A Speech Interface for Forms on WWW

Sunil Issar
Department of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213 USA
E-mail: si@cs.cmu.edu

ABSTRACT

There is a wide variety of forms that a user encounters on the world wide web (WWW). In this paper, we describe the design of a speech interface that can be used over the web to fill forms. This presents several problems, for example, communicating with speech recognizer, parsing one or more forms embedded in text, generating appropriate language models and dictionary entries, and presenting appropriate information (responses and queries) to the user.

Many database and non-database retrieval tasks can be viewed as form-filling tasks. Goddeau [2] also describes a form-based dialogue manager for spoken language understanding tasks. This tends to support our belief that a speech interface for forms is an important first step in the design of distributed spoken language systems, which can assist the user in problem solving activities.

1. INTRODUCTION

Spoken language understanding systems have mainly focused on specific tasks ([1], [7]), for example, airline travel. It is difficult to build spoken language understanding systems because it requires integrating speech recognizer and natural language understanding system. The application developer also needs to specify a dialogue manager that manages interaction with the user. The application development is further complicated by the shortcomings in these underlying technologies. Some problems arise because of errors in speech recognition output and complexities of parsing spontaneous speech, which may not be fluent at sentence level. In addition, many human computer interaction issues need to be addressed because the spoken language understanding system may have its own model of interaction (for example, when to speak) and the user is not aware of system limitations. We also need to systemize the development of dialog manager since present approaches [2] are not very portable.

We are trying to develop a spoken language toolkit, which will simplify the task of developing spoken language applications by automating routine tasks. The developer would specify information at the task (for example, flight information task) level:

- Required (depart location) and optional (arrival time) fields
- Relative importance (From, To, Date, Time)
- Query: What time would you be leaving?
- Algorithm for canonical representation (7:00 PM –> 1900 hours)
- Database Interface (Depart Time 1900 –> flight.departure_time = 1900)
- Action: What do you do when the form is filled or when the user submits the form?

The developer need not know how to build language models, parse user input or tune the speech recognizer. The toolkit would also provide a uniform user interface for different tasks, and avoid reinventing the wheel in different contexts. The dialogue manager would be based on speech-enabled forms.

A form is a collection of text fields, check boxes (yes/no answers), radio buttons (select one or more options) and other information items. HTML provides a standard specification for these forms. Commercial browsers (for example, Netscape, Internet Explorer) can parse and display them. Many database retrieval (for example, ATIS) and non-database (for example, Secretarial Assistant) tasks can be viewed as form filling tasks. The user fills the required fields and one or more optional fields in any order and can then submit the form. The system prompts the user for necessary information, fills in defaults for unfilled fields based on current state, and provides an appropriate response when the form is submitted. This interaction provides a simple model for mixed-initiative dialogue, which can be used to help users. In addition, the user responses may be terse, because the user is filling in specific information. Hence, there may be less disfluencies and hesitation. This may avoid some problems associated with spontaneous speech tasks. We can also use very specific language models to improve speech recognition performance. This paradigm provides a systematic way of extracting information and interacting with the user. We believe speech-enabled forms provide a first step towards a simple dialogue model that can be used in a variety of spoken language understanding tasks.

Speech research is data driven: we need lots of data to train acoustic and language model in a new system. Sphinx interface [5] to the TRAINS system provides an example of using language model trained on data from a different domain (ATIS). However, this approach degrades the performance of recognition system. We are building shared libraries of common components (for example, What, When, Where, Why, Who, How, Date, Time, Location) that can be used in different domains. This also avoids repeated effort in different tasks. In particular, it will provide a collection of language models and phoenix grammars that can be used in the form-filling task.

In this paper, we will first describe the system architecture, then look at the current system that can be accessed over the WWW, and finally describe future work aimed at improving the performance of the system.

2. SYSTEM ARCHITECTURE

Figure 1 shows the overall structure of the system. This system is being designed to run many different applications, which include a form filling application.

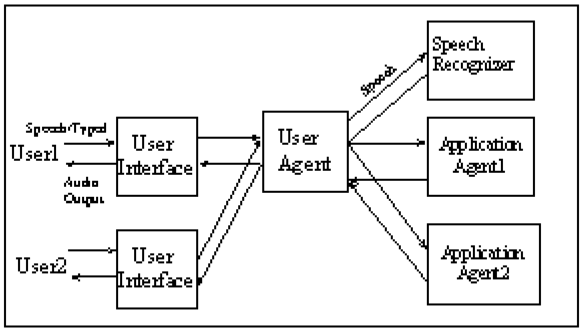

Figure 1: SLS Components

We use a Java applet for user interface. This allows the user interface to be platform-independent. It is loaded in the browser at run-time. The applet connects to a user agent. It sends user interface events, for example, start recording, and fix decoder hypothesis to the agent. It receives various messages (for example, URL to fetch from WWW, recognition output, help messages) that need to be displayed to the user. We use a plugin for audio input and output, since the applet cannot access the audio device. The applet communicates with the plugin (for example, connect to user agent). The user can also replay the last utterance and fix recognition errors.

The plugin provides audio input and output capability, and can also do front end processing (cepstral computation, adpcm compression). It can send speech data to the user agent in 1 of three ways: raw samples (256kb/sec), adpcm (64kb/sec) or cepstral coefficients.
It also receives audio data (system response or query) and plays it.

The user agent mediates between the user (interface), decoder and various applications. It sends speech data to the recognizer, speech output, typed input, or corrections in speech output made by user to the application, and sends application response and decoder output to the user interface (applet). It also uses pre-recorded responses or text-to-speech sythesizer (TTS) to generate audio output.

We use the Sphinx-II speech recognizer [4], which is a continuous–speech, speaker independent system. The user can either use a microphone or a telephone for speech input. We use the CSR acoustic models for microphone input, and a separate set of models for telephone input. The recognizer can switch between pre-defined language models and can also load dynamically generated language models. The speech recognizer can generate N-best hypothesis as well as multiple hypotheses for a given region. However, only the top-scoring hypothesis is passed to the application.

The application parses the user query using the Phoenix parser [8]. This is a flexible frame-based parser, which tries to maximize the amount of parsed input. It uses a semantic grammar, which is based on semantic entities known to the system (for example, date, time). This approach is oriented towards extracting information that is relevant to the task.

The current system can be run on one machine or on many different platforms (for example, the decoder can be running on an alpha station, while an application is running on a PC). This modular approach allows the system to be easily configured in different ways (for example, using microphone or telephone for speech input or connecting to a specific application). It also allows the applications to interact, and thus avoid the need to prompt the user for shared information that is already available elsewhere. Of course, this necessitates a common ontology.

3. TASK INDEPENDENT COMPONENTS

Speech research is data driven: we need lots of data to train acoustic and language model in a new system. In addition, we need data to generate Phoenix grammars. As we experiment with 10,000 spoken language applications [7], we need to build libraries of common components. This would avoid duplicating the effort, ensure that applications have similar performance on common components, and more importantly ensure that important information is not missing from the training data; For example, it is possible that only some *months* occur in the training data.

We built word pair language models for common components (what, when, time, date, day of week). We also extracted some Phoenix grammars from the ATIS systems. The user can specify one of these language models when filling the form.

4. SPEECH ENABLED FORMS

The current system runs under Netscape 3.0+ and can be accessed over the WWW. We can run several spoken language applications using this system:

- Any Form on the WWW
- ATIS
- SCS Directory

However, we will only describe interface for filling a form:

1. The user loads any document by specifying its URL, and can also follow links.

2. The user clicks the *Ready* button when he wants to fill a form. The system uses Netscape Liveconnect to access the form fields, connects to a user agent and sends the form information.
3. The system then waits for the user to speak the field (text, check boxes, radio button, drop down lists) values, interprets the speech recognizer output and enters it in the appropriate field. The user can also type or use the mouse to fill some of the field values.
4. The system can also fill in (appropriate defaults) or update other fields based on the last value that was specified. It can also prompt the user for required fields that do not have default values. It automatically scrolls up or down based on the field that was specified.
5. The user can submit the form, in which case the system simulates the submit button on the form.

There are several problems that arise in processing unseen forms on the WWW:

- A document can have several frames. In addition, there may be several forms in a frame.
- Information about a form field is not easily accessible; for example, it can be an image.

Speech-enabled forms avoid these problems by encoding this information in the name field (name = Prompt__LanguageModel). We display drop-down lists in the help frame. In addition, information in the form is not represented in canonical way. For example, the form may contain *72 mb* as one of the options. We need to translate it to seventy-two megabytes before generating a language model. If this seems easy, consider the option *STB ViRGE[TM], 4MB, 3D 64-bit Graphics*: What's the canonical representation? What is the user going to say? Is it simpler to use a mouse to select this option? Should the system transform it to a different representation, for example, yes/no question? We also need to convert the decoder output from the canonical representation to the representation used in the form.

5. OTHER SYSTEMS

This system presents a model for designing distributed spoken language systems that can be used with commercial browsers. Unlike earlier speech enabled browsers ([6]), it works with commercial browsers (for example, Netscape) without any modification.

The current system is similar in many ways to the MIT [2] system. The main differences are as follows:

- Our system has no built in knowledge of the form. The user can specify any form on the WWW.
- We have a built in library of language models. The user can choose one of them for a particular field. The user need not specify a language model in which case the system uses a default (general) language model. This provides a way of using specific language models. We can also define speech-enabled forms that encode this information. We also dynamically generate language model for some contexts (for example, drop down lists, radio buttons). We are trying to generalise this mechanism by using heuristics to identify appropriate language model for a form field.

6. FUTURE WORK

The speech interface to web forms avoids some of the complexities encountered in designing spoken language systems, for example, it needs very limited context information ([3])). Earlier systems (ATIS systems, Galaxy [1] system) mainly used rule based approaches to model dialog. The form-filling model presents a more structured approach to the problem.

The current commercial spoken language applications (for example, AT&T's operator assistance) and our experience with the ATIS systems demonstrate the feasibility of developing spoken language understanding systems for limited vocabulary tasks. We are trying to use dialogue state information to identify more specific language models based on what the user is likely to do. This also restricts the vocabulary in any dialogue state.

The system described in this paper is an initial model. We are working on several problems aimed at improving performance of this system:

- Task-specific language models:
  - ♦ What's the best language model to use when you have sparse task specific data?
  - ♦ How to use dialogue state information to select a language model
- Improve user interface
  How do we minimize the information displayed or at least highlight the important information?
- Tighter Coupling between application and recognizer
  Can application-specific knowledge help to fix certain recognition errors?
  REF: GO FROM CHICAGO TO **TOLEDO**
  HYP: GO FROM CHICAGO TO *TO LEAVE AT*
  Ringger [5] uses a post processor to address this problem. Can we use alternate hypothesis in the error region (as determined by parser) or can we search this region again using a more specific language model.

These changes will also help in the development of a spoken language toolkit, which will be used for rapid prototyping of spoken language understanding systems.

7. ACKNOWLEDGEMENTS

We would like to thank Raj Reddy for his support and encouragement, Alex Rudnicky for many discussions and comments, and Ravi Mosur for help with decoder issues. This research was sponsored by the Department of the Navy, Naval Research Laboratory under Grant No. 00014-93-1-2005. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

8. REFERENCES

[1] David Goddeau, et. al., "*GALAXY: A Human Language Interface to On-line Travel Information*", Proceedings of ICSLP 94, pp 707-710, Yokahama, September 94

[2] David Goddeau, et. al., "*A form-based dialogue manager for spoken language applications*", Proceedings of ICSLP 96, Philadelphia, October 96

[3] Sunil Issar and Wayne Ward, "*CMU's Robust Spoken Language Understanding System*", Proceedings of Eurospeech 93, September 93

[4] Mosur K. Ravishankar, "*Efficient algorithms for speech recognition*", PhD., Carnegie Mellon University, Pittsburgh PA, May 1996

[5] Eric K. Ringger and James F. Allen. "*Error Correction via a Post-Processor for Continuous Speech Recognition.*" In Proceedings of the 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'96). Atlanta, GA. May 1996

[6] Alex Rudnicky, et. al., "*Speechwear: A mobile speech system*", Proceedings of ICSLP 96, Philadelphia, October 96

[7] Stephen Sutton, David Novick, Ronald Cole, et. al., "*Building 10,000 Spoken Dialogue Systems*", Proceedings of ICSLP 96, Philadelphia, October 96

[8] Wayne Ward, "*Understanding Spontaneous Speech: The Phoenix System*", Proceedings of ICASSP 91, pp 365-367, May 1991