Development and Preliminary Evaluation of the MIT ATIS System[1]

Stephanie Seneff, James Glass, David Goddeau, David Goodine, Lynette Hirschman, Hong Leung, Michael Phillips, Joseph Polifroni, and Victor Zue

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

This paper represents a status report on the MIT ATIS system. The most significant new achievement is that we now have a speech-input mode. It is based on the MIT SUMMIT system using context independent phone models, and includes a word-pair grammar with perplexity 92 (on the June-90 test set). In addition, we have completely redesigned the back-end component, in order to emphasize portability and extensibility. The parser now produces an intermediate semantic frame representation, which serves as the focal point for all back-end operations, such as history management, text generation, and SQL query generation. Most of those aspects of the system that are tied to a particular domain are now entered through a set of tables associated with a small artificial language for decoding them. We have also improved the display of the database table, making it considerably easier for a subject to comprehend the information given. We report here on the results of the official DARPA February-91 evaluation, as well as on results of an evaluation on data collected at MIT, for both speech input and text input.

INTRODUCTION

In June 1990, we reported on the initial development of the MIT ATIS system, and participated in the first round of DARPA common evaluation using text input [5]. Since then, a number of changes have been made to our system, particularly the back-end component that transforms the parse tree into a representation that can be used to maintain discourse, generate confirmation messages, and produce SQL queries for accessing the OAG database. We have also connected the SUMMIT speech recognizer to our ATIS system, so that it can now accept verbal input.

This paper gives a progress report on the MIT ATIS development, with particular emphasis on how the system processes an input utterance to achieve understanding and generate responses. We will also report on the evaluation of the system for both text and speech input, using the data provided by TI through NIST, as well as the data that we have collected over the past few months [3]. Aspects of the system involving discourse and dialogue are based on similar principles as before, but has been modified to reflect the new semantic representations. A detailed description of our discourse model can be found in a companion paper [4].

[1]This research was supported by DARPA under Contract N00014-89-J-1332, monitored through the Office of Naval Research.

SYSTEM DESCRIPTION

In this section we will describe those aspects of the system that have changed significantly since our report last June [5]. The most significant change has been the incorporation of the speech recognition component. We begin by describing the recognizer configuration and the interface mechanism we are currently using. In the natural language component, the parser and grammar remain unchanged, except for augmentations to improve coverage. However, we have completely redesigned the component that translates from a parse tree to executable SQL queries, and the component that generates verbal responses. Both of these areas are described here in more detail.

Speech Recognition Component

The speech recognition configuration is similar to the one used in the VOYAGER system and is based on the SUMMIT system [6]. For the ATIS task, we used 76 context-independent phone models trained on speaker-independent data collected at TI and MIT [3]. There were 1284 TI sentences (read and spontaneous versions of 642 sentences) and 1146 spontaneous sentences taken from the MIT training corpus. The lexicon was derived from the vocabulary used by the ATIS natural language component and consisted of 577 words. In order to provide some conservative natural language constraints, the speech recognition component used a generalized word-pair grammar derived from the speech training data augmented with a large number of additional sentences pooled from all available sources of ATIS related text material. The word-pair grammar was generated by parsing each sentence, and then generalizing each word in a terminal node to all words in the same semantic class. Thus for example, an instance of the word "Boston" would generalize to all cities. In the

case where a sentence did not parse, no additions were made to the word-pair grammar. When evaluated on the TI June-90 data set of 138 sentences, the word-pair grammar had a coverage of 70% and a perplexity of 92. Overall, 3.6% of the sentences that parsed failed to pass the word-pair grammar.

The interface to the natural language component was implemented with the N-best mechanism we have described previously for the VOYAGER system [6]. In our original implementation, the first N-best output which parsed was used by the back-end to generate a response. Since our natural language component (TINA) is able to produce a parse probability derived from training data, we have tried to make use of the probability in the selection of the N-best output. In both the VOYAGER and ATIS domains we have found that a linear combination of the acoustic score produced by SUMMIT and the parse score produced by TINA improved the overall system performance [1]. In ATIS the improvement in recognition accuracy was about 2% on the TI June-90 data set.

In order to control the number of false alarms produced by the system, we investigated the use of several pruning measures which could be applied to the N-best outputs. To date we have found the N-best rank and the relative acoustic score (relative to the first choice output) to be effective parameters.

The ATIS back-end

After reassessing the status of our ATIS system last June, we were concerned that the design of the back-end component might not be as easily extended or ported to new domains as we would like. We therefore decided to redesign the system, with the goal of emphasizing both system modularity and system portability. In choosing a design for the new system, we had two major goals. One was to design a semantic frame representation that would capture all necessary information from the sentence and serve as a focal point for all components of the back-end. The frame design should be flexible enough to be able to extend to other domains. The second goal was to provide a mechanism that would permit the domain-dependent aspects of the system to be entered completely through table-driven mechanisms, without requiring any explicit programming.

Processing of a sentence involves several steps. The first step is to provide a parse tree for the input word stream. A second-pass treewalk through the parse tree yields a semantic frame, which is then integrated with available frames from the history. Both an SQL query and a generated text response are derived from the completed frame. The verbal response is spoken to the subject and a table is retrieved from the database through the database management system ORACLE. A table post-processing step converts the table to a much more readable and informative form prior to display. Finally, the system examines the goal plan and optionally initiates an additional response, based on its assessment of

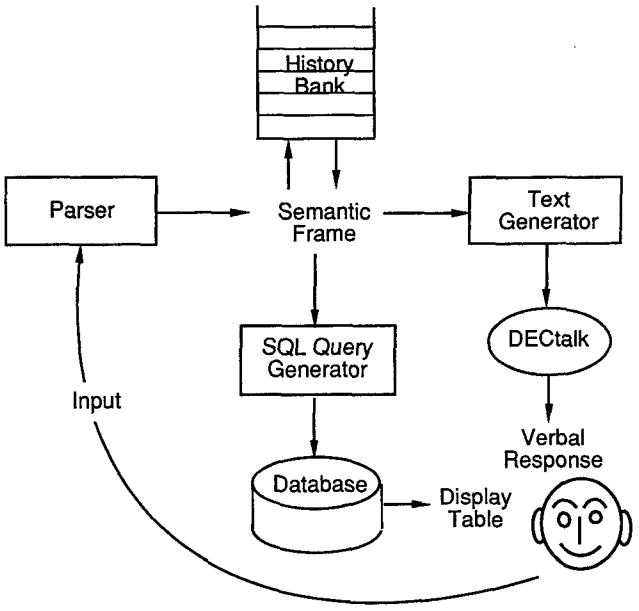

Figure 1: Block diagram of system structure showing central role of semantic frame.

a likely next step. A thorough description of dialogue and discourse aspects of the system along with an example flight reservations dialogue can be found in [4].

The Semantic Frame: The parse outputs of TINA are first converted to a semantic frame representation which serves three critical roles, as shown in Figure 1: it is translated to SQL through table-driven pattern matching devices, it is delivered to a text-generation program to construct appropriate verbal responses, and it serves as input to the discourse history used to restore implicit information in subsequent queries and resolve explicit anaphoric references.

Each frame is associated with a name, a type, and a set of (key: value) pairs. The value can be an integer, a string, a symbol, another frame, or a set of frames. There are only a small number of possible types of frames, such as *clause*, *predicate*, *qset* (for common noun phrases), *reference* (for proper nouns), and *quantifier*. The type *reference* always has a special key *reftype* associated with it, identifying the class of proper nouns it belongs in (i.e., *city-name* would be the *reftype* for "Boston.")

Conversion of Parse Tree to Semantic Frame: The process of producing a semantic frame involves a second-pass tree walk through a completed parse tree. Only the names of the nodes are needed, because of the semantic nature of the grammar. In the tree walk, nodes pass along frames, modifying them if necessary according to the node's seman-

tic significance. A completed semantic frame is ultimately returned to the top-level sentence node and delivered as-is to the back end for further processing.

About half of the nodes in the ATIS grammar have no semantic significance, and hence they simply pass along to their children and later to their right sibling whatever was delivered to them. Each of the active nodes is associated by name to a particular semantic name, which is often the same as its "given" name. Each semantic name is in turn associated with a particular functionality. There are fewer than twenty possible functions, and during the tree walk, the particular function to choose is dictated by the association. Each function is called with three arguments: the semantic name, the subparse tree and the current frame.

A simple example may help to clarify this process. The node named *dir-object* is associated with the semantic name *theme* which calls the function *process-noun-phrase*. This function, during the top-down cycle, creates an empty frame of type *qset* and inserts it into the current frame under the key *theme* as specified by the argument. It then passes the empty frame along to its children, who will fill it in. Finally, it passes the original frame to its right siblings, with a completed entry under the key *theme*.

Decoding the Frame: A completed semantic frame is passed to the back-end for interpretation. The top-level frame is always of type *clause*, and its name determines a particular clause-level analysis function to be executed. Options include *request*, *statement*, *yes-no-question*, *clarifier*, etc. For example, the function for a yes-no question makes two separate calls to the database. The first one determines the set of all objects as specified by the topic, and the second one finds the set defined by the topic restricted by the predicate (or complement). A final step seeks a non-null intersection between the two sets. There are three possible types of response, namely "There is no <topic>," "Yes, <topic> does do <predicate>," and "No, <topic> does not do <predicate>." Thus, to answer the question, "Does the earliest flight serve lunch?" the system finds both the earliest flight and the earliest flight that serves lunch, and determines whether they are the same flight.

In addition to the high-level interpretation of clauses, some low level routines serve to reorganize certain information in the frame as delivered by the parser. For example, there are many modifiers which can be attached to either flights or fares. We decided that it would be easier for later processing if all fare modifiers are physically transferred to a flight object, which is created if it didn't exist explicitly in the sentence. Thus if the person says, "Show fares from Boston to Denver," the sentence is converted into: "Show fares *for flights* from Boston to Denver." In addition, phrases about time and date are regularized and turned into absolute references. Thus "the following Wednesday," is decoded as "the date which is on the subsequent Wednesday to the

```
  Frame Format:
 [name type
  key1: value1
  key2: value2
        ...]
  Frame:
[request clause
 predicate: [display predicate
             for-poss:["me" reference reftype: pronoun]
             theme: [fare qset
                     for: [auto qset
                           car-type: limousine
                           to:  ["oakland" reference
                                 reftype:city-name]]]]]
```

Figure 2: Semantic frame representation of the sentence, "Show me the price of a limousine to Oakland."

date stored in the history table." After the frame is properly restructured, it is sent off to the discourse module, which augments noun phrases (mainly flights and fares) with appropriate modifiers from the history.

SQL Query Generation Mechanism: All of the domain-dependent information needed to map frames into SQL queries is contained in a small set of tables, which are decoded through a simple artificial language involving a small number of special operations. The basic unit of recognition is a pattern containing (*name* (*key* value-type)), where *name* is the name of the parent frame, and value-type is the uniquely defined identifier for the value associated with the *key*. For example, the value-type of a *qset* is simply its name, the value-type of a *reference* is its *reftype*, and the value-type of a string is STRING.

We will explain the interface between the semantic frame and the back end by walking through a simple example. The semantic frame derived from the sentence, "Show me the price of a limousine to Oakland," is given in Figure 2, and the table entries needed to decode that frame are shown in Figure 3. The final SQL query generated is given in Figure 4.

The top-level *display-table* defines a set of elements to be displayed and the set of database tables in which to find these elements. For our simple example, the instructions are to display all elements in the *ground_service* table, given a qset named *fare* with a *for* key whose value is a qset named *auto*. The final set of elements and tables to be displayed is constructed as the union of all sets whose patterns are matched in *display-table*. In some cases, entries from multiple tables must be displayed, and for these cases there is an additional table that defines how to link the two database tables.

The *qset-table* contains a set of patterns particular to frames of type qset, which trigger the augmentation of a simple database SQL query with a set of *where-clause*'s. The system processes a top-level qset through recursive processing of possible nested qsets. In our example, both the top-

```
One entry in the "display-table":
  ((fare  (for AUTO)) ground_service (*))

Three entries in the "qset-table":
  ((fare (for AUTO))
    (add-where-clauses () $1))
  ((auto (car-type STRING))
    (= transport_code (use-table auto-table)))
  ((auto ((from to in) (CITY CITY-CODE CITY-NAME)))
    (in city_code (cvt CITY-CODE $1)))

One entry in the "conversion-table":
  ((CITY-NAME CITY-CODE)
    (sql city city_code () ((= city_name $1))))

An auto-table:
(("taxi" T)("limousine" L)("air-taxi" A)
 ("rental-car" R)("car" R))
```

Figure 3: Table entries needed to decode the sentence, "Show me the price of a limousine to Oakland," whose semantic frame is shown in Figure 1

level *fare* and the *auto* entry under the *for* slot are qsets. The entry under *fare* that matches this pattern instructs the system to add to the parent query all the where clauses that are generated by the *auto* qset where the special code $1 stands for the argument.

There are two entries under *auto* that are activated by our frame. The one matched by *car-type* constructs the *where-clause* for the unit: "where transport_code = 'L' " and the one under the key *to* constructs the *where-clause* for the *city_code*. The decoding of the city "Denver" is done through the *conversion-table,* keyed by the special operator *cvt* (convert). The operator *sql* in *conversion-table* triggers the construction of another SQL query, "select distinct city_code from city where city_name = 'DENVER,' " which is inserted into the *where-clause* for *city_code* in *ground_transport*. What is constructed through this decoding step is not the actual string appropriate for calling the database, but rather a hierarchy of structures representing queries and where clauses, which can be converted to the query string through a *print-query* function, resulting in the SQL command shown in Figure 4.

The Table Display: We felt that in many cases the raw information from the database would not be readily comprehended without a further transformation. Therefore, we wrote a set of conversion routines associated with each column heading that would make the table easier to understand. Thus a clock time would be converted from "1426" to "2:26 P.M.", an airline name from "DL" to "Delta", and a fare class from "QX" to "QX: coach class discounted weekday." In some cases, we felt the database column was sufficiently

```
select distinct * from ground_service
where transport_code = 'L'
and city_code in
  (select distinct city_code from city
   where city_name = 'OAKLAND')
```

Figure 4: The SQL query for the sentence "Show me the price of a limousine to Oakland."

confusing that it was better to leave it out altogether, especially in cases where the text response redundantly carried the information. For instance, we never display the column "flight days," since the verbal response will always say, "on Tuesday" when appropriate. Likewise, we omit the flight-code column because it invites the user to refer to flights by their flight code using unpredictable language constructs. Our paper on database collection [3] discusses the effects of this transformation on solicited speech.

Verbal Response: A completed semantic frame is sent to a text generation program along with the database table indicating the answer. Text generation is mostly guided through tables, associating keys with both a print function and a positional specification within the parent frame's overall scheme. For example, adjectival modifiers *precede* the main noun, a flight-number *immediately follows* the main noun, and a post-modifier such as a relative clause or a gerund occurs at the *end.* Clause level generation is done through specialized functions, each associating with a particular clause type, such as *yes-no-question.* The database table is used both to infer what should be said at the top-level, and to determine whether the noun phrase is singular or plural. Thus, for example, an existential clause would be required to produce one of, "There is" There are" or "There are no" preceding a noun-phrase describing the intended flight set. When a person asks a wh-query, such as "What meals do these flights serve?" the system detects the trace under the object of the verb "serve" and inserts the canned phrase, "the following meals" into the verbal response. The database table is then displayed providing the answer in a meals column.

Other Aspects of the System: There are two other major components of the system that have not yet been discussed. These are the discourse history management system and the dialogue component. Both of these are described in detail in [4] and therefore will only be briefly mentioned here.

Discourse is managed through a history table containing several types of elements derived from the previous sentences, including both semantic frames identifying named objects such as flights and dates, display tables from the database, and, in the case of bookings, previous states of the ticket. Most of the history revolves around a flight-event object. Modifiers are inherited from the history either if they are not explicitly mentioned in the current frame or if no

"masker" modifiers are present. For each history modifier, a set of maskers is specified in a table. We determined the masking conditions based on experience with real data. For example, if the subject asks about "non-stop" flights, then a connection-place would not be inherited. The most complex history management involves references to "return flights," in which a previously mentioned source and destination must be "swapped," unless the previous sentence also concerned return flights. In addition, only fare restrictions and airline should be inherited, along with source and destination. Any previous references to a date or a flight number would be dropped when talking about return flights.

The computer essentially always gives a verbal response to the subject's question identifying the contents of the displayed table. Dialogue is maintained through a *dialogue state stack* which is popped and evaluated after each input sentence is fully processed. A clear division is kept in the computer code between the subject's half of the conversation and the computer's half. During the analysis of the subject's contribution, the dialogue *state* may be modified, but none of the dialogue execution routines are called. Most of the time the dialogue stack is empty, and it rarely contains more than one previous state. Dialogue is used mostly during bookings, which involve a complex interplay between the subject and the computer. For example, if the subject says, "Book the cheapest flight." the system must remember that a booking is underway, but must first ask whether the subject wants a one-way or round-trip fare. Hence the stack becomes two-deep at this point.

EVALUATION

Table 1 summarizes our results for the three obligatory system evaluations using the February-91 test set provided by NIST. The first test takes as input the transcriptions of the so-called Class A sentences, i.e., sentences that are context independent, and produces a CAS[2] output. The second test is the same as the first one, except that the sentences are Class D1, i.e., their interpretation depends upon a previous sentence, which is provided as additional input. The last test is the same as the first, except that the input is speech rather than text. For each data set, we give the percent correct, percent incorrect, percent with no answer, and the overall score, where the score penalizes incorrect answers weighted equally against correct answers.

Comparing the first row of Table 1 with last June, our current implementation makes considerably fewer false alarms for text input. The two errors that the system made were due to a minor system bug; while the correct answer was displayed to the user locally, we inadvertently sent the wrong one to the comparator. We are encouraged by this result, since the errors were all due to factors unrelated to the development of the natural language technology, and as such can be fixed trivially.

Data Set	No. of Sentences	Correct (%)	Incorrect (%)	No Answer (%)	Score (%)
Class A Text	145	56.6	1.4	42.1	55.2
Class D1 Text	38	47.4	5.3	47.4	42.1
Class A Speech	145	31.7	13.1	55.2	18.6

Table 1: Results for the standard ATIS test sets for three test conditions.

The results for the context-dependent sentences are given in column 2 of Table 1. Our system provided correct answers for 18 of the 38 context pairs, and made only 2 errors. This is a more stringent test than the first one, since providing the correct answer in this case demands that *both* sentences be correctly understood. One of the errors was due to the same system bug describe above, i.e., the right answer was displayed but not sent. In the second one, which we considered to be the only error made by the natural language system, the system simply ignored the context.

The results for the Class A sentences with speech input are given in Column 3 of Table 1. Of the 19 sentences that provided an incorrect answer, 2 were correctly recognized, but failed due to the system bug mentioned above.

We recently collected a sizable amount of spontaneous speech data, using a paradigm very different from the one used at TI. Our preliminary analyses of the two data sets have indicated significant differences in many dimensions, including the speaking rate, vocabulary growth, and amount of spontaneous speech disfluencies [3]. We thought it might be interesting to compare our system's performance on the two data sets. To this end, we asked B. Bly of SRI to help us generate the CAS reference answers for the designated development-test set of our database. The test set consists of 371 sentences, of which 198 were classified by Ms. Bly as Class A. Since no aspects of our system had been trained on these data, we consider it to be a legitimate test set for purposes of this experiment, although we plan to use it in the future as a development test set.

The results for CAS output with both text and speech input for the MIT data are given in Table 2. We should point out that in an initial run of the text-input condition, several answers marked as "incorrect" were judged by us to be dubious. We submitted these questionable answers to Ms. Bly, as part of the customary follow-up process of "human adjudication" established at NIST. The results in Table 2 thus represent the final outcome. Some of the discrepancies were due to an error on the part of the reference answer, several were due to the "yes/no" *vs.* table problem, several were due to the fact that SRI assumed 1990 for all dates, whereas most

[2]CAS, or Common Answer Specification, is a standardized format for the information retrieved from the OAG database, which is compared against a "reference" CAS using a comparator provided by NIST.

Data Set	No. of Sentences	Correct (%)	Incorrect (%)	No Answer (%)	Score (%)
Class A Text	198	74.2	1.5	24.2	72.7
Class A Speech	198	39.9	8.1	52.0	31.8

Table 2: Results for 198 Class A utterances taken from the MIT Development Test Set, with CAS reference answers provided by SRI, for both text and speech input.

of the dates were actually in January of 1991.

For the text-in/CAS-out test condition, we obtained an overall score of 72.7%, which is a dramatic improvement over our results on the TI data. In two of the three errors, the back-end ignored certain critical modifiers in the frame. The third error was fairly subtle: we interpreted "I'd like to book a flight between Boston and San Francisco with stops in Denver and Atlanta," to mean *or* rather than *and* for the stops.

We produced a correct answer for almost 40% of the utterances when the speech recognizer was included in the system, with an 8% false alarm rate. This gave an overall score of 32%, which is again substantially higher than the 18.6% score we received for the recognizer results on the standard test set. The MIT test was run after some bug fixes, which would have improved the score for the TI data to 24% (see Table 3). However, this is still substantially lower than the score for the MIT set. This is all the more surprising since the MIT test data were not prescreened for speech disfluencies[3] – we included *all* of the Class A sentences of each test speaker. There are several possible explanations for the discrepancy. We believe that the MIT sentences are spoken more fluently, as suggested by the results of a statistical analysis reported in [3]. We also suspect that MIT subjects tend to use constructs that are more straightforward and conform more closely to standard English. Finally, the MIT sentences include very few table clarification questions, a feature which allowed us to reduce the size and perplexity of our grammar.

In general, the speech recognition error rate for our system is significantly higher in the ATIS domain than what we have experienced with the Resource Management domain. One conclusion we may draw is that spontaneous speech, with out-of-domain words and novel linguistic constructs, can combine to degrade recognition performance drastically. There are at least several other reasons that contribute to our increased speech recognition error rate. The version of our recognizer in the ATIS system used only context-*independent* phoneme models. This is because we have focused our research attention in speech recognition primarily on the Resource Management task, and did not really devote any effort to the ATIS domain until late January. The word-pair language model that we developed has a coverage of only 50% of

[3]The TI class A sentences were screened to exclude disfluencies for the basic test set.

Data Set	No. of Sentences	Cor (%)	Sub (%)	Del (%)	Ins (%)	Error (%)
TI Feb 91	145	65.2	28.1	6.8	8.8	43.6
MIT Development	198	78	18	4	3	25.6

Table 3: Speech recognition results

the TI test set sentences. It also has a perplexity of over 90, which is higher by a factor of at least four compared to what others have used (see, for example, [2]). In the next few months, we intend to incorporate context-dependent modelling to the ATIS domain. We will also replace the word-pair language model with a bigram so as to increase the coverage and lower the perplexity.

SUMMARY

This paper gives a current status of our ATIS development effort. A significant change since the last workshop is that the system can now accept verbal input. We have also rewritten the back-end component, adopting an approach that should promote extension and portability. We have extended our evaluations this time to include both Class A and Class D1 sentences. Our evaluations for both speech and text input were performed on two different data sets, collected at TI and MIT. The system's performance appears to depend strongly on the conditions under which the data were collected.

REFERENCES

[1] Hirschman, L., Seneff, S., Goodine, D., Phillips, M., "Integrating Syntax and Semantics into Spoken Language Understanding," *These Proceedings.*

[2] Paul, D. B., "New Results with the Lincoln Tied-Mixture HMM CSR System," *These Proceedings.*

[3] Polifroni, J., Seneff, S., and Zue, V. W., "Collection of Spontaneous Speech for the ATIS Domain and Comparative Analyses of Data Collected at MIT and TI," *These Proceedings.*

[4] Seneff, S., Hirschman, L., and Zue, V. W., "Interactive Problem Solving and Dialogue in the ATIS Domain," *These Proceedings.*

[5] Zue, V.,Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., and Seneff, S., "Preliminary ATIS Development at MIT," Third DARPA Speech and Natural Language Workshop, Hidden Valley, PA, June 1990.

[6] Zue, V., Glass, J., Goodine, D., Leung, H., McCandless, M., Phillips, M., Polifroni, J., and Seneff, S., "Recent Progress on the VOYAGER system," Third DARPA Speech and Natural Language Workshop, Hidden Valley, PA, June 1990.