Okay, flights from San Francisco to Boston departing between 5:00 am and 11:00 pm on Sunday November 11th. One moment please... I have flights on American, Sun Country Air, United, and U S Airways. The earliest nonstop United flight leaves at 7:00 a.m. and arrives at 3:36 p.m.

Figure 2. Example dialogue interaction between a user and the MERCURY system: part I. Utterances where the system made recognition errors are indicated by a [misrecognized] tag.

Several research efforts have led to the development of computer generation systems for text-based applications, such as summarization, text-based dialogue systems, translation, and report generation (Chu-Carroll & Carberry, 1998; McDonald & Bolc, 1998; Bangalore & Rambow, 2000; Reiter & Dale, 2000). Generation efforts specifically targeted towards spoken dialogue applications include Baptist and Seneff (2000), Glass, Polifroni, and Seneff (1994), Moore *et al.* (1997), Oh (2000), Oh and Rudnicki (2000), Portele (2000), and Ratnaparkhi (2000). Initially, most of the generation components designed for spoken dialogue systems utilized template-based approaches, which are

USER: Do you have a Northwest flight going from San Francisco to Boston on november eleventh?

SYSTEM: Okay, Northwest flights from San Francisco to Boston on Sunday November 11th. One moment please... Can you provide a departure or arrival time? There are no nonstop flights. There are at least 6 connecting Northwest flights from San Francisco to Boston on Sunday November 11th.

USER: Connecting one would be fine in the late afternoon.

SYSTEM: Okay, connecting Northwest flights from San Francisco to Boston departing between noon and 6:00 pm on Sunday November eleventh. One moment please... I have a flight connecting in Detroit leaving at 12:45 p.m., a flight leaving at 3:25 p.m., and a flight connecting in Minneapolis Saint Paul leaving at 3:40 p.m. Do you like any of these?

USER: A flight leaving at 12:45 p.m. [misrecognized]

SYSTEM: it connects in Detroit.

USER: That would be fine.

SYSTEM: Northwest flight 342 connecting to Northwest flight 248 in Detroit has been added to your itinerary. you depart San Francisco at 12:45 pm and arrive in Boston at 11:10 pm. Would you like me to find out how much your trip will cost?

USER: In Detroit I need a rental car.

SYSTEM: I don't have any information on rental cars. Would you like me to find out how much your trip will cost?

USER: Yes that would be fine.

SYSTEM: Pricing your itinerary will take a while. In the meantime I'll summarize your itinerary for you. On Friday November second on Northwest flight 389, you depart from Boston at 11:55 a.m. and arrive in Detroit at 2:07 pm. On Tuesday November sixth on Northwest flight 343, you depart from Detroit at 9:10 a.m. and arrive in San Francisco at 11:16 a.m. On Sunday November eleventh on Northwest flight 342, you depart from San Francisco at 12:45 pm and arrive in Detroit at 8:08 p.m. From there, on Northwest flight 248 you depart at 9:20 pm and arrive in Boston at 11:10 pm. The cheapest price for your itinerary is \$861.50. Shall I e-mail your itinerary to you?

Figure 3. Example dialogue interaction between a user and the MERCURY system: part II.

very efficient but tend to be specific to a particular domain, and thus less obviously generalizable to other tasks. In many cases, output sentences can be generated as simply predefined word strings, with inserted slot fills, in text or pre-recorded speech form. As applications have grown in complexity, however, such an approach is clearly becoming inadequate. As a consequence, template-based systems have become more sophisticated, and have introduced linguistically motivated mechanisms.

In recent years, researchers have begun to devote more attention to generation aspects of dialogue systems. Two distinct approaches to spoken language generation are

emerging, which can loosely be labeled as template/rule-based and corpus-based. In the former, sentences are typically generated by concatenating substrings based on patterns specified by (typically domain-dependent) rules. Each substring is generated by applying the rules recursively, and syntactic features such as person, gender and number are incorporated in a separate lexicon (Glass *et al.*, 1994; Baptist & Seneff, 2000; Portele, 2000). When the generation capabilities are sufficiently sophisticated, this approach can produce well-formed responses that convey the appropriate information to the user. The research reported here is based on such an approach.

The corpus-based methods Oh (2000), Oh and Rudnicki (2000), and Ratnaparkhi (2000) borrow from research ideas from the speech recognition community, utilizing statistical approaches to support the selection of generated strings, usually on a criterion of low perplexity. Some systems are now combining corpus-based with template based approaches (Galley, Fosler-Lussier, & Potamianos, 2001), where the corpus statistics can be used to post-select from a set of generated candidates. An interesting example where trainable methods are applied at the level of sentence *planning* is the work at AT&T on the “SPoT” sentence planner (Walker, Rambow, & Rogati, 2001). Corpus-based approaches are also showing up in the text generation community, as witnessed by the recent paper on the Fergus system (Bangalore & Rambow, 2000), which selects from a tree of hypotheses generated by a syntax-driven system using n -gram statistics obtained from a large corpus.

This paper describes the response planning and generation components of our MERCURY flight reservation system. Since MERCURY is by far the most complex dialogue system that we have developed, it places heavy demands on rather sophisticated response generation capabilities. The paper is organized as follows: we will first discuss the various dialogue strategies that one can adopt when developing conversational systems, since the degree of sophistication of response planning and generation is closely tied to this choice. This is followed by a brief description of the system architecture. The bulk of the paper is devoted to a detailed description of the design considerations, the control strategy, the planning of the deep structure form, and the generation of the surface form strings. We conclude with some evaluation of system performance, and suggestions for future work.

2. General dialogue strategy

In terms of dialogue strategy, spoken dialogue systems can be classified along at least two important dimensions: the degree to which the system takes control of the conversation, and the degree to which the system seeks to confirm information provided by the user. In the first dimension, systems are typically categorized into one of three main classes (Walker & Whittaker, 1989): system-initiated, where the system attempts to take complete control of the dialogue, mixed-initiative, where the system may try to guide the user through explicit prompts, but does not require the user to answer the questions, and user-initiated, where the system simply answers each user question as best it can, without providing any further prompts to elicit new information. Along the dimension of confirmation, a system may explicitly confirm every important piece of information the user provides (e.g., “Did you say to Dallas?”). It might instead confirm user requests implicitly, thus expecting the user to take the initiative to amend any recognition errors (e.g., “Okay, flights from Boston to Dallas on March third, one moment please.”). A third alternative, which is potentially more powerful but also more difficult

SYSTEM: Welcome Angie. How may I help you?

USER: I would like to <uh> [mis-recognized as: I would like to Atlanta]

SYSTEM: Okay from Hartford to Atlanta. What date will you be travelling?

USER: October thirty first.

SYSTEM: Okay, from Hartford to Atlanta on Wednesday October 31. Can you provide the approximate departure time or airline?

USER: Late morning.

SYSTEM: Okay, flights from Hartford to Atlanta departing between 10:00 a.m. and noon on Wednesday October 31. One moment please... I have a nonstop Delta flight leaving at 11:20 a.m. and arriving at 1:45 pm? Would that work?

USER: No.

SYSTEM: There are no nonstop flights. I have a U S Airways flight connecting in Charlotte leaving at 10:45 a.m., a United flight connecting in Washington Dulles leaving at 11:20 a.m., and a U S Airways flight leaving at noon. Please select one of these flights or change any constraint you have already specified.

USER: I would like to change the arrival city.

SYSTEM: Please enter the destination city using the telephone keypad ending with the pound key.

USER: 6752636# [user enters "Orlando" via keypad]

SYSTEM: I heard you say you want to arrive in Orlando. Is that correct? If you need help, you can say, for example, what cities do you know in Oregon.

USER: That is correct.

Figure 4. Example of a dialogue involving a successful entry of an arrival city using the telephone keypad.

to implement, is to confirm implicitly except when the user appears to be having some difficulties, in which case the confirmation becomes explicit.

The MERCURY system adopts the middle ground along both dimensions. The system ends most responses with a closing “continuant,” inviting the user to take the floor, but also, in many cases, encouraging the user to focus on a particular subtask that the system deems appropriate at this time. Thus, it might say, “Can you provide a departure time or airline preference?” if the flight set is as yet unconstrained except for source, destination, and date. The user could answer this request with, for example, “I want to connect in Denver,” or “any time,” or even, “no, to Austin Texas.” At critical junctures, the system takes the initiative to begin the next phase of the dialogue, through prompts such as, “Shall I help you with the return flight now?” or “Shall I e-mail your itinerary to you?” At all times, the full vocabulary and language model are available to the recognizer and understanding component, although biases are introduced in the search to favor responses that are compliant (Seneff & Polifroni, 2001).

In terms of confirmation, the MERCURY system generally assumes that it has understood the user’s query correctly, and uses implicit confirmation by summarizing its

understanding of the flight constraints in an intermediate reply that is spoken simultaneously with database access. This summary also includes any attributes that may have been inherited from the dialogue history. The user then has the opportunity to correct any misunderstandings in a subsequent query. The system keeps a detailed record of the prior activities for critical attributes such as source, destination, and date, and uses this record to detect conditions where understanding appears to be compromised, taking into account as well the confidence scores provided by the recognizer for its hypotheses (Seneff & Polifroni, 2001). In such cases, it reverts to an explicit confirmation mode, and finally, after continued misunderstanding, or having received the user's explicit notification of a problem with a particular attribute, it invites the user to enter the information using the telephone keypad. Of course, if the user has access to a graphical interface, they can type in any information that is difficult to understand as spoken input.

Figure 4 shows an example subdialogue where the destination city was successfully entered using the telephone keypad. Interestingly, the user delayed the correction until the system invited them to change any constraint that was already specified. This particular user probably believed that they were required to respond to the prompts, although it is conceivable that the user's delayed response was due to inattentiveness. This dialogue thus reveals some of the difficulties encountered due to users' false assumptions about the system behavior.

3. Overall system architecture

MERCURY makes use of the Galaxy Communicator architecture (Seneff *et al.*, 1998, 1999), consisting of a number of specialized servers that communicate with one another via a central programmable hub. The flow of control among the different servers is outlined in Figure 5. An audio server captures the user's speech via a telephony interface, and transmits the waveform to the speech recognizer (Glass, Chang, &

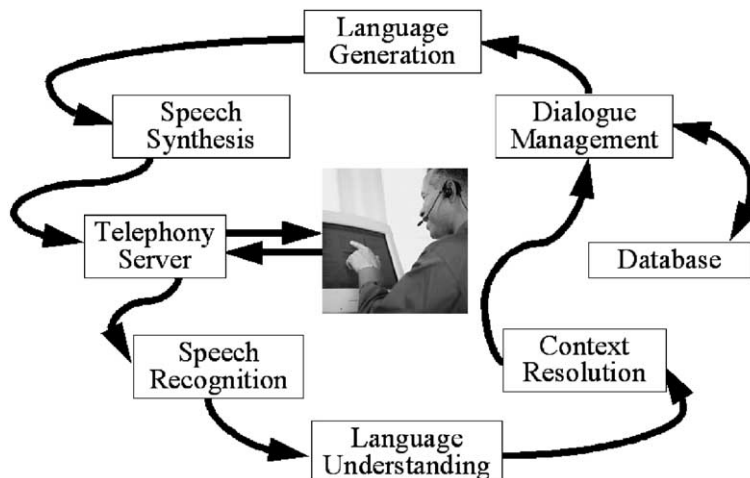


Figure 5. Block diagram of program flow in a typical turn in a MERCURY dialogue. The various servers communicate with one another via a central hub (not shown), under the control of a hub program.

| | |
|--|------------------------|
| (1) flight.leg returning & !date | → need_return_date |
| (2) !destination | → need_destination |
| (3) !source | → need_source |
| (4) !date | → need_date |
| (5) !departure_time & !arrival_time & airline all_airlines | → need_time |
| (6) !departure_time & !arrival_time & !airline | → need_time_or_airline |

Figure 6. Selected rules from the dialogue control table concerned with pre-retrieval slot filling. Note that “!” means “NOT” and “&” is a logical AND. The rules are numbered for expository purposes only.

| | |
|---|------------------------------|
| topic which_date & selected_flight | → speak_flight_date |
| num_found =1 & which_time arrival | → speak_arrival_time |
| flight details & topic aircraft | → speak_aircraft |
| num_found =0 | → speak_answer0 |
| num_nonstops =1 | → speak_nonstop_flight |
| time_selected & departure_time | → speak_best_two_departures |
| num_nonstops >1 & num_nonstops <4 & !arrival_time | → speak_three_departures |
| num_nonstops >3 & !arrival_time | → speak_first_nonstop_flight |

Figure 7. Selected subset of dialogue control rules associated with response generation, once a set of flights has been retrieved.

McCandless, 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 context resolution component (Seneff, Goddeau, Pao, & Polifroni, 1996). This component outputs a frame-in-context, which is transformed into a flattened E-form (electronic form) by the GENESIS generation server (Baptist & Seneff, 2000). This E-form is delivered to the dialogue manager, and provides the initial settings of the dialogue state.

The dialogue manager consults a *dialogue control* table to decide which operations to perform (Seneff & Polifroni, 2000a) (see Section 6, along with Figures 6 and 7), and typically engages in a module-to-module subdialogue to retrieve tables from a database provided by the Sabre flight reservation system. It prepares a response frame, which is sent to GENESIS for transformation in parallel into both a text string and an annotated string that specifies the input controls for the speech synthesizer. This synthesis string is dispatched to the ENVOICE speech synthesizer (Yi & Glass, 1998; Yi, Glass, & Hetherington, 2000), which completes the process of waveform generation by expanding into waveform segments those portions of the reply that had been left unspecified by the GENESIS system. A WAVEFORM server looks up and concatenates the now fully specified waveform segments, and sends the resulting waveform to the telephony server for transmission over the telephone network to the user. Meanwhile, if a GUI server is included in the system interaction, it receives the generated textual response for display in a Web browser window, along with the list of displayed flights, if present, represented in HTML. The entire dialogue is recorded in detail in a log file for later examination.

4. General considerations

Response generation in MERCURY, as in all of the Galaxy domains, is implemented in two stages. The first stage, also known as deep structure generation, involves deciding

what to say, and the second stage, surface-form generation, concerns deciding how to say it. Response planning is the most significant task performed by the dialogue manager, whereas surface-form generation is carried out by the language generation server.

In a mixed-initiative dialogue system, response planning is a much more challenging task than it is for a strictly system-initiated model. The main reason for the additional complexity is that the system must be able to respond appropriately to anything the user might say, as contrasted with simply dealing with the N different alternatives explicitly programmed into each designated dialogue state. It is the non-compliant responses that provide the greatest challenge, since they are so open-ended and unpredictable. Understanding error further contributes to the challenge, as the system can never be certain that the user really said what they appear to have said. Furthermore, understanding error is usually higher than in systems where the user is tightly restricted at each turn, due to the increased perplexity of the full-domain language model.

In our system design, response planning is one of the many responsibilities of the dialogue manager (Seneff & Polifroni, 2000a). It is often intimately tied to other activities, and it is difficult to tease apart the response planning aspects from other tasks. The dialogue manager is also responsible for interpreting the user query, resolving ambiguities, and deciding when explicit verification is necessary. In addition, it maintains a detailed history of the dialogue plan, including a user model, various sets of flights to which the dialogue has been exposed, elements in focus, and detailed records of previous responses, all of which may become pertinent to preparing the response for the current turn. When a query is deemed well-formed and complete, it issues a request to the database. It may find it necessary to issue multiple requests while loosening constraints. For example, if it finds no flights on the user-designated airline, MERCURY omits the airline constraint for a second database retrieval.

MERCURY's dialogue manager also prepares context-setting frames that are utilized by other servers in a subsequent turn (Seneff & Polifroni, 2001). These frames, which are intimately tied to the response, include a context-update frame, which encodes the system-side activities that need to be incorporated into history, a system initiative, which will allow the context resolution component to properly interpret a fragment in the context of an explicit prompt for additional information, and a filter list, which is used to bias the selection process in the NL server based on the recognizer's hypotheses. Further, details of this aspect of MERCURY can be found in Seneff and Polifroni (2001).

Finally, the dialogue manager has a rather sophisticated help mechanism, which provides different kinds of help under different circumstances. For example, whenever the user verbally deletes the previous turn through a meta-level "scratch that" command, the dialogue manager provides a summary of the dialogue state for grounding purposes. The dialogue manager also devotes considerable effort towards maintaining a detailed record of the prior patterns for critical attributes, namely, source, destination, and date. At every turn, it decides whether to prompt for verification of one of these attributes. In rare cases, it decides to ask the user to enter the problematic attribute using the telephone keypad. These complex monitoring exercises evolved as a consequence of our awareness that persistent misunderstanding of these attributes can completely derail a dialogue. In fact, probably the most common reason for failure to obtain an itinerary is difficulty in communicating a critical fact of the itinerary such as the destination city. Again, please refer to Seneff and Polifroni (2001) for more information.

The response that the dialogue manager prepares is represented in the format of a semantic frame (see Figures 10 and 11). The highest level structure is always a clause, whose name reflects the particular top-level message being delivered. In addition, it often also contains a number of specific variables for speaking, along with a number of special elements. These may include an embedded noun phrase that represents the main topic, and one or more comments, maintained in a list of clauses. As well, there is often a list of flights to be spoken about. Finally, most response frames contain a final prompt, intended to suggest possible next moves. The information is represented semantically, but we have attempted to keep the formatting as simple as possible to ease the task of response preparation.

Thus the dialogue manager is responsible for providing a response frame, along with, optionally, a system initiative frame, a context-update frame, and a filter list to influence hypothesis selection. Once the response frame is fully prepared, it is passed on to the generation server, which interprets it simultaneously into a well-formed textual response to be displayed in the graphical interface, if present, as well as a formatted string that is then sent to the *ENVOICE* server for speech synthesis. When the user is interacting with the system via a Web page, the dialogue manager also sends to the hub a full flight list meeting the constraints of the user query, for ultimate display in the graphical interface. It is the responsibility of the generation server to convert this list into HTML code, supporting hyperlinks and clickable icons (see Section 7.2).

5. Response planning: control strategy

Response planning is a complex process which presents significant challenges in terms of organization and control. It is also the aspect of dialogue systems that seems to be most resistant to decomposition into a domain-independent core computation associated with externally specified domain-dependent tables or rules. We have made a significant step in this direction through the implementation of a dialogue control table mechanism (Seneff & Polifroni, 2000a), where a set of ordered rules specifies the execution of a set of specialized routines, each of which typically carries out a very focused activity, as a small contribution to the overall plan for the turn. A typical dialogue turn in *MERCURY* involves up to 20 or 30 sequential rules, some of which only update internal representations, whereas others obtain information from a database or prepare responses and/or other updates of information that will be delivered back to the hub as context conditions for other servers, such as a filter list for *N*-best processing in the parser. This approach is much more powerful than a finite state implementation of dialogue, because implicit state is constructed dynamically through the sequence of rules that fire in any given turn. Among other approaches to dialogue control described in the literature, probably the *AMICA* framework advocated by AT&T (Pieraccini, Levin, & Eckert, 1997) most closely resembles our approach.

MERCURY's dialogue control table currently contains over 350 rules. As illustrated in Figures 6 and 7, each rule consists of a simple test on variables maintained in a dialogue state. At each turn, the dialogue state is initialized as a set of keys and associated values derived from the context-resolved user query. When a rule fires, it results in the execution of an operation specified in the rule. Rules often alter state variables, which can then lead to a cascade of interdependent rules choreographing a complex plan. At each dialogue turn, the system begins again at the top of the rules list, and executes in order the sequence of rules whose constraints are matched. Usually,

after an operation is completed, control returns to the next rule in the sequence. However, several distinguished operations return a “stop” condition, signalling a premature exit from the rules list. Most of MERCURY’s rules are domain dependent, although we are currently pursuing a line of research that attempts to formalize a subset of the rules into a more generic specification, for wider applicability.

MERCURY’s 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. Another dozen or so are concerned with meta-level interactions such as apologies for missing services (no flight status information available), and requests for help or repetition. Many 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 n th flight, or it could be an inquiry about the “United flight” or the “nine a.m. flight.” Another category involves interpreting various references to relative dates and/or times such as “the following Thursday,” or “that same afternoon.” Other rules are concerned with assessing the critical constraints and deciding whether to explicitly confirm them or issue a keypad request. There are additional rules concerned with retrieving information from the database and updating context information. The largest single category concerns preparing the response frame, after the database retrieval has already taken place. Over 30 rules are concerned with this task, and they are typically triggered on a combination of the number of flights retrieved and the specific request of the user (e.g., “where do they connect?”).

We have found that a key component to managing complexity is to constrain the interactions among the core operations, such that they are not permitted to directly call one another. Instead, their sequencing is controlled by the dialogue control table. This does not mean that operations are unable to influence the execution of other operations. Rather, if the execution of one operation leads to the need for a second operation to be executed, this process would be mediated via the dialogue control table. That is, the first operation would typically set a variable to a value that would then trigger the execution of the subsequent operation. This has the desired consequence of enforcing a linearization of the execution of the plan, while at the same time exposing the unfolding of the plan at the highest level, to ease maintainability.

In MERCURY, many of the operations are specific to the flight domain, although we have tried to maintain larger generality wherever feasible. The dialogue control mechanism will become clearer in the next section, where we show some examples of rules that control aspects of response generation such as prompting for missing constraints or speaking about retrieved flights.

For convenience, we will divide the response planning problem into two clear subcategories: pre-retrieval and post-retrieval. By pre-retrieval, we mean every response that is not conditioned on a database retrieval. This is typically in situations where the query contains either insufficient constraint or information that is suspect and therefore requires further confirmation. The post-retrieval phase generally involves the dialogue turns beginning the first time a set of flights between two cities is proposed, and ending at the point at which the user selects a flight to be added to the flight plan.

When the dialogue manager is ready to retrieve flight information from the database server, it sends a frame back to the hub encoding the flight constraints, along with an

intermediate response frame that will be processed in parallel with database retrieval. The intermediate reply serves both to fill dead time and to provide feedback as to the system's current understanding of the constraint space.

Once the database has returned a set of flight candidates to the dialogue manager, a response frame can then be constructed, describing the contents of the retrieved list. This is the post-retrieval phase of the dialogue plan. When control flow reaches the end of the rules list, the turn has been completed, and the response frame should be fully prepared at this time. It is then sent back to the hub, and redirected to the *GENESIS* generation server for further processing.

In the next two sections, we will describe the deep structure generation followed by the surface form generation aspects of *MERCURY*.

6. Deep structure generation

Response planning most obviously means deciding what to say about the information that has been retrieved from a database. Although this is a significant component of *MERCURY*'s response planning mechanism, there are other kinds of responses that are necessary whenever the system detects inadequacies or inconsistencies in the user's query. As well, there are conditions under which the system may decide to take the initiative, or to offer helpful hints to help guide the user towards a successful completion of the mutually evolving plan. Finally, should the user ask for help, it is a complex task to decide what to say conditioned on the dialogue state. These other aspects are much less well defined than the task of speaking about a set of flights, and it can take several iterations of refinement through careful examination of user interactions to ultimately realize an effective interaction.

6.1. Pre-retrieval

The *MERCURY* system requires a source, destination, and date before it can retrieve flights from the Sabre database. It also seeks one additional piece of information, optionally either an airline or a time restriction, anticipating that otherwise the flight set will be too large to summarize effectively. The rules controlling these requirements, a subset of which are shown in Figure 6, are very straightforward. They are not like traditional AI planning rules, but are rather a short-hand notation for "if-then" conditions. They are intentionally meant to be as simple as possible, to minimize confusion on the part of the system developer, with regard to program flow. The order in which they appear in the dialogue table controls the order in which the vacant slots will be solicited. For example, rule (1) in the table says that a return flight must have a date. Rules (2), (3), and (4) solicit the critical slots, source, destination, and date. Rules (5) and (6) are soliciting additional optional information. Rule (5) fires when the user has already stated explicitly that any airline is fine. Otherwise it will prompt for either airline or time. The user has the option of saying "any time," in which case the system will retrieve flights in spite of the large search space. These rules are intended to control the top-level execution plan of the turn rather than, for example, to enforce pragmatic constraints such as the fact that the arrival time must be later than the departure time.

A critical aspect of this phase is the successful communication of the source, destination, and date, all of which are susceptible to recognition error. Therefore, the system

maintains detailed monitoring of the evolution over time of these attributes. For source and destination, it tabulates at each turn whether the attribute was inherited, repeated, or changed. If a change is detected after flights have already been retrieved, the system prompts for confirmation of the surprise move, anticipating possible recognition error. After persistent confusion, or if the user asks explicitly to change the source or destination, the system requests the user to enter the city by spelling it using the telephone keypad. It turns out that MERCURY's 500 cities are uniquely identifiable through their keypad codes, although if this were not the case a follow-up disambiguation dialogue could be arranged. This keypad mechanism provides as well the opportunity to confirm whether the desired city is known or unknown.

A similar process takes place for dates. If the user appears to repeat the date, without providing any other information, there is the suspicion that a misrecognized date has again been misrecognized the same way.² In this case, the system tries to find an alternative hypothesis for the date, by re-examining the N-best list of recognizer hypotheses, and, in any case, also asks for user confirmation. As is the case for cities, the system invokes the keypad upon repeated date corrections.

6.2. Post-retrieval

Once a query is fully specified, the system can retrieve a set of flights that match the constraints. The generation task can now be formulated as the process of deciding how to represent these flights verbally so as to allow the user to begin the selection process. This aspect is controlled by an ordered set of dialogue table rules that are typically conditioned on the number of flights retrieved, as well as other significant variables such as the number of nonstops and whether or not the user has specified arrival time restrictions.

As mentioned earlier, there are currently over 30 rules concerned with preparing a response frame, once the flight set has been determined. A selected subset of these rules is shown in Figure 7. If the user has asked for a specific attribute, such as the date, the aircraft, or the connection airport, it calls a specialized routine that provides the relevant information. If there are any nonstop flights, these are singled out for response generation. If there are more than three nonstop flights, the system will not enumerate them, but instead just lists the available airlines and mentions explicitly only the earliest flight. This is done to define one edge of the time window and set up a reference frame for subsequent interaction. The key "time_selected," shown in the figure, is an indicator that the user has asked for flights around a certain specific time. In such a case, the system offers the best matching flight, as well as an optional second alternative, if a reasonable second choice exists.

If there are no more than three flights matching the constraints, the system lists these flights in temporal order, focusing on either the departure time or the arrival time, conditioned on the particulars of the user query. The system identifies the time and airline for each flight, as well as the connection airport(s), if present. If the source or destination city is associated with multiple airports, it also identifies the airport for each flight.

²It turns out that our recognizer tends to favor certain dates over others, and this can have the effect of a consistent error in a second attempt.

| |
|---|
| Shall I add this flight to your itinerary? Would one of these work? Shall I help you with the return flight now? Shall I price your itinerary? Shall I e-mail your itinerary to you? Is there something else I can do for you? |
|---|

Figure 8. Examples of some possible continuants that occur at selected points in the dialogue plan.

To reduce verbosity, the system preprocesses the flight list so as to remove common elements, aggregating them into a summary statement. Thus, if all of the flights are United flights and connect in Denver, the system will say: “I have three United flights connecting in Denver: a flight leaving at . . .”

6.3. *The continuant*

Unless the response inherently requests new information, the system generally ends its response with a remark intended to focus user attention, which is analogous to a prompt in a scripted dialogue, but is distinguished by the fact that it is not intended to imply that the user must answer the prompt to proceed. We refer to this component of the response as the “continuant,” examples of which are shown in Figure 8. If there is a single flight that meets the specifications, the system asks, “Shall I add this flight to your itinerary?” If a list of flights is presented, it asks a less focused question, such as “Would one of these work?” Other continuants involve a direct attempt to move on to the most logical next phase of the overall plan, such as, “What date will you be returning on?” “Shall I price your itinerary?” “Shall I e-mail your itinerary to you?” These pro-active requests can be very effective for moving the plan forward, although the situation becomes far more complex when the user chooses not to be compliant, due to diverging goals.

The choice of continuant moves has evolved over time, as a consequence of accumulated experience with naive users. Much to our surprise, we found that a significant number of naive users responded to the question, “Would one of these work?” with the simple direct answer “Yes.” Our intent was to provide a polite form of a request for selection, implying that selection was not mandatory. We believe that a simple “yes” answer would be highly abnormal behavior if a human agent had asked the question. Apparently, users have an internal model of computer dialogue systems that is divergent from their expectation for a human. This is, of course, not a new observation—see, for example, Jonsson and Dahlback (1988) for a discussion of the special nature of human–computer interaction. This phenomenon certainly presents an interesting additional burden to a system that is attempting to emulate human behavior.

To ameliorate this problem, we introduced state dependency into the continuant for the specific situation of a listing of multiple flights. The first time multiple flights are introduced, the system says: “Please select one of these flights or change any constraint you have already specified,” as a kind of help mechanism.³ The second time, it says,

³The reader is referred back to Figure 4 to see an example where this “verbose_select_one” prompt was critically important.

simply, “Please select one.” Subsequently, one of several different polite form requests is used, such as “Are any of these okay?”

Another problematic stage with regard to continuants is the point at which the user has just added the first flight to the itinerary. The problem is that, in the usual case, we have not yet resolved whether this is a one-way, round-trip, or multi-leg itinerary. The system could at this point prompt for the three-way choice, but this would require an additional dialogue turn. Since the vast majority of trips are not one-way, we felt it was more productive to assume the user would be continuing on, but to allow them to conclude a one-way trip by ignoring the prompt. Then the only question is whether to assume a round-trip (the dominant condition) or to ask the more general question, “Where would you like to go next?” Again, this would consume a turn if it is a round-trip.

During its first year of operation, MERCURY’s next-leg response was “Where would you like to go next?” However, over the past year, we have been fielding a system which simply assumes a round-trip and asks, “What date will you be returning on?” We feel that this system is more effective than the original one: users seem to know to say, “I want to go on to San Francisco,” etc., when they are trying to book a multi-leg trip. And they can solve the task more efficiently in the most common round-trip condition. With either of these prompts, there remains the problem of detection that a user intends only a one-way trip. Initially, we had some difficulty with detecting this condition, but once we were able to collect several examples of users specifying, in a variety of different ways, that their trip concluded after just the one leg, we were able to handle this situation fairly well.

6.4. *The comment list*

In addition to the main response and the continuant, the MERCURY dialogue manager may also decide to introduce one or more comments into a comment list, which typically precedes the main response. Some example comments are shown in Figure 9. For example, if the user asks for Delta flights and the system is unable to find any Delta flights meeting the other specifications, it adds the comment, “I couldn’t find any flights on Delta,” drops the airline constraint, and reissues the database request.⁴ The first time that it determines that all of the retrieved flights have at least one connection, it adds the comment, “There are no nonstop flights.” It remembers that it has already said this in follow-up queries, to avoid extra verbosity.⁵

If the system encounters a constraint that is inconsistent with the context or incompletely specified, it warns the user in a comment that it is ignoring the constraint. A typical situation where this might occur is when the user has referred to a flight by its time, but the time reference was recognized incorrectly, such that no flights meet that requirement. It is not clear yet whether it is productive or counter-productive to inform the user of the ignored constraint—sometimes this leads to additional confusion, such that it might be better to just discard the misinformation. We are continuing to monitor user dialogues to help us decide the best way to deal with these difficult situations.

⁴By returning control back to the top of the dialogue table rules.

⁵The system has access to its previous reply frame in a history record, which is preserved for each dialogue turn.

I couldn't find any flights on Delta.
There are no nonstop flights.
I heard you say nine a.m. which I'm ignoring.
I have flights on American and United.
If you need help you can say, for example, 'What cities do you know in Oregon.'

Figure 9. Examples of some possible comments that might appear before the main clause of a response.

6.5. *The intermediate response*

Whenever the system needs to go to the database, it provides an intermediate response that is spoken while database access is taking place, to fill in the delay time as well as to inform the user of its understanding of their query. This response is basically a paraphrase of the flight constraints as the system has interpreted them at this point. It will include explicit specification of its interpretation of vague terms such as “evening flights” or “the following Tuesday.” It also serves to inform the user of any constraints that may have been remembered from context, and of the source and destination as understood by the system, which could, of course, be erroneous. An example intermediate reply might be, “Okay, American flights from Dallas to San Diego leaving between five p.m. and midnight on Tuesday March 4th. One moment please.”

There are likely to be extra long delays when the system is pricing an itinerary. Therefore, at this opportune time, the system takes the initiative to summarize the entire flight plan via an intermediate reply.

6.6. *Response frame representation*

In the discussion above, we have presented the responses as if they are fully formed English strings, in the interest of clarity to the reader. Of course, in actuality the responses are encoded in semantic frames, which are later translated into spoken responses by the generation server.

Two examples of response frames are given in Figures 10 and 11, along with their corresponding generation strings. In the next section, we will explain how the frames are converted into well formed English strings, as well as into speech waveforms for speaking. The first response is for the case where there are too many flights to speak about separately. Instead, the system summarizes by listing airline options in a comment, and mentions explicitly only the first nonstop flight. The continuant is intentionally empty. The second example is a case where there are three connecting flights meeting the constraints. They share a common airline and a common connection airport. The continuant, “verbose_select_one,” is the prompt used the first time a flight list is introduced. There is also a comment regarding the absence of nonstop flights.

7. **Surface-form generation**

Surface form generation in *MERCURY* is realized in a separate step that is carried out by the *GENESIS* generation server. Generation and synthesis are tightly coupled by encoding pointers into waveform segments directly into the generation lexicon. Thus the synthesizer can make use of the pre-existing structure of the response frame to directly control prosodic aspects of waveform selection. This approach has led to extremely

Response Frame

```
{c speak_first_departure
:departure_time "7:00"
:arrival_time "10:25"
:depart_xm "a.m."
:arrive_xm "a.m."
:airline "UA"
:continuant {c empty }
:comment_list ( {c list_airlines
:airlines ( "AA" "TW" "UA" ) } ) }
```

Response String

"I have flights on American, TWA, and United. The earliest nonstop United flight leaves at 7:00 a.m. and arrives at 10:25 a.m."

Figure 10. An example response frame for the case where there are too many nonstop flights to speak about individually, along with the corresponding response string.

Response Frame

```
{c speak_departure
:comment_list ( {c no_nonstops } )
:num_flights 3
:common_connection "ATL"
:common_airline "DL"
:flight_list ( {c departing_flight
:departure_time "7:00"
:depart_xm "a.m." }
{c departing_flight
:departure_time "12:50"
:depart_xm "p.m." }
{c departing_flight
:departure_time "5:40"
:depart_xm "p.m." } )
:continuant {c verbose_select_one } }
```

Response String

"There are no nonstop flights. I have 3 Delta flights connecting in Atlanta: a flight leaving at 7:00 a.m., a flight leaving at 12:50 p.m., and a flight leaving at 5:40 p.m. Please select one of these flights or change any constraint you have already specified."

Figure 11. An example response frame for the case where there are three connecting flights with a common connection airport and airline, along with the associated response string.

high quality synthesis, although there are strong domain-dependencies inherent in the design, since the waveforms are carefully selected from carrier phrases that reflect similar prosodic contexts.

If a graphical interface is present, a text string is presented to the user, which, at present, is very similar to the spoken utterance. The generation of a well-formed text string for display in a graphical window and the generation of the speech waveform for speaking currently share a common rule set that controls the sequencing of words and phrases to produce well-formed English sentences. A separate set of generation rules is used to produce hyper-linked and clickable tables in HTML for presentation of the full list of flight options.

In contrast to deep-structure generation, surface-form generation can be formulated as a task that is accomplished using generic library code that is completely independent of the domain or language, with externally specified rules and lexica specifying the needs of each particular domain and language. Several years ago, we developed the *GENESIS* system as a specialized tool for language generation (Glass *et al.*, 1994). More recently, based on our extensive experience with *GENESIS* across many languages and domains, we decided to undertake a significant redesign initiative. The resulting system, *GENESIS-II*, is a powerful tool that is both more capable and more easily mastered than its predecessor. *MERCURY*'s response generation task is relatively straightforward and does not utilize many of the more sophisticated features of *GENESIS-II*. The interested reader is referred to Baptist and Seneff (2000) for a detailed description of *GENESIS-II*, to Seneff and Polifroni (2000b) for information about how *GENESIS-II* handles constituent movement in wh-queries, and to Wang *et al.* (2000) and Zue *et al.* (2000) to understand how it deals with frame reorganization required for fluency in translation tasks.

Both *GENESIS* and *GENESIS-II* are based on the principle of generating surface form patterns by applying a set of generation rules, invoked recursively via the contents of the semantic frame, beginning with the top-level clause and expanding downward until lexical items are encountered. Each frame has both a name and a type, and its generation rule is indexed under its name. For a more compact rule specification, system developers can group constituents that have the same generation rule into a set that share the common rule. If there is no rule present under its name or its group, a default generation rule can be applied based on its type (one of: clause, topic, predicate). Generation begins with the top-level clause constituent, which looks up its generation rule, and constructs a string by concatenating subconstituents in the order indicated by the grammar rule. A lexicon is consulted to construct the final surface form, which includes both information about inflectional endings as well as any semantic-context or prosodic-context dependent alternatives.

| | |
|--------------------------|--|
| direct quote | A directly quoted string, which is placed "as is" |
| vocabulary lookup | A symbol-string preceded by "!", signalling that it should be expanded according to its <i>vocabulary entry</i> before placement |
| recursive rule expansion | A symbol-string preceded by ">", which signals that another <i>rule</i> should be looked up under the name given and fully expanded |
| keyword | A symbol string preceded by ":", meaning that the entry in the frame under the given <i>key</i> should be fully expanded into a string |
| predicate | A symbol string without a preceding marker, indicating that any <i>predicate</i> in the frame with this name should be generated |

Figure 12. A list of the dominant generation expansion variants on the right-hand side of generation rules.

| Selected Rules | |
|-----------------------|--|
| speak_first_departure | :comment_list !earliest_nonstop :airline !flight >leaves_at >and_arrives_at . :continuant |
| leaves_at | !leaves_at (:departure_time :dpt.time) :depart_xm |
| and_arrives_at | !and_arrives_at (:arrival_time :arr.time) (\$Low :arrive_xm) |
| :comment_list | :nth |
| list_airlines | !have_airlines :airlines . |
| :airlines | \$List >singleton >but_last >last |
| singleton | :singleton |
| but_last | :butlast , |
| last | \$Low !and :last |

| Selected Vocabulary Entries | |
|-----------------------------|--|
| AA | O “[mercury061 32821 40650 American]” |
| AA | LIST “[mercury263 116294 124760 American]” |
| a.m. | O “[mercury068 63384 71092 a.m.]” |
| a.m. | Low “[mercury090 79463 87864 a.m.]” |
| and_arrives_at | 0 “and arrives at” |

Figure 13. Selected rules and vocabulary entries in GENESIS-II controlling generation of the response string for Figure 10.

The easiest way to explain how GENESIS-II works is by way of a couple of detailed examples. The reader is referred to Figure 12 to help elucidate the process. We begin with generation from the frame given in Figure 10. As illustrated in Figure 13, generation begins by consulting the rule indexed under the name of the top-level frame, in this case, “`speak_first_departure`.” The rule says that the generation result for the “`:comment_list`” should appear first, followed by lexical expansion (!) of the lexical entry “`earliest_nonstop`,”⁶ the entry under the “`:airline`” key, if present, the word “`flight`,” and then, finally, the two predicates, “`leaves_at`” and “`and_arrives_at`.” The “`>`” marker indicates that a rule is to be expanded at this point, indexed under the subsequent symbol string. The further expansion of “`leaves.at`” and “`and_arrives_at`” follows their corresponding rules, as shown. The first set of parentheses in the “`and_arrives_at`” rule enclose alternates, thus allowing some flexibility in the key chosen to represent arrival time. The second set of parentheses illustrate the use of a prosodic marker, “`$Low`,” attached to the lexical realization of the “`:arrive_xm`” key. The lexical items, “`a_m`” and “`p_m`” have separate entries under “`$Low`,” chosen to reflect the low phrase-final prosodic contour.

The “`:comment_list`,” if present, is always a list of one or more clauses. Its generation rule is simply “`:nth`,” a special key used for lists which simply instructs the generator to walk the list, generating each clause in turn. In our particular example there is only one clause, “`list_airlines`,” whose generation rule states that the generation string for the “`list_airlines`” vocabulary item (referenced through the special symbol `Score`) should be followed by the generation string for the “`:airlines`” key.

The “`:airlines`” rule handles lists of airlines, which may contain only a single airline (>singleton). The “`$List`” flag is used to select a different prosodic contour for airline

⁶In actuality, this could be represented directly as an in-line string in the rule, by omitting the exclamation point and the underbar, and inserting quotes around the words, except for the fact that the words must be mapped into a speech waveform in the synthesis lexicon.

names acting as nouns in this list format, as opposed to the selection for adjectival airlines modifying a flight. A separate rule is devoted to the last item in the list, which handles insertion of the “and.” In addition, the \$Low flag overrules \$List, and allows for an appropriate phrase-terminal prosodic contour.

Selected vocabulary items for entries relevant to this example are also listed in the figure. Pointers into waveforms are provided using the special structure enclosed in brackets, identifying the waveform file name along with the particular segment’s end points, and an optional text string providing the linguistic content for expository purposes. Vocabulary entries can also be entered simply as English words, as illustrated for “and_arrives_at.” In such cases the ENVOICE speech synthesizer will determine automatically which waveforms to concatenate using a finite state transducer based search mechanism (Yi *et al.*, 2000). In general, direct specification of the waveform files in the lexicon results in superior speech synthesis quality, as well as reduced computational load.

A slightly more complex example is given in Figure 11, for the “speak_departure” clause. Selected rules are given in Figure 14. An interesting point to make here is that the various “common” entries may or may not be present in any particular case. These are the attributes that were determined to be identical in all the flights in the list. A generation policy in *GENESIS-II* is that, if none of the items specified in the right-hand side of a rule are present in the frame, the rule is omitted. Thus, in our example there is no common source, so the string “departing_from” would be omitted, as is appropriate. This single feature allows for a much more parsimonious set of rules than would otherwise be required.

As mentioned previously, the ability to map directly from the meaning representation to the marked-up synthesis string allows for refined selection of appropriate prosodic contours without any additional analysis of syntactic structure. A few words in *MERCURY* are ambiguous as to their semantic interpretation as well as their prosodic context. A good example is the word “AUG,” which could be an airport code representing “Augusta, Maine” or a reference to the month “August.” Augusta is further complicated by the fact that the system knows “Augusta, Georgia” as well, and therefore it needs to say “Maine” when it speaks about this city, to disambiguate it from the other Augusta. On the other hand, if it is listing the cities it knows in Maine, it should say the short form, “Augusta,” since the state context is already known. A listing of the four distinct realizations for “AUG” in the vocabulary file is given in Figure 15. By default, “AUG” maps to “August,” which could appear with either a low or a high prosodic marker. Other rules set the selector \$Cty (when mentioning a source or destination) or \$State (when listing the cities *MERCURY* knows in a particular state), as appropriate.

| |
|---|
| <pre> speak_departure :comment_list li_have (>num_nonstops >num_flights) >leaving_at , >arriving_at . >common_source >common_connection >common_destination : :flight_list . :continuant common_source !departing_from :common_source , common_connection !connecting_in :common_connection , num_flights :num_flights :common_airline !flights num_nonstops :num_nonstops !nonstop :common_airline !flights departing_flight :airline !flight >from_airport >to_airport >connection_place >leaving_at </pre> |
|---|

Figure 14. Selected rules in the generation grammar controlling generation of the response string for Figure 11.

| | | |
|-----|-------|--|
| AUG | Low | "[mercury075 92772 100360 1 1 1 August]" |
| AUG | High | "[mercury108 13279 19572 1 1 1 August]" |
| AUG | Cty | "[mercury076 57332 71062 1 1 1 Augusta Maine]" |
| AUG | State | "[mercury108 104425 111562 1 1 1 Augusta]" |

Figure 15. Four distinct entries in the vocabulary file for the symbol string "AUG." Selection is controlled by the generation rules. See text for details.

7.1. Spoken and textual response differences

As should be apparent by now, MERCURY performs speech synthesis by a direct mapping from the response frame into a marked-up synthesis string. There are strong advantages to this approach over the non-integrated alternative of generating a single response string which could then be displayed in the graphical window and sent on to a synthesizer for re-analysis. It seems counter-productive and error-prone to discard the structural information that is available through the semantic frame, only to have to reconstruct its equivalent through additional syntactic and semantic analysis, performed perhaps in the synthesis server. As has been illustrated above, the system has tight control over which prosodic form and/or which semantic interpretation to invoke for every particular situation. The only possible disadvantage is that the system needs to generate two distinct, but strongly related, response strings from the same frame. Fortunately, nearly all of the differences can be encoded in the separate vocabulary files, while maintaining a common rule set to be shared by both generation tasks (spoken and textual).

There are very few differences between the generation rules for the English text and those required by ENVOICE for synthesis. To encode these differences, each vocabulary file has an entry under "language" which sets a flag as either ":english" or ":envoice." Those few rules which require a distinction can then just begin with "!language," and flag on the presence or absence of ":envoice." An example of a generation string that is significantly divergent is the string that states the price of an itinerary. For the text version, it would just need to produce a dollar amount, such as "\$426.33," whereas the synthesis generation needs to spell out, "four hundred and twenty six dollars and thirty three cents." Rules and vocabulary entries concerned with this task are shown in Figure 16. An example frame along with generation strings for both English text and synthesis are shown in Figure 17.

The MERCURY dialogue manager contains several routines that expand certain keys into a more detailed accounting necessary for ENVOICE. One such routine decomposes a price into separate numeric values associated with each digit's place, as shown in the figure. The generation rule tags for prosodics, which allows it to select the appropriate entry in the ENVOICE lexicon. The label "\$High" is for the *generic* high prosodic encoding, whereas the "\$Int" code is used to select for the intermediate level of stress in the lower-order digits of a fare.

Aside from numeric expressions, the other major distinction between synthesis and text generation is in punctuation and capitalization. GENESIS supplies special capitalization commands that support either capitalization on only the first word of the generation string (useful for capitalizing the entire sentence), capitalization on every word in the string (used for proper nouns such as a city name), and capitalization of every character in the string (for abbreviations such as "TWA"). Such capitalization in

| | |
|---------------------|--|
| !speak_flight_price | !language !the :fare_class !flight_price_is (\$if :envoice >envoice_price >english_price) :continuant |
| english_price | “\$” :price . |
| envoice_price | (\$High >thousands) (\$High >hundreds) (\$Int >tens) (\$Int >cents) |
| thousands | :thousands !thousand |
| hundreds | :hundreds !hundred |
| tens | !and :tens (\$if :cents !dollars_and !dollars) |
| cents | :cents !cents |

Figure 16. Selected rules used for generation of a price string in both English text and the mark-up language used by our ENVOICE speech synthesizer.

Response Frame

```
{c price_itinerary
  :price "1588.60"
  :fare_class "cheapest"
  :thousands 1
  :hundreds 5
  :tens 88
  :cents 60
  :continuant {c new_itinerary_request } }
```

Textual Response String

The cheapest price for your itinerary is \$1588.60. Is there something else I can do for you?

Spoken Response String

```
the cheapest
[ mercury130 4960 30322 price for your itinerary is ]
[ mercury131 13399 17254 one ]
[ mercury131 17254 25113 thousand ]
[ mercury135 12556 18006 five ]
[ mercury131 28365 33062 hundred ]
[ mercury131 33062 34869 and ]
[ mercury136 38030 42607 eighty ]
[ mercury135 43782 47124 eight ]
[ mercury131 42698 53507 dollars and ]
[ mercury141 38964 45408 sixty ]
[ mercury142 45859 55194 cents ]
[ mercury155 0 32800 Is there something else I can do for you? ]
```

Figure 17. Response frame for an itinerary pricing, containing a dual encoding of the price, and corresponding response strings, both textual and spoken.

MERCURY’s rules is conditioned on the “:english” flag, since it is not required for the spoken form.

Of course, it would have been possible to maintain entirely separate rules for the textual generation string and the spoken generation string, but this would have meant

that every rule would need to be doubly maintained. It is fortunate that GENESIS-II supports mechanisms that allow us to specify branching rules only when necessary, leading to much more consistent maintainability of the two response types.

7.2. Graphical interface

If the user is interacting with the MERCURY system at a Web page, the textual version of the verbal response is displayed, along with the full set of retrieved flights, allowing the option of a multi-modal interaction. The user can click on any one of these flights and refer to it anaphorically, such as, “Please book it.” or “Can you tell me more about this flight?” The clickable option is implemented through GENESIS-II rules in a separate HTML rules file, invoking a Java script to trigger the GUI server to send a frame to the hub containing the selected flight index, which is then incorporated into the discourse context. Thus, when a user has graphical support, they can interact by typing into the GUI window or by speaking into the telephone handset, where both typed and spoken input can be integrated with mouse-click selections.

The database returns the database entries as a list of frames, which are then sent to GENESIS-II to be converted into an HTML format for display. In addition to formatting the flights for tabular display and creating the clickable option, the rules also provide links into Web pages where more information can be obtained concerning selected airports and airlines. For example, the HTML grammar rule for airline might appear as follows:

```
airline “<TD align = center>” :airline “</TD>”
```

and the corresponding lexical entry might be:

```
AA “<a href = http://www.aa.com><img src = AA.gif></img></a>”
```

8. Evaluation

It is extremely difficult to devise a mechanism to formally evaluate response generation as a separate component. Response generation depends critically on several other system components, and whenever failure occurs at an earlier stage, the resulting response may appear to be faulty, even though generation may have been executed appropriately, given the system’s erroneous understanding of the user’s query. When the fault lies with the dialogue manager, it is unclear whether the error was incurred during query analysis or during response generation, since the border between these two aspects is fuzzy. Furthermore, whether the generation string is too verbose or too terse is often a matter of personal taste, and is thus a very subjective measure. It was rarely the case that the generation string was ill-formed, either semantically or syntactically, and so this measure appears pointless. In a pilot experiment, we tried to assess subjectively the quality of the generation output, but we found this to be a very difficult task, and the process seemed counterproductive. Computation has never been an issue; generation for even complex responses takes a small fraction of a second of elapsed time.

MERCURY has been operational since April of 1999, and we have attempted to keep it up and available for calls at all times. To date, over 25,000 utterances have been collected from nearly 1700 dialogues with users. These dialogues represent a diverse

user population, including expert users, naive users, and people who were just playing with the system. Between April and October of 2001, a data collection initiative was launched by National Institute of Standards and Technology (NIST), which invited frequent travelers, distributed across the United States, to make their flight plans using one of eight flight reservations systems being developed by different sites participating in the DARPA Communicator program. Each user was instructed to always call their assigned system, so that our “NIST users” have been exposed only to the MERCURY system.

After each call, the NIST users were asked to fill out a Likert scale questionnaire, in which they were asked to rate the following five questions along a scale ranging from “strongly agree” to “strongly disagree.”

1. “It was easy to get the information that I wanted.”
2. “I found the system easy to understand.”
3. “I knew what I could say or do at each point of the dialogue.”
4. “The system worked the way I expected it to.”
5. “Based on my experience using this system to get travel information, I would like to use this system regularly.”

The results of these experiments were tabulated by NIST for the eight participating sites. Out of the eight sites, the MERCURY system obtained a significantly higher ranking than any of the other sites on all of the questions except the third one (“I knew what I could say”), for which MERCURY was essentially tied with four other sites for top ranking position. Since we intentionally designed the system for flexibility in what the user could say at any point in the dialogue, it would be expected that the third question would be less likely to obtain a strong positive response than some of the other questions. MERCURY was the only system which obtained a substantially better than “Neutral” ranking on the critical final question, “I would like to use this system regularly.” We consider these user surveys, overall, to be a significant indicator that the MERCURY system is performing in a way that users find to be intuitive and productive.

In addition to the above user rankings, we have also devised several automatically computable metrics that we feel are of value in assessing overall dialogue success. We selected for evaluation data that had been collected over the period from April 1 to June 15, 2001, which are the most recent data available to us at the time of writing of this paper. The data have been divided logically into two sets, one of which (the “NIST” set) is the subset obtained from subjects who participated in the multi-site evaluation experiment mentioned above, and the other (the “MIT” set) consisting of users who participated in MIT’s internal data collection experiments. Both sets of users called the same system, but the NIST users were guaranteed to be solving either a specified scenario or a real trip they were intending to take, whereas the MIT user tasks are not controlled.

We are particularly interested in establishing evaluation metrics that require as little as possible of human effort to obtain. This is motivated by the fact that, for a fully deployed system, one would expect at least thousands of calls per day, a number far too great to support any detailed manual examination of interactions. To this end, we have developed a facility for reprocessing log files, in order to obtain some significant measures of success, such as degree of completion of an itinerary and total time to completion. We have also implemented a sophisticated program for reprocessing log files after the original utterances have been transcribed, which allows us to obtain two

| | | | | |
|-------------------|-----------------|---------------|----------------|--------------|
| :clause | :topic | :date | :source | :destination |
| :connection_place | :flight_number | :flight_mode | :nth | :airline |
| :and_airline | :departure_time | :arrival_time | :where_connect | :which_time |
| :superlative | :select | :truth_value | :close_off | |

Figure 18. The set of concepts that were monitored to measure query density and concept efficiency. See text for details.

measures that we feel are critically important. These measures, called concept efficiency and query density, are more fully described in Glass, Polifroni, Seneff, and Zue (2000). Concept efficiency is a measure of the number of times, on average, that a concept had to be repeated before it was understood by the system. Its maximum value is 1.0, which means that every concept was understood the first time it was spoken. A concept which was never understood obtains a concept efficiency of zero. Query density is a measure of the average number of concepts that were communicated to the system at each turn. A high value indicates an efficient dialogue in which the user was able to communicate multiple constraints to the system in a single turn. The set of concepts that were monitored for these measures is given in Figure 18.

Some values for these and other easily obtained measures for our two data sets are given in Figure 19. To obtain these measures, a human annotator must first transcribe the recorded utterances of the dialogue orthographically. The effort involved is small, as they begin with the selected recognizer hypothesis and modify it to correct recognition errors. The value of 0.92 obtained for concept efficiency indicates that, on average, users have to repeat a concept before it is understood only about 8% of the time. The

| Measure | MIT data | NIST data |
|----------------------|----------|-----------|
| total dialogues | 71 | 55 |
| total utterances | 1124 | 826 |
| mean words/utt | 4.5 | 4.5 |
| mean utts/dialogue | 16 | 15 |
| task completion time | 299 sec. | 353 sec. |
| query density | 1.93 | 1.98 |
| concept efficiency | .92 | .91 |
| word error rate | 15.3% | 14.4% |
| concept error rate | 11.0% | 10.6% |
| transcription parsed | 89.7% | 94.1% |

| category | num | % |
|--|-----|----|
| Itinerary reached closure and system invited pricing | 26 | 55 |
| User quit early inexplicably – just playing? | 12 | 26 |
| Recognition failure due to noisy conditions and/or strong accent | 7 | 15 |
| User misled by dialogue problems | 1 | 2 |
| User misunderstood domain to be flight status | 1 | 2 |

Figure 19. (Top) Some performance measures for the MERCURY dialogue interactions. (Bottom) Analysis of dialogues collected during February, 2002. See text for discussion.

values for concept efficiency are close to 2.0, a significantly higher measure than we had been obtaining in assessments on earlier data sets (Glass *et al.*, 2000). We find this to be particularly encouraging, as it suggests that our system has improved in efficiency over time. We suspect that a main contributor to this improvement is the change in the response subsequent to booking the first leg of the itinerary. When the user is booking a round trip, which is the most common condition, the saving of a dialogue turn to specify the return leg leads to increased efficiency.

Other more familiar measures included in the figure are word error rate and concept error rate.⁷ The latter is only computed on the subset of the utterances whose orthography parses. When the orthography fails to parse (overall, about 8% of the user utterances), we are unable to provide a reference against which to assess the system's hypothesis, unless we were willing to permit additional human effort to reword these orthographies in such a way that the overall meaning is preserved but the natural language component is able to analyze them correctly.

Of course, an extremely significant component of evaluation is to assess how often users are able to accomplish what they set out to do. In practice, this is not always straightforward, as users will often call the system only to explore its capabilities, so that the dialogue is not sufficiently goal-oriented to allow a clear judgement of success or failure. Given this caveat, we decided to manually judge task completion on a set of 47 dialogues collected during February, 2002, as shown in the bottom table in Figure 19. Over half of the dialogues were completely successful, judged by the fact that the system asked to price the itinerary. Another quarter fell in the nebulous category that, while there were no apparent difficulties, the user simply hung up prematurely, suggesting that they were just exploring the system capabilities. Another 7% were clearly cases where recognition difficulties were disturbing the dialogue to a degree that the user lost patience and hung up prematurely. All of these dialogues involved noisy conditions or a heavy accent, sometimes in combination. One user clearly misunderstood the capabilities of the system, believing they could ask about flight status. Finally, one dialogue failed due to inadequacies in the dialogue module's interpretation of the meaning of "latest flight." This is worth explaining further, as it points out the difficulties of judging where to attribute fault when dialogue interaction breaks down. The user was booking a round trip between Boston and Bombay, India, and had nearly completed the interaction, selecting "the latest flight" for the return leg when four alternatives were presented. However, in processing "latest," the system incorrectly filtered the flights to leave before midnight, whereas all of the available flights left between 1 and 5 a.m. The system misled the user into believing no flights were available, and the user therefore gave up. The fault lies in part with language generation, which made the false (due to the logic error) presupposition assumption that, if there is no latest flight there is no flight. This problem has since been repaired.

9. Summary and future work

In this paper, we have described two critical aspects of the *MERCURY* flight reservation system: response planning, carried out in the dialogue manager, and response genera-

⁷These are defined as the sum of substitution, deletion, and insertion errors over the total number of words/concepts.

tion, performed by the generation server. Response planning is executed through a set of operations, which are invoked whenever conditions are met in an associated rule in the dialogue control table. The planning process results in the creation of a response frame, encoding the meaning of the intended reply. Response generation makes use of the GENESIS-II library, which is both domain and language independent. Dependencies are encoded in a recursive set of rules, and the final details of the generation string are provided in an associated lexicon. The language generation process expands the components of the frame into their respective generation patterns, by recursively consulting the rules associated with the frame contents.

A unique aspect of MERCURY is that the response frame is converted, via a shared set of generation rules, into two distinct response strings, a well-formed English text string containing the response message, to be presented in the graphical interface, and a marked-up synthesis string, to be further processed by the ENVOICE speech synthesizer. Each of these strings is only generated if the user has access to the associated interface modality. The synthesis string contains explicit pointers into waveform files, resulting in extremely high quality synthesis. A strong benefit of this technique is that the linguistic knowledge, both prosodic and semantic, encoded in the semantic frame can be used directly to control selection of the appropriate waveform segments.

To evaluate the capabilities of our generation system, we have reported on the results of both user surveys and automatic quantitative measures of dialogue success. An independent data collection initiative conducted by NIST gave our system the highest ranking among eight competitors, on the basis of user exit surveys. We have been monitoring over time two related measures of success, defined in (Glass *et al.*, 2000), which we call “query density” and “concept efficiency.” We have observed a trend towards increasing values of query density over the most recent year of monitoring, which is likely due in part to improvements in the dialogue management strategy. The current value is close to 2.0, meaning that, on average, two concepts are introduced with each query.

In the future, we would like to further improve the quality of users’ experience in interacting with the system. We believe it would be productive to carefully study interactions between users and a human agent, to better understand how human agents present the data they are retrieving. We have found that simple techniques such as varying the phrasing of the continuant (“Would one of these work?” “Do you like any of these?”) leads to a perceived greater naturalness and less monotonous experience. We can extend this idea by providing alternate phrasings of the more substantive contributions to the response, or possibly simply variants in the intonation pattern of otherwise identical carrier phrases and/or content words. We could possibly adopt approaches similar to those advocated in (Portele, 2000), where a “chooser” selects from a set of candidate generation variants.

We are beginning to address the very challenging issue of the domain-dependent implementation of the dialogue manager. At this time, this is the only component of our dialogue systems that has domain dependencies embedded in source code. We believe that some aspects of prompt generation can be implemented in a more generic way than what has been presented here, and we also expect to be able to deflect certain aspects of dialogue management, such as date and time decoding, into separate servers specializing in these generic world-knowledge needs. Such activities should reduce the cost of developing dialogue managers for future domains.

Many members of the SLS group have contributed to the development of the MERCURY system. They include Jim Glass, who has been involved with recognition and synthesis, T.J. Hazen, responsible for speech recognition, who also provided the confidence scoring algorithm, Joe Polifroni, who has played a role in developing the dialogue manager and has been heavily involved in data collection and performance evaluation, and Lee Hetherington, who developed the interface with Sabre. I am grateful to Victor Zue for carefully reading the paper, and providing many helpful comments. The diligence of two anonymous reviewers has led to significant improvements as well.

References

- Bangalore, S. & Rambow, O. (2000). Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany, pp. 42–48.
- Baptist, L. & Seneff, S. (2000). Genesis-II: a versatile system for language generation in conversational system applications. *Proceedings of the ICSLP '00, Beijing, China*, volume 2, pp. 271–274.
- Chu-Carroll, J. & Carberry, S. (1998). Response generation in planning dialogues. *Computational Linguistics* **24**(3).
- Galley, M., Fosler-Lussier, E. & Potamianos, A. (2001). Hybrid natural language generation for spoken dialogue systems. In *Proceedings of Eurospeech '01, Aalborg, Denmark*, pp. 1735–1738.
- Glass, J., Chang, J. & McCandless, M. (1996). A probabilistic framework for feature-based speech recognition. *Proceedings of the ICSLP '96, Philadelphia, PA*, volume 4, pp. 1–4.
- Glass, J., Polifroni, J. & Seneff, S. (1994). Multilingual language generation across multiple domains. In *Proceedings of the ICSLP '94, Yokohama, Japan*, pp. 932–986.
- Glass, J., Polifroni, J., Seneff, S. & Zue, V. (2000). Data collection and performance evaluation of spoken dialogue systems: the MIT experience. *Proceedings of the ICSLP '00, Beijing, China*, volume 4, pp. 1–4.
- Jonsson, A. & Dahlback, N. (1988). Talking to a computer is not like talking to your best friend. In *Scandinavian Conference on Artificial Intelligence-SCAI'88*, pp. 53–68.
- McDonald L., Bolc D.D. (Eds) (1998). In *Natural Language Generation Systems (Symbolic Computation Artificial Intelligence)*, Springer Verlag, Berlin.
- Moore, R. C., Dowding, J., Bratt, H., Gawron, J. M., Corfu, Y., Cheyer, A. (1997) CommandTalk: a spoken-language interface for battlefield simulations. *Fifth Conference on Applied Natural Language Processing, Washington, DC, April, 1–7*.
- Oh, A. H. (2000) Stochastic natural language generation for spoken dialog systems. MS Thesis, CMU.
- Oh, A. H. & Rudnicki, A. I. (2000). Stochastic language generation for spoken dialogue systems. In *Proceedings of ANLP/NAACL Workshop on Conversational Systems, Seattle, WA*, pp. 27–32.
- Pieraccini, R., Levin, E. & Eckert, W. (1997). AMIGA: the AT&T mixed initiative conversational architecture. In *Proceedings of EUROSPEECH '97, Rhodes, Greece, Sept.*, pp. 1875–1878.
- Portele, T. (2000). Natural language generation for spoken dialogue. *Proceedings of the ICSLP'00, Beijing, China*, volume 3, pp. 310–313.
- Reiter, E. & Dale, R. (2000). *Building natural language generation systems*, Cambridge University Press, Cambridge.
- Ratnaparkhi, A. (2000). Trainable methods for surface natural language generation. In *Proceedings of NAACL, Seattle, WA*, pp. 194–201.
- Seneff, S. (1992). TINA: a natural language system for spoken language applications. *Computational Linguistics* **18**, 61–86.
- Seneff, S., Goddeau, D., Pao, C. & Polifroni, J. (1996). Multimodal discourse modelling in a multi-user multi-domain environment. In *Proceedings of the ICSLP '96, Philadelphia, PA*, pp. 192–195.
- Seneff, S., Hurley, E., Lau, R., Schmid, P. & Zue, V. (1998). Galaxy- II: a reference architecture for conversational system development. In *Proceedings of the ICSLP '98, Sydney, Australia*, pp. 931–934.
- Seneff, S., Lau, R. & Polifroni, J. (1999). Organization, communication, and control in the GALAXY-II conversational system. In *Proceedings of Eurospeech '99, Budapest, Hungary*, pp. 1271–1274.
- Seneff, S. & Polifroni, J. (2000a). Dialogue management in the MERCURY flight reservation system. In *Proceedings of ANLP-NAACL 2000, Satellite Workshop, Seattle, WA*, pp. 1–6.
- Seneff, S. & Polifroni, J. (2000b). Formal and natural language generation in the MERCURY conversational system. *Proceedings of the ICSLP '00, Beijing, China*, volume 3, pp. 767–770.
- Seneff, S., Polifroni, J. (2001). Hypothesis selection and resolution in the MERCURY flight reservation system. *DARPA Human Language Technology Workshop, San Diego, CA*, pp. 145–152.

- Walker, M. A., Rambow, O., Rogati, M. (2001) SPoT: a Tractable sentence planner. *2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL '01)*.
- Walker, M., Whittaker, S. (1989). Mixed initiative in dialogue: an investigation into discourse segmentation. *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 70–78.
- Wang, C., Cyphers, D. S., Mou, X., Polifroni, J., Seneff, S., Yi, J. & Zue, V. (2000). MUXING: a telephone-access mandarin conversational system. *Proceedings of the ICSLP '00, Beijing, China*, volume 2, pp. 715–718.
- Yi, J. R. & Glass, J. R. (1998). Natural-sounding speech synthesis using variable-length units. In *Proceedings of the ICSLP '98, Sydney, Australia*, pp. 1167–1170.
- Yi, J. R., Glass, J. R. & Hetherington, I. L. (2000). A flexible, scalable finite-state transducer architecture for corpus-based concatenative speech synthesis. *Proceedings of the ICSLP '00, Beijing, China*, volume 2, pp. 322–325.
- Zue, V., Seneff, S., Polifroni, J., Nakano, M., Minami, Y., Hazen, T. J. & Glass, J. (2000). From Jupiter to Mokusei: multilingual conversational systems in the weather domain. *Proceedings of MSC2000*, pp. 1–6.

(Received 24 September 2001 and accepted for publication 26 April 2002)