Galaxy-II as an Architecture for Spoken Dialogue Evaluation

Joseph Polifroni and Stephanie Seneff

Spoken Language Systems Group
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA
{joe, seneff}@lcs.mit.edu

Abstract

The GALAXY-II architecture, comprised of a centralized hub mediating the interaction among a suite of human language technology servers, provides both a useful tool for implementing systems and also a streamlined way of configuring the evaluation of these systems. In this paper, we discuss our ongoing efforts in evaluation of spoken dialogue systems, with particular attention to the way in which the architecture facilitates the development of a variety of evaluation configurations. We furthermore propose two new metrics for automatic evaluation of the discourse and dialogue components of a spoken dialogue system, which we call "user frustration" and "information bit rate."

1. Introduction

Through our experience over the last decade in designing spoken dialogue systems, we have come to realize that an essential element in being able to rapidly configure new systems is to allow as many aspects of the system design as possible to be specifiable without modifying source code. To this end, we recently redesigned our core architecture to support complex system configurations controlled by a run-time executable scripting language. Using this new framework, which we call "GALAXY-II" (Seneff *et al.*, 1998), we have been able to configure multi-modal, multi-domain, multi-user, and multilingual systems with much less effort than previously. We are discovering that we can now configure systems whose capabilities are well beyond what was previously considered feasible[1].

As new and increasingly complex spoken dialogue systems are built, the task of evaluating these systems becomes both more important and more difficult. In the first place, component technologies are interdependent. Typically, a spoken dialogue system is comprised of multiple modules, each of which performs its task within an overall framework, sometimes completely independently but most often with input from other modules. Secondly, once a mechanism is in place for running data through an off-line system, a simple reprocessing of data with a new version of any component can lead to an incoherent interaction, as only one side of a two-sided conversation has changed. Finally, there is the question of what to evaluate (e.g., individual component vs. overall system behavior) and how (error rates vs. some measure of usability).

This paper describes our efforts to expand the applicability of the GALAXY-II scripting language, initially developed to configure dialogue systems that interface with users, to also support evaluation runs that assess the performance over time of both entire systems and specific system components. The focus of the paper will be on the resulting tools we have developed within the GALAXY-II architecture for evaluating spoken dialogue systems. These tools consist mainly of a suite of hub programs, which allow us to rapidly configure several different combinations of servers for various evaluation runs, including two specialized servers designed to augment evaluation capabilities. We will also discuss two new metrics we have devised for the automatic evaluation of the discourse and dialogue component of our systems.

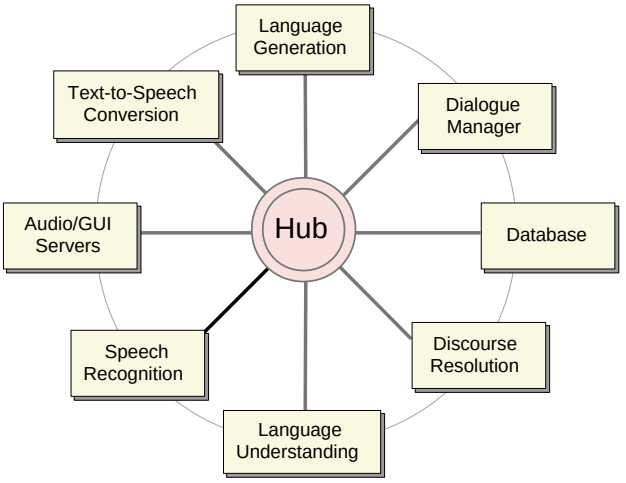

Figure 1: A typical configuration of a spoken dialogue system, showing the central hub and the various specialized servers.

2. Galaxy-II Architecture

The GALAXY-II (Seneff *et al.*, 1999) architecture consists of a central hub that controls the flow of information among a suite of servers, which may be running on the same machine or at remote locations. Figure 1 shows a typical hub configuration for a generic spoken dialogue system. The hub interaction with the servers is controlled via

[1] GALAXY-II has been designated as the initial common architecture for the multi-site DARPA Communicator project in the United States.

```
RULE:    :ref_tried & :rec_tried & !:rec_score --> evaluate_rec_string
LOG_IN:  :sro_string :rec_string
IN:      (:ref_string :sro_string) (:hyp_string :rec_string) \
         :eval_mode :sro_status :rec_status
OUT:     :rec_score :ref_string :hyp_string
LOG_OUT: :rec_score
```

Table 1: An example rule in a hub program that invokes the evaluation server operation, *evaluate_rec_string*. The rule provides a reference and hypothesis string as input, and records in a log file the resulting recognition score, as well as the pair of strings being compared.

a scripting language. A hub program includes a list of the active *servers*, specifying the host, port, and set of operations each server supports, as well as a set of one or more *programs*. Each program consists of a set of *rules*, where each rule specifies an *operation*, a set of *conditions* under which that rule should "fire," a list of INPUT and OUTPUT *variables* for the rule, as well as optional STORE/RETRIEVE variables into/from the discourse history. When a rule fires, the input variables are packaged into a *token* and sent to the server that handles the operation. The hub expects the server to return a token containing the output variables at a later time. The variables are all recorded in a hub-internal *master token*. The conditions consist of simple logical and/or arithmetic tests on the values of the typed variables in the master token. The hub communicates with the various servers via a standardized frame-based protocol.

The GALAXY-II architecture has proven to be a powerful tool for evaluation. It has made possible a wide range of system configurations specifically designed for monitoring system performance resulting in a suite of hub programs concerned with evaluation. In some cases, we are only interested in evaluating a particular aspect of system performance, such as recognition or understanding. In other cases we're interested in assessing the performance of the entire system, perhaps comparing a new version with the version that existed at the time a log file was first created. At other times we might be interested in looking at ways of measuring system performance as it relates to user satisfaction, along measurable dimensions. We also routinely run large numbers of queries through a system in a batch mode, to assure system robustness, particularly prior to the release of a new version.

In the following section, we will describe how we configure architectures that utilize these servers for various types of evaluation. Section 4 describes previous work in evaluating understanding accuracy and how the GALAXY-II architecture has enabled us to streamline the procedures. Section 5 describes new metrics we have devised to automatically evaluate overall dialogue performance. After summarizing other miscellaneous but significant aspects of evaluation, we conclude with a look towards the future.

3. Programming Evaluation Runs

We have been concerned for some time with developing and maintaining a way of continually evaluating our systems, both holistically and at the component level. The new hub-based architecture has enabled us to streamline the evaluation process by making it subject to the same hub programs that control all other system functions. It has also allowed us to augment our suite of evaluation metrics, since an evaluation server can take input from any of the other component servers within the system that we wish to evaluate.

The interaction among the servers in a spoken dialogue system can be quite complicated. We feel it is very important for off-line runs on training and development data to match as closely as possible the on-line system configuration. The transparency that the GALAXY-II architecture provides to the system developer makes it easy to accomplish this goal. The ability to program the interaction of servers means that these modules can run as servers the same way in both on-line and off-line (batch) modes. No explicit code changes or run-time flags are needed for the servers to run in evaluation mode. Furthermore, the changes necessary to configure a system for a specific type of evaluation are usually localized to a few lines in a top-level hub program, with the constituent programs being identical to those of a live system.

In order to perform these types of evaluation runs, we have developed two new servers that play an important role: a "batchmode" server and an "evaluation" server. The batchmode server stands in place of a user interface; it extracts appropriate inputs from a log file and initiates dialogue turns. The evaluation server tabulates performance statistics on a wide range of metrics, and writes a final summary into the resultant second-generation log file.

3.1. Batchmode Server

The purpose of the batchmode server is to process user queries through the system off-line. It operates from a variety of different inputs, including orthographic transcriptions, N-best lists, word graphs, parse frames, waveform files, and even system log files created from previous live interactions. A hub program can be configured to produce a log file using any of the above inputs, alone or in combination. A batchmode run often includes calls to a special evaluation server, as described below.

Every conversation with our live systems is recorded in a logfile, at a level of detail that is controlled by the hub program. The program supports the specification of any input or output variables to be written to the log, associated with each rule as it fires. A subsequent evaluation program informs the batchmode server which elements from the logfile are of interest in a particular run. For example, in assessing run-time performance, the batchmode server must extract both the selected hypothesis and the transcription of

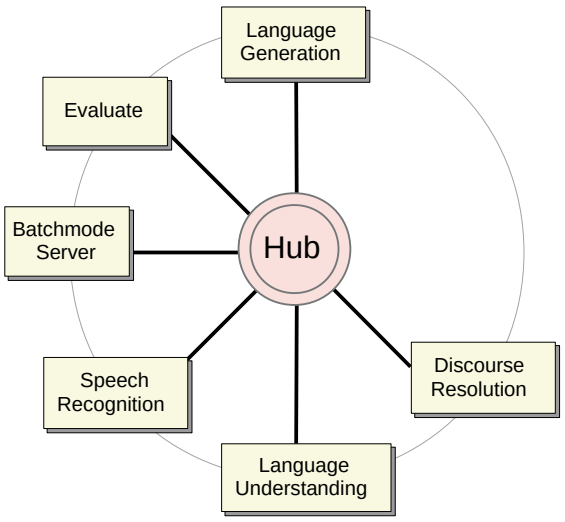

Figure 2: A GALAXY-II configuration showing the servers involved in evaluating recognition and understanding performance.

the user's speech from the logfile. The GALAXY-II architecture, combined with the hub scripting language, made control straightforward for this type of logfile evaluation.

We can also use the batchmode server to reprocess stored waveform files. In this case, the batchmode server behaves like an audio server, invoking the module-to-module communication protocol to connect to a recognizer. The recognizer processes the stored waveform file as it would any other utterance, i.e., producing either an N-best hypothesis or a word graph. This representation is sent back to the hub where it follows the path determined by a standard hub program for processing.

3.2. Evaluation Server

As mentioned previously, we have developed a separate evaluation server for performing comparisons and accumulating performance statistics. This server can determine both word error rate[2] and concept error rate, where the latter is based on an E-form representation of the understood user query. As will be discussed later in this paper, we have also developed two additional metrics that we feel may be useful for evaluating the discourse and dialogue components as well as the recognizer and understanding components. We call these the "user frustration" measure and the "information bit rate." The tabulations needed to compute these measures are maintained in the evaluation server.

We can easily configure hub programs that run multiple versions of the same server, to compare new versions of the recognizer against old ones, for example, or to compare two versions of a particular grammar. Furthermore, the rules of a hub program provide a clear tabulation of the parameters being evaluated. An example of a rule invoking an evaluation server operation is given in Table 1.

[2]Using the standard National Institute of Standards and Technology scoring algorithm as a library for this purpose.

4. Automatic Methods for Understanding Evaluation

In (Polifroni *et al.*, 1998) we proposed an E-form evaluation metric, which compares an E-form obtained by parsing the original orthography against that obtained by parsing the selected recognizer hypothesis. At that time, the evaluation process was fragmented into a number of sequential isolated steps. The original recognition outputs were retreived from a session log file. New recognition outputs were created through a stand-alone process and saved out to a file. The interaction between the recognition and NL components, currently mediated by the hub, had to be essentially simulated in a special stand-alone process. This process included the natural language library and performed the parsing and E-form generation steps. The process of synchronizing with the context information recorded in the original log file and the new recognition outputs was cumbersome and idiosyncratic.

Since we believe that E-form evaluation is a powerful metric for monitoring the performance of the recognizer and the parser, it has served as a good test case for the feasibility of using hub programs to run evaluation procedures. We found that the GALAXY-II architecture provided effective tools to streamline and generalize the process.

For assessing overall system understanding, we have written a hub program that first uses the batchmode server to process a logfile utterance-by-utterance, sending both a hypothesis and an orthographic transcription to the hub, with subsequent routing to TINA (Seneff, 1992) and GENESIS (Glass *et al.*, 1994), our natural language understanding and generation components, respectively. Once the appropriate inputs are created, the hub program sends them to the evaluation server, where they are used to assign scores. The results are returned to the hub program along with all other relevant data for a particular utterance for logging purposes. The evaluation server also outputs cumulative statistics at the end of each batch run. An appropriate hub configuration for performing this type of evaluation is shown in Figure 2. Notice that the batchmode server replaces the audio server in this figure, and that various other servers concerned with processing beyond the NL component are missing.

The hypothesis can be either the original one produced at the time the data were collected, or a new one produced with an updated version of the recognizer and/or of the parser. Two different configurations can be run in parallel to help assess which one is exhibiting a superior performance. Among the evaluation experiments we have run are: (1) logfile recognition/understanding performance at run-time, (2) word graphs compared with N-best lists, and (3) comparisons of old and new versions of a grammar. We have been able to use the same hub program for evaluating many different types of input conditions, by simply adjusting a few top-level variables.

5. Automatic Methods for Dialogue Evaluation

We have found that it is extremely useful to be able to rerun data through dialogue systems to monitor progress. We have thus acquired large corpora of speech data, transcriptions, and dialogue session logs, all of which are used

for confirming that new versions of our systems are healthy prior to their release. A minimal test is that there are no catastrophic failures (e.g., server crashes). More detailed confirmation of performance can be obtained in some cases through direct string comparisons of system responses, although care must be taken to ensure repeatability. It is also productive to examine the outputs of the batch runs manually on an utterance-by-utterance basis, although this is inefficient. We have therefore been motivated to establish parameters that can capture quantitative performance automatically. In this section, we first discuss some of the issues that arose in reprocessing logged data, and then describe the automatic metrics we devised for dialogue evaluation.

5.1. Issues in Reprocessing Data

The chief purpose of reprocessing data is to monitor system performance, in order to verify that a new version of the system is functioning well. With real database-query systems, this becomes problematic because they provide timely information that is dynamic in nature. For example, as a flight database changes, queries based on the flights available at data collection time can become incoherent. Furthermore, users frequently ask about flights in the very near future, which then become past events when the system is rerun at a later time. Finally, the outputs of batch runs are subject to incoherence due to changes over time in the dialogue model, or simply improvements in recognition and understanding.

We have developed various mechanisms for dealing with these problems. One such problem is the changing nature of the information we provide to users. Weather information for our Jupiter domain (Zue *et al.*, 2000), though timely, did not pose as much of a problem in this regard, since we harvest the weather information several times per day, parse it into a meaning representation that the system backend can understand, and then store a representation of the data in a relational database. It is not difficult to periodically store a frozen version of the database tables, along with ancillary files containing weather information, for future processing. We then need only change the pointers to files used by the database server when processing data in batch mode. We also have to inform the Jupiter backend of what date to consider as "today" so that references to relative dates in user queries are properly interpreted by the backend. We then expect identical answers for repeatable runs, aside from any changes that may have occurred in the system during the intervening development period.

We were faced with a bigger problem in reprocessing queries to our Pegasus flight status system and our Mercury flight schedule system (Seneff & Polifroni, 2000). To answer these queries, we access information dynamically, from content providers who either continually update a locally maintained database or who give us privileged access to their information. These data are impossible to freeze, since they either change on a minute-by-minute basis (i.e., the flight status information) or represent too large and complex a corpus to characterize and access completely (i.e., flight schedules for hundreds of cities worldwide). However, these data are extremely timely and the type and nature of follow-up queries are completely dependent on the information that the user heard at the time of data collection.

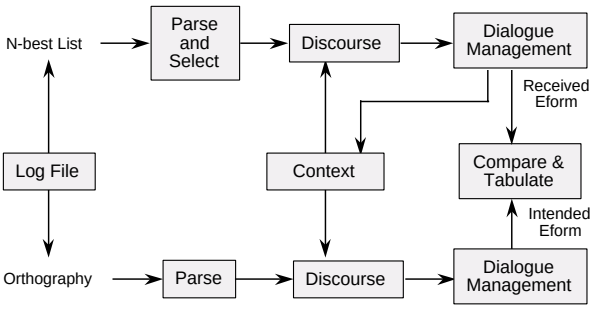

Figure 3: A flow graph of the procedure for computing information bit rate and user frustration.

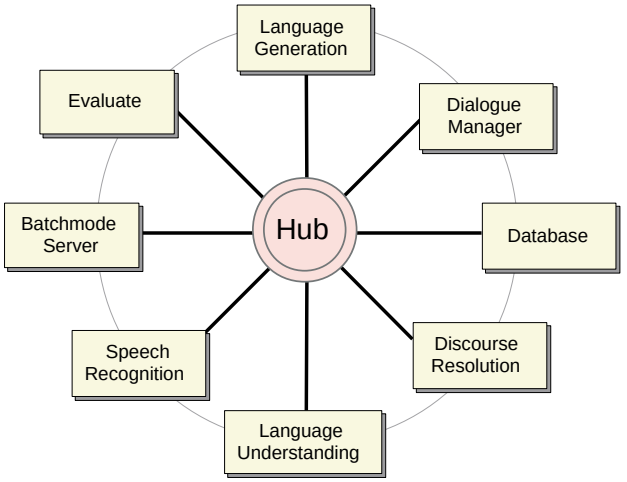

Figure 4: A GALAXY-II configuration showing the servers involved in evaluating discourse and dialogue performance using information bit rate and user frustration measures (see text for details).

In the flight reservation domain, another serious problem with reruns is that the dates the user specified quickly become stale due to elapsed time. One solution we have found is to artificially offset all date references during the evaluation phase by N weeks, where N is a number guaranteed to place all dates in the future, relative to the reprocessing date. We still may run into other problems due to seasonal changes in flight availability, and it is unclear how long we can continue to reprocess data in this way given that flight numbers and even airlines change. For example, in one case the user said, "Book it," in response to a single flight being proposed, but due to changes in the flight schedule, the system proposed three flights in the rerun and the dialogue became incoherent from that point on. However, we have been reasonably successful up to now by simply moving the date forward and maintaining the same day of the week. In this case, of course, a simple string compare on the responses will no longer be viable as a means of assuring system stability; instead, system developers must examine the script of the second-generation dialogue to assess the system's capabilities.

1a U:	I'D LIKE TO FLY FROM SEATTLE TO CHICAGO ON DECEMBER TWENTY SEVENTH
1b M:	*From Seattle to Chicago on December 22nd. Can you specify a time or airline preference?*
2a U:	I SAID DECEMBER TWENTY SEVENTH
2b M:	*From Seattle to Chicago on December 27th. Can you specify a time or airline preference?*

Table 2: Example of a short dialogue containing an error in the Mercury flight travel domain. U = User, M = Mercury.

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

Table 3: Distribution of evaluable user utterances in terms of number of new attributes introduced with each dialogue turn. IBR = Information Bit Rate.

5.2. Proposed Evaluation Metrics

We have long been interested in seeking automatic evaluation metrics that can apply on a per-utterance basis but evaluate a significant portion of the system beyond the recognizer. To this end, we recently devised two new evaluation metrics, which we believe are useful measures for assessing the performance of the recognizer, parser, discourse, and dialogue components, collectively. To compute the measures, we must reprocess the log file after the orthographic transcription has been provided for the user queries. As illustrated in Figure 3, both the recognizer hypothesis and the original orthography are run through the system utterance by utterance, with the discourse and dialogue states being maintained exclusively by the recognizer branch. For both branches, the E-form that is produced after the dialogue manager has finished processing the query is sent to the evaluation server. This server maintains a running record of all the attributes that appear in the orthography path, comparing them against their counterparts in the recognizer path. Figure 4 shows a hub configured for this type of evaluation. The only server that is not represented at least functionally in this diagram, as compared with a regular system configuration, is the speech synthesis server. The batchmode server here can provide either a waveform or an N-best list for processing by the NLU component.

The two parameters that emerge from comparing these E-forms we refer to as information bit rate (IBR) and user frustration (UF). IBR measures the average number of new attributes introduced per user query. A subsequent query that reiterates the same attribute is excluded since it does not introduce any new information. The UF parameter tabulates how many turns it took, on average, for an intended attribute to be transmitted successfully to the system.

Take, for example, the dialogue in Table 2. A recognition error occurs on the date in turn 1, where the system misrecognizes "december twenty seventh" as "december twenty second." A subsequent dialogue turn is required to repair this error. In the scheme outlined above, the first utterance introduces three new concepts (i.e., source, destination, and date). The second utterance introduces none, thus contributing a 0 count to the IBR parameter. For the user frustration parameter, source and destination each took one turn, but the date took two.

In a pilot study, we processed a subset of our data through this evaluation configuration. We identified a set of 17 attributes that could be monitored. Five percent of the utterances had orthographies that failed to parse. These are unevaluable without human reannotation, and are hence eliminated from the pool in the discussion below, although they clearly are likely to be very problematic. Table 3 summarizes the results for information bit rate for the remainder of the utterances. The average information bit rate was 1.46 concepts per utterance. A surprisingly large percentage of the utterances introduce no new concepts. Some, but not all, of these are similar to the date misrecognition example given above. Others are cases where the user was confused about the state of the system's knowledge, and decided to simply repeat all the preceding constraints just to make sure. Some are also misfirings of the endpoint detector producing content-free utterances such as "okay." In other cases the user intended an action, but the system's understanding mechanism was not sophisticated enough (e.g., "That's good" meaning "book it"). We were encouraged by the percentage of sentences that contained more than one attribute. We believe that a typical directed dialogue would have far fewer utterances with more than one attribute.

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

6. Miscellaneous Features

There are a number of aspects of the GALAXY-II architecture that enable better troubleshooting capabilities and the flexibility to configure partial systems to focus on particular components. Here we briefly discuss two such examples.

It turned out to be relatively straightforward to use a hub program to provide spoken feedback to users and developers when a major system error occurs. When a particular server crashes, the resulting disconnect with the hub causes an abort message to be generated. The hub then sends a message to the audio server, to inform it to disconnect the call. Rather than abruptly terminating the call, however, the audio server is able to relay to the user a message providing information about which server caused the problem, along with an apology (e.g., "I'm sorry. Our content provider is currently unavailable. Please call back later.").

The flexibility of the GALAXY-II architecture has also enabled us to develop hub programs for development runs to evaluate any aspect of system development. We recently needed to check the output of a synthesizer that was under development in our group (Yi & Glass, 1998). By scripting the hub session to take typed input and produce and speak a synthesized answer through local audio output, we were able to quickly cycle through a set of queries to target the sorts of responses we were interested in checking, without involving a phone line. We can also use hub scripts to bypass the initial stages of the system entirely and run a session from reply frames only. These reply frames are sent to the NL component for generation and then on to the

synthesizer for speaking. We are able to check hundreds of different responses in one session in this way. We are using this facility, for example, to test the pronunciation and linguistic well-formedness of responses in the Jupiter weather domain produced in Spanish.

7. Summary and Future Work

We have found that the GALAXY-II architecture has provided a unified mechanism for performing the many different types of evaluation required for monitoring and understanding the performance of a complex system. We plan to extend our use of the GALAXY-II architecture in several ways. One is in monitoring the performance of on-line systems. A hub program could, for example, notice that a session is particularly problematic (e.g., by noticing a large number of help or error response messages) and send mail to system developers with a pointer to the logged record of the session. Furthermore, the hub maintains a record of the state of each particular server within a given configuration. Though we have developed a "keep-alive" mechanism for insuring that all servers come back to life after crashes, it is useful to know when these crashes occur and what state the system was in at the time. The hub, with its knowledge of each particular server, could send mail to system developers in cases of server crashes, as well. One final way we are looking into adding to our evaluation suite is in the multilingual versions of our various systems. Because the hub is able to mediate a seamless switch among all the languages under development, we feel it will be useful in comparing and evaluating the performance of each.

8. Acknowledgements

This research was supported by DARPA under contract N66001-99-1-8904, monitored through Naval Command, Control, and Ocean Surveillance Center.

9. References

Glass, J., J. Polifroni, and S. Seneff, 1994. "Multilingual Language Generation across Multiple Domains" *Proc. ICSLP, '94*, 983–986.

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

Seneff, S., 1992. "TINA: a Natural Language System for Spoken Language Applications," *Computational Linguistics*, 18:61–68.

Seneff, S., E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, 1998. "GALAXY-II: A Reference Architecture for Conversational System Development," *Proc. ICSLP '98*, 931–934.

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

Seneff, S., and J. Polifroni, 2000. "Dialogue Management in the Mercury Flight Reservation System," *Proc. ANLP-NAACL2000 Workshop Workshop on Conversational Systems*, to appear.

Yi, J.R.W., and J. Glass, 1998. "Natural-sounding Speech Synthesis Using Variable-length Units," *Proc. ICSLP '98*, 1167-1170.

Zue, V., S. Seneff, J. Glass, J. Polifroni, C. Pao, T.J. Hazen, and L. Hetherington, 2000. "JUPITER: A Telephone-Based Conversational Interface for Weather Information," *IEEE Trans. on Speech and Audio Process*, 8:85–96.