CMU's Robust Spoken Language Understanding System

Sunil Issar and Wayne Ward
School Of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890
USA

Abstract
This paper outlines the general strategies followed in developing the CMU (Carnegie Mellon University) speech understanding system. Our system is oriented toward the extraction of information relevant to a task. It uses a flexible frame-based parser. Our system handles phenomena that are natural in spontaneous speech, for example, restarts, repeats and grammatically ill-formed utterances. It maintains a history of the key features of the dialogue. It can resolve elliptical, anaphoric and other indirect references. In this paper, we pay particular attention to how the context is modeled in our system. We will describe how the system handles corrections and queries that exceed its capabilities. We also address the issue of loose vs. tight coupling of speech recognition and natural language processing.

The system has been used to model an Air Travel Information Service (ATIS) Task. In the November 92 DARPA Spoken Language Systems benchmark evaluation, the CMU ATIS system correctly answered 93.5% transcript inputs and 88.9% speech inputs. These were the best numbers reported for the evaluation.

Keywords: Spontaneous Speech, Flexible Parser, Dialogue, Loose vs. Tight Coupling, Corrections

Introduction
Understanding spontaneous speech presents several problems that are not found either in recognizing read speech or in parsing written text. Spontaneous speech often contains ungrammatical constructions, stutters, filled pauses, restarts, repeats, interjections, etc. In addition, users are not familiar with the lexicon and grammar used by the system. It is therefore very difficult for a speech understanding system to achieve good coverage of the lexicon and grammar that subjects might use. In order to achieve better coverage, many systems [1, 2, 3, 5] augment standard parsers with other techniques, which are used if the parse fails. We address these problems by using a flexible frame-based parser, which parses as much of the input as possible. Our system is oriented toward the extraction of information relevant to a task, and seeks to directly optimize the correctness of the extracted information (and therefore the system response). This approach leads both to high accuracy and robustness. We will look at how the system handles some of the phenomena associated with spontaneous speech. Since the user may not be familiar with the limitations of the system, it is only natural that the user will exceed the capabilities of the system. We will describe how the system seeks to detect and respond to these cases.

We have been developing a natural language understanding system Phoenix that understands spontaneous speech. We have implemented a version of this system for the Air Travel Information Service (ATIS) task, which is being used by several ARPA-funded sites to develop and evaluate speech understanding systems for database query tasks. Users are asked to perform a task that requires getting information from an Air Travel database. The only input to the system is by voice. In general, the user will build the solution incrementally, and the system should allow implicit and explicit references to earlier queries and responses. In this paper, we describe how we model context in the ATIS task.

Finally, we also address the issue of loose vs. tight coupling of speech recognition and natural language processing. In a loosely coupled system, the recognition and understanding portions of the system are separated and may use completely different language models (bigram vs rule based types). Intuition and experience in other areas suggests that tightly coupled systems should work better. The more knowledge used and the earlier in the process that it is applied, the better the performance should be. Yet, in the current ARPA Spoken Language Technology program, loosely coupled systems are prevalent. We explore the reasons why this is the case.

System Overview
Our Spoken Language System uses the Sphinx-II speech recognizer which is loosely coupled to a natural language understanding system. The recognizer uses a back-off class bigram language model in decoding the input. The speech recognition system generates the N-best hypotheses. However, only the top scoring hypothesis is passed to the Natural Language understanding system.

The NL understanding system (Phoenix) uses a flexible frame-based parser [7, 9], which is based on semantic frames. Slots in frames represent the basic semantic entities known to the system. The system gains flexibility by processing semantic fragments rather than complete sentences. Semantic fragments correspond to the slots in the
frames. Recursive Transition Networks are used to specify word patterns constituting semantic fragments. The fragment patterns match sub-strings in the sentence. The parser operates by matching the word patterns for slots against the input text. Possible interpretations are pursued in parallel. As slots are recognized, they are added to the frames to which they apply, and may be filled in any order. A beam search is used, and at the end of an utterance the parser selects the best scoring frame as the result. This is the frame that accounts for the maximum number of words in the input sentence. Heuristics are used to select a single best scoring frame if there are multiple candidates.

As described in [9], our flexible parsing strategy can handle some phenomena that are natural in spontaneous speech, for example, restarts, repeats and grammatically ill-formed utterances. In the rest of this section, we will see how the system handles corrections and queries that exceed its capabilities. We will illustrate our mechanisms with sentences from the ATIS training corpus as well as examples used in [2, 4].

**Corrections**

Spontaneous speech contains disfluencies, e.g., restarts, repeats, verbal edits, where the user is trying to change something that was said earlier in the utterance. Acoustic and prosodic analysis can be used [4] to detect some of these corrections. In the simplest form, the user repeats a sequence of words or the edited phrase is marked by cue words, e.g., no or sorry or I mean. We have a very simple mechanism for handling some corrections:

1. A preprocessor removes words that are identical to the previous word in the sentence. However, it does not remove digits or letters. The sentence *How many American airline flights leave Denver on June June tenth* is replaced by the sentence *How many American airline flights leave Denver on June tenth*.

   In addition, the preprocessor replaces certain word sequences with other word sequences. For example, it replaces a.m. p.m. with p.m. The sentence *Show me the flights after three a.m. p.m.* is replaced by the sentence *Show me the flights after three p.m.*.

2. We modify the grammar so that the elements can end with the cue words used for marking corrections. The back-end simply ignores elements that end with these cue words. For example, the sentence *Can you give me information on all the flights from San Francisco no from Pittsburgh to San Francisco on Monday* is parsed as follows:

```
[flight_field_list] [list_spec] CAN YOU GIVE ME 
[info] INFORMATION ON [flight_fields] [flights] 
[all_flights] ALL THE FLIGHTS 
[DEPART_LOC] FROM [depart_loc] [city] 
[cityname] SAN FRANCISCO [correction] NO 
[flight_type] FROM [depart_loc] [city] [cityname] 
[flights] PITTSBURGH TO [arrive_loc] [city] [cityname] 
[flights] SAN FRANCISCO 
[DEPART_DATE_RANGE] [depart_date_range] 
ON [on_date] [date] [day_of_week] MONDAY
```

Let us look at another example. The sentence *Can you give me information on flights from Pittsburgh to Bal-

**Exceeding System Capabilities**

A necessary attribute of a Spoken Language System is the ability to respond to any query in a reasonable way. The system may not be able to answer a query for a number of reasons, for example, the user may inquire about information not in the database. It's also possible that the system may not be able to parse some significant portion of the sentence. In any event, instead of providing a generic response, the system should tell the user why it cannot respond. Our strategy in handling these limitations is simple — parse the sentence as usual and then:

1. If there is something significant about the words that were not parsed, ask the user about them. This may involve a clarification dialogue. In the examples below, the words left out by the parser are marked with a *, and the words that are not in the lexicon used by the parser are marked with a -.

   **User:** I want to go from Pittsburgh to New York and Boston
   **Recognizer:** I want to go from Pittsburgh to New York Boston
   **Parser:** I want to go from Pittsburgh to New York
   **System:** Please specify whether Boston is an arrival, departure or stop city.

   **User:** Now I am on a very strict diet is it possible to get a meal of my choice with a diet
   **Parser:** *NOW I am *ON *A-VERY-STRCT
   **System:** I don't understand the word diet.

   The parser currently flags the words as shown, but the clarification dialogue has not yet been fully implemented.

2. Otherwise, generate an appropriate response. For the system, this query is a normal query, except that it cannot answer it. The system simply informs the user about the problems in the query. For example,

   **User:** What is the weight limit on the bags that I can carry
**System:** Sorry, I can’t answer questions about baggage

**User:** I want to go from Boston to Tokyo

**System:** Sorry, I don’t know about Tokyo

### ATIS Back-end

In order to allow the users to build solutions incrementally, a Spoken Language System should allow implicit and explicit references to earlier queries and responses. As such, the system must keep track of the dialogue and resolve a number of issues, for example:

1. whether the user is referring to information from the immediately preceding utterance or something that happened earlier in the dialogue
2. what features of the dialogue should be saved
3. what features should be retrieved from an earlier part of the dialogue

In this section, we describe how we achieve some of these objectives. The ATIS back-end maintains the context history necessary to carry on the dialogue. The history can be viewed as a set of contexts, where

1. each context contains the objects needed to generate an SQL query.
2. each context must contain either an arrival or a departure location.

We do not keep track of the system responses or tables in our history mechanism. The back-end also resolves elliptical, anaphoric and other indirect references. It builds and executes an SQL query to retrieve information from the database, and formats the results in the appropriate form.

The best scoring frame generated by the parser is passed to the ATIS back-end. The slots in the best scoring frame are then used to build objects that are needed to generate an SQL query. In this process, all dates, times, names, etc. are mapped into a standard form for the routines that build the database query. For example, all times are converted to the military notation and dates are converted to days of the week.

At this stage ellipsis and anaphora are resolved. Resolution of ellipsis and anaphora is relatively simple in this system. The slots in frames are semantic, thus we know the type of object needed for the resolution. For ellipsis, we add the new objects. For anaphora, we simply have to check that an object of that type already exists.

We first address some idiosyncrasies that arise in the ATIS domain. These were determined by examining the training data, and we describe some of them below:

1. If arrival and departure locations are not specified in the current query, but the airline name is specified and is the same as the one in the immediately preceding query, remove the airline name from the current objects. We illustrate the need for this rule with an example. In the first case, the user is asking for the fares on a specific flight, while in the second case the user is asking for fares on all TWA flights listed initially:

   - Boston to New York around ten a m
   - Does C 0 three forty nine serve any meals
   - What are the fares on the Continental flight

   Boston to New York around ten a m
   - Does C O three forty nine serve any meals
   - What are the fares on the TWA flight

2. If arrival location is specified and is the same as the stop location in the immediately preceding query, but the departure location is not specified, reinterpret the query in specific cases. Again, we illustrate with an example. In this example, the user is asking for the stop times and not the arrival times:

   - Pittsburgh to Boston via New York
   - When do the flights arrive in New York

   We next resolve implicit and explicit references. We need to decide whether any earlier context needs to be propagated, and select the appropriate context from the history. We use simple heuristics for each context in the history, some of which are listed below:

1. Are the departure and arrival locations in the current query the same as in the context?
2. If a flight number is specified, but neither an arrival nor a departure city is specified, check whether such a flight exists between the cities.
3. If an airport name is specified as an arrival (departure) location, check whether the airport serves the city in the arrival (departure) location.

   At this stage, we resolve arrival and destination locations of return flights, as in [6]. However, we do not compute dates of return flights as described in [6].

   After an appropriate context has been determined, the objects from this context are merged with the new objects from the current query. Clearly, none of the objects in the context can replace any of the new objects extracted from the current query. However, we again use simple heuristics to decide whether an object from the selected context should not be used in the current query. We describe some of these heuristics:

1. If the user specifies departure and arrival locations but does not specify a flight number and the arrival and departure locations are the same as in the immediately preceding query, do not propagate any objects from the context.
2. If a fare is specified, cheapest and most expensive flags should not be propagated.
3. If the user is asking about cheapest flights, do not propagate fare constraints.
4. If the user is asking about flights on a specific airline, do not propagate earliest, latest and flight numbers.
5. If the user specifies an arrival time and it is too close to the departure time specified in the context, do not propagate the departure time.

   Each frame has an associated function. After the context has been merged with the objects extracted from the current frame, the frame function is executed. This function takes the action appropriate for the frame. It builds a database query (if appropriate) from objects, sends it to SYBASE (the Database Management System we use) and displays output to the user.

   Finally, the back-end decides whether to update the history. This is a relatively simple decision. If the query had a non null response and there is either an arrival or de-
parture location, the history is updated:
1. It deletes the context from the history that has the same arrival and departure locations as the current objects.
2. It inserts the current objects as a new context in the history.

Interaction Between Recognition and Understanding

In designing a robust system, the interaction between the speech recognition and language understanding components represents a major architectural decision. In a loosely coupled system, the recognition and understanding portions of the system are separated and may use completely different language models. In tightly coupled systems, the language understanding process is integrated with the speech recognition. In the current ARPA Spoken Language Technology program, loosely coupled systems dominate. This is largely due to robustness requirements. Spontaneous speech is both acoustically and grammatically challenging to recognize. Grammatically, it contains mid-utterance corrections and verbal edits, out-of-vocabulary words, metalinguistic comments, dysfluencies, ungrammatical constructions and partial utterances. While traditional Finite State and Context Free Grammar based recognizers have been used successfully for decoding read speech, the standard implementations are less successful for spontaneous speech. They are less robust than stochastic language model recognizers to the disfluent, ungrammatical and verbally "corrected" utterances encountered in spontaneous speech. These difficulties are primarily caused by the challenge of generating grammatical rules that cover commonly occurring spontaneous phenomena as described above. It is very difficult to generate rules that provide good coverage of the word sequences people produce when speaking spontaneously. After an error is made, it is difficult for the system to recover because it is so constrained by the grammar.

Stochastic language models (n-grams or trigrams) are more robust to unseen word sequences since they can be smoothed and their scope is short enough to "get back on track" after an error. When smoothed, these language models assign a non-zero probability to any sequence of words from the lexicon. While they provide robust decoding, stochastic language models may provide a poor match with the Natural Language understanding portions of a system. They do not enforce applicable syntactic, semantic and pragmatic constraints, and often produce word strings that can't be parsed by standard NL parsers. In order to interpret recognition output produced using a stochastic language model, we have found it necessary to use flexible parsing strategies [7, 8, 9]. While we continue to experiment with integrated systems, we are currently using a loosely coupled system because of its combination of robustness, simplicity and efficiency.

Results

ATIS is the task used by ARPA for common (across sites) evaluation of Spoken Language Systems. Test data is gathered using a Wizard-of-Oz paradigm. For the evaluation, whole sessions are processed as dialogs. Utterances are classified as class A (context independent), class D (context dependent) or class X (unanswerable). The systems do not know the classification, and have to decide whether the context should be propagated. Utterances are scored correct if the answer output by the system matches the reference answer for the utterance. The reference answer is the database output, not a word string. The SLS evaluation has two parts, transcripts of the utterances are processed by the NL portion of the system, and then the speech input is processed by the entire system. Processing transcripts shows the NL coverage of the system and gives a baseline measure of how well it would do if recognition were perfect. Processing starting with the speech input then shows how much performance is lost due to recognition errors.

The most recent evaluation was in November 1992. This test set contains 1001 utterances, 427 class A, 247 class D and the remainder class X. In this evaluation, the CMU ATIS system correctly answered 93.5% transcript inputs and 88.9% speech inputs. These were the best numbers reported for the evaluation.

References