

## Spoken language recognition in an office management domain

Alexander I. Rudnicky, Jean-Michel Lunati, Alexander M. Franz

Carnegie Mellon University, Pittsburgh, PA 15213

### Abstract

Fast and accurate speech recognition systems bring with them the possibility of designing flexible voice driven applications. In this paper, we highlight the needs related to a voice interface and describe the implementation of a general-purpose spoken language interface, the Carnegie Mellon Spoken Language Shell (CM-SLS). CM-SLS provides voice interface services to different applications running on the same computer. CM-SLS was used to build the Office Manager, a collection of applications that includes an appointment calendar, a personal database, voice mail and a calculator. The performance of several system components is described.

### Introduction

The intrinsic properties of speech communication (e.g., the presence of malformed utterances) and the characteristics of current recognition technology (inaccurate recognition) pose special problems for the design of a speech interface. We are interested in understanding these problems and in identifying an interface structure that allows speech to be a useful form of computer input. Ultimately, our goal is to understand how to turn speech into a conventional input modality, well integrated into a multi-modal interface that includes keyboard and mouse. To fully exploit the advantages of spoken communication, a spoken language system must afford the user the following forms of flexibility: natural production, natural language, and a natural flow of interaction. The Carnegie Mellon Spoken Language Shell (CM-SLS) attempts to provide such flexibility through the use of speaker-independent continuous-speech recognition, natural language processing, as well as rudimentary "conversational skill" heuristics.

### The Spoken Language Shell

A good interface design embodies a clear functional decomposition which in turn simplifies system implementation and allows for independent development of different components. The CM-SLS design has proven useful, allowing us to implement several different recognition systems while maintaining modularity. The design decomposes a recognition system into functionally independent units, each corresponding to a necessary function in the speech interface. Note that we have not created novel elements. Each of these functions are implicit in all existing recognition systems but typically are not explicitly identified or recognized as separable components of the interface. The present decomposition provides an explicit separation of these functions, simplifying the exploration of issues that correspond to each component. Figure 1 shows the functional components of the spoken language interface: the Attention Manager, the Recognition Engine, the Confirmation Manager and the Task Manager. The following sections provide more detailed descriptions.

#### Attention Manager (AM)

Humans are remarkably adept at attending to speech in their environment. Computer systems are remarkable in the degree to which they lack this ability.

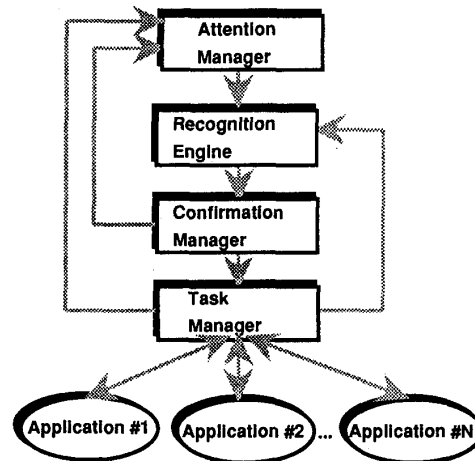


Figure 1: A modular decomposition of the Spoken Language Shell

Spoken language systems need to approximate this ability in order to relieve users of the burden of monitoring the input process. This is the role of the Attention manager.

In our system, the signal processing component produces a continuous stream of coded speech. The AM segments utterance-sized units from this stream and routes these utterances to the recognition engine. The AM implements a range of strategies. At one extreme, the user explicitly controls the signal acquisition process by indicating to the system both the start and the end of an utterance (Push to Talk and Push and Hold modes). At a more complex level of function, the system determines one or both of these points through automatic end-point detection (Push to Start and Continuous Listening modes). We have found that individual preferences vary widely, though more naive users appear most comfortable with the Push and Hold mode. Ideally, the AM should be capable of determining not only the bounds of a true utterance but also know to reject unintentional utterances (and noise) and be able to determine whether the user is actually addressing the computer (as opposed to another agent in the environment).

#### The Recognition Engine (RE)

The Recognition Engine decodes the input utterance into an ASCII string. Recognition imposes a high computational load and it is often impractical to have this process reside on a computer on which several applications (themselves potentially requiring substantial resources) are active. We therefore run the RE of a separate computer. In its present implementation, the RE functions as a dedicated server and allows multiple clients to share the same recognition facilities. Ideally, the recognition engine would be implemented

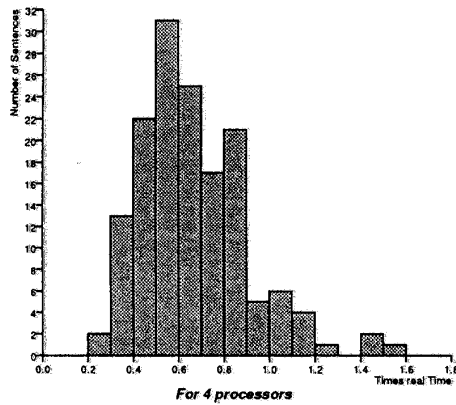


Figure 2: Histogram of system response time for a four-processor recognition engine (running on an Apollo 10040), calculated over 150 utterances.

as a specialized co-processor within the computer.

The RE maintains separate knowledge bases for each application. Control signals communicated by the Attention Manager (which obtains this information from the Task Manager) allow the Recognition Engine to select the correct knowledge base for each utterance. The RE does not maintain any context information of its own, treating each utterance as a separate event. This does not preclude the use of contextual constraint provided by a particular application, and some do so through knowledge-base switching.

A critical attribute of a recognition engine is its ability to decode speech in real-time. Real-time response (or rather response that is within a 200-300 msec delay of the end of an utterance) maintains the rhythm of interaction. Slower response times force users into devoting resources to monitoring system availability instead of concentrating on the task at hand [8]. Figure 2 displays a histogram of "times real time" performance for a four-processor parallel implementation of the Sphinx recognizer [5, 6], calculated over a standard set of 150 Resource Management utterances, using a perplexity 20 grammar. Only the search time component is shown. Using the conventional method (calculating a mean), we might characterize system response as "better than real time", since the mean search time is 0.68 times real time. However, this would be misleading, as the system actually responds slower than real time 15% of the time. From the user's point of view, the recognizer introduces a delay at least fifteen percent of the time (in addition to any delays in the application). Since delay has a clear impact on what users do [1, 9, 7], we believe that a "percent real-time" measure is more relevant to characterizing spoken language system performance than a simple mean response time. The system in Figure 2 is therefore a 85% real-time system.

### Confirmation Manager (CM)

The errorful nature of speech recognition compels the introduction of an additional component not typically found in other interface technologies, the Confirmation Manager (CM). The CM allows the user to intercept or edit a recognition before it is acted upon by the application. In terms of human communication, the CM performs the error repair necessitated by breakdowns in the communication channel (such as might be caused by a noisy telephone line or a loud interruption). It does not concern itself with the consequences of errors due to some misunderstanding on the part of the user (although it does offer an opportunity for the immediate undoing of a just-spoken utterance).

Minimally, the system can pass all utterances through without intervention, though at a cost in throughput [2]. The system can also require the user to provide some acknowledgment (vocal or manual) of the correctness of an input, though again at a cost. The Confirmation Manager additionally allows for editing the input (either by mouse and voice or by keyboard) and for the

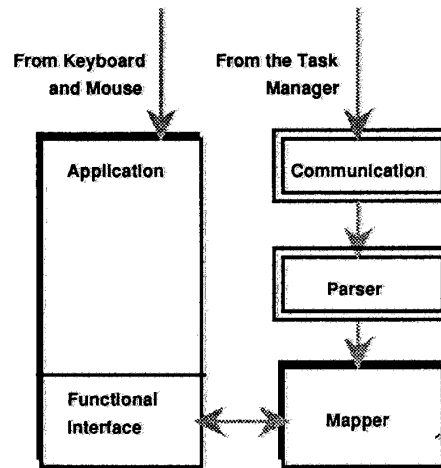


Figure 3: Components of the application interface. The double-framed boxes are black boxes provided to the application programmer. The other components are application-specific.

generation of "undo" signals for the benefit of the application.

The general problem of dealing with input errors actually requires solutions at different levels of a spoken language system. Simple editing of the input, either as provided by the current CM or implemented as a voice editor, is effective only in situations where inputs consist of strings of otherwise unrelated items. This is the case for digit string entry, since there is no relationship among the items that can be exploited. A dictation machine that relies only on a general statistical model of language is another example. In contrast to this are error recovery techniques that exploit the semantic structure (gained either from partial understanding of an utterance or through use of task context) to partially correct an input, then query the user for confirmation or additional information.

### Task Manager (TM)

Speech recognition systems are often built as monolithic processes. While this approach is adequate for a computer that runs a single speech application it is inefficient on a multi-tasking computer that can support a variety of speech-enabled applications. In the latter case it becomes more efficient to centralize speech processing resources and to allocate them dynamically to individual applications. The purpose of the Task Manager is to supervise, in the context of multiple voice-addressable applications, the assignment of the speech channel to the proper application. In our implementation, the actual services performed by the Task Manager also include the maintenance of context information and the communication of this information to the Recognition Engine, where it controls the selection of recognition knowledge bases.

The Task Manager performs a control function comparable to that of the window manager in a window-oriented system. In our design, the voice-capable computer allows the possibility of two parallel input channels, vocal and manual, allowing the use to type or mouse to one application (e.g., an editor) while speaking to another (e.g., a calculator). Ideally a single manager would perform this function, though in our current implementation these are handled separately.

## Applications

An important goal for the design of a general speech interface is to minimize the changes necessary to voice-enable an application. At the same time, the design must enforce a common approach to insure coherence between applications. Figure 3 shows the components of a voice-enabled application in CM-SLS.

The availability of spoken language disposes the user to express requests in terms closer to goals and similar abstractions, instead of sequences of explicit commands, essentially what the advantage of a spoken language interface should be: freeing the user from the need to explicitly specify command sequences and hiding this specification process within the natural language component of the system. The application interface incorporates a frame-based parser, taken from [10], that provides sophisticated natural language processing capabilities. Frames produced by the parser are translated by the Mapper into invocations of methods made available by the application.

Situations arise in which a user request is either underspecified or contains ambiguities. Some of these can be dealt with by using parser mechanisms such as anaphora resolution. Others require the user to further specify their intention. To handle such cases, applications that need this facility can engage the user in a clarification dialog. Currently, the clarification procedure handles cases of ambiguity by informing the user of the situation then asking them to interactively resolve the ambiguity (by choosing one of several alternatives). More complex interactions are possible within this framework, though we have not as yet had the need to consider them. We anticipate the possibility that clarification may need to be provided as an independent service and not embedded in each individual application.

## Human factors in system design

In analyzing the requirements for a spoken language system from a human factors perspective, we applied four principles of human-computer interaction:

**Coherence across applications** Different applications must react similarly to requests that are similar in content and to always react to certain standard inputs (such as requests for help). Doing so allows the user to maintain as much as possible a single style of (spoken) interaction. We address this problem by sharing linguistic components between applications.

**Conciseness inside an application** An application should allow users to express requests in simple economical forms. Providing natural language processing capabilities is one aspect of this. Another aspect is allowing the user to use a variety of expressions, including minimal telegraphic forms. A case-frame based parser allows for such flexibility.

**A meaningful and appropriate system of feedback** The user must be able to easily maintain an accurate model of system state. An explicit indication that the recognizer is available is an example, as is providing a read-out of the recognition result. The ability to respond in real-time underlies the effectiveness of feedback.

**A natural structuring of activities** The system should be able to guide the user into acceptable modes of interaction or to otherwise anticipate how the user will approach it. Developing a language that is suited to the task is one aspect, while incorporating clarification dialogues and hints is another.

## The Office Manager Domain (OM)

The Spoken Language Shell has been used to implement the Office Manager, a system that provides voice access to several common office applications. The Office Manager (OM) domain is interesting for the following reasons: It provides a range of interaction requirements, from tight-loop (e.g., calculation) to open-ended (e.g., database retrieval); it supports meaningful problem-solving activity (e.g., scheduling, information search); and since the applications are directly usable in real-life settings, the activity of actual users can be studied

	conference N=111	department N=664
exact match	76.6	71.5
fuzzy match (good)	8.1	4.5
fuzzy match (poor)	9.9	15.7
no match	5.4	8.3

Table 1: Percentage of surnames from two sources matched to a database of 69,669 words (about 50,000 of which are surnames). N is the number of surnames in the sample.

over extended periods.

OM includes a personal database, an appointment calendar, a mailer interface and a calculator. OM itself understands a 36 word vocabulary and provides control functions, such as starting up and switching between applications, help invocation, etc. The current implementation of the system includes a database of names and addresses for 172 conference participants. It is used by the Voice Mail and Personal Information Database components of OM. Database customizing tools are provided, allowing the non-expert user to create or modify entries in existing databases and to create new databases. Changes to a database result in automatic updates to the recognition knowledge base, allowing users who lack a speech background to easily extend the system.

Our goal was to minimize the amount of speech knowledge required on the part of the user wishing to add or modify database contents. Pronunciation for words added by the user in the course of database modification were located in a 69,669 word pronouncing dictionary and used for knowledge base compilation. Table 1 shows lookup success for two sets of surnames, conference attendees and members of an academic department at Carnegie Mellon. Roughly 20%–30% of the names in these populations could not be found in the database (which includes approximately 50,000 of the most frequent surnames in the United States). To improve on this, we implemented a form of the Soundex algorithm [4, pp. 371–372] to allow fuzzy matches within the database. A “good” fuzzy match means that the correct pronunciation was located, though with a slightly different spelling (e.g., RUDNICKY and RUDNICKI). A “poor” fuzzy match indicates that while a substantially correct match was located, the pronunciation would have to be further modified (e.g., RUDNICKY and RUDNICK). To help the user select and edit pronunciations, a speech synthesizer was integrated into the system, allowing candidate pronunciations to be listened to and iteratively modified.

All components of the system, with the exception of the Recognition Engine are implemented on a NeXT computer, using Objective-C and the Nextstep interface. The RE server is implemented in C as a parallel program on a 4-processor Apollo 10040. Other platforms (DEC DS5000 and IBM R6000) are currently in use as RE's.

## Recognition Performance

To understand system performance we evaluated the PID component of the system. The PID includes the most difficult utterances in the OM, since so little constraint can be applied to database retrieval requests. Using a word-pair grammar, a typical database query has a perplexity of 105 (the branching factor for the identifier field is about 300, considering combinations of first and last names, nicknames and affiliations).

The tests described in this section make use of a recognizer knowledge base configured using a set of 2200 general English models [3]. No task specific training was carried out (had it been, recognition accuracy would be higher). The performance data reported here is based on 194 utterances, collected from 3 female and 7 male talkers. A mix of utterances were included in the test set, whose mean perplexity was 42.

Table 2 shows the various metrics that were calculated. Sentence accuracy notes whether the recognizer produced an exact transcript of the input, semantic accuracy notes whether the intended action was performed. For example, if the utterance was *Show me the address for Smith* and the recognition was *Show address for Smith*, the correct action would still be performed by the

	baseline	tuned lexicon
word accuracy	85.4	90.8
sentence accuracy	59.8	74.2
semantic accuracy (FSM)	78.4	89.2
semantic accuracy (frame)	85.9	93.3

Table 2: Recognition system performance evaluation (percent word accuracy for 814 words in 194 utterances).

system (looking up the Smith address), even though recognition is technically incorrect. We give semantic accuracies for two types of parsing grammars, finite state (FSM) and frame-based. Frame-based grammars can tolerate greater distortions in input and appear to be particularly useful for interpreting the output of a speech recognizer, at least for the class of task that we have been examining.

The difference between the tuned and baseline results reflects the contribution of two modifications: (1) adding multiple entries in the lexicon for function words, (2) improving lexicon entries (consisting for the most part of personal names) by replacing monophone models (indicating that the necessary triphones in question did not exist in the 2200 model database) with close approximations from the existing set. In isolation, the latter manipulation produced less improvement (85.4%  $\rightarrow$  86.5%) in word accuracy than the former (85.4%  $\rightarrow$  90.7%). Ideally, it should be possible to automate these manipulations as part of a lexicon-level tuning process and integrate it into knowledge base compilation.

## Conclusion

We have described a number of innovations in the design of spoken language interfaces. We have advanced a particular functional decomposition for the interface that takes into account human factors principles, and have argued that it identifies key areas in which advances are needed. We have briefly described the Office Management task, which we believe to be particularly suited for the study of spoken language interface issues.

Our future work includes the development of techniques for structuring recognition and parsing knowledge bases along "object" lines, to permit individual applications to inherit language characteristics from their environment (the OM), and to encourage the modularization and reusability of language components. The goal is to simplify the process of creating languages for particular applications by providing the developer not only with standard interface components but also with standard language components.

Meaningful study of spoken language interaction requires a system that will be used on a daily basis and whose utility will persist past the initial stages of play and exploration. We believe that the Office Manager is such a system. Systems that do not have this persistence of utility will ultimately have little to tell us about spoken communication with computers.

## Acknowledgments

We would like thank a number of people who have contributed to the work described in this paper, including Kathryn Baker for work on natural language, Paul Arceneaux for work on the PID, and Eric Thayer for work on the front-end. We would also like to thank Keren Everett for suggesting alternative function-word pronunciations, Robert Weide for help in reselecting triphones, and Jeanette Dravk for her work on the pronunciation database.

The research described in this paper was sponsored by the Defense Advanced Research projects Agency (DOD), Arpa Order No. 5167, monitored by SPAWAR under contract N00039-85-C0163. 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 Defense Advanced Research Projects Agency or the US Government.

## References

- [1] GROSSBERG, M., WIESEN, R., AND YNTEMA, D. An experiment on problem solving with delayed computer responses. *IEEE Transactions on Systems, Man and Cybernetics SMC-6*, 3 (March 1976), 219-222.
- [2] HAUPTMANN, A. H., AND RUDNICKY, A. I. A comparison of speech versus typed input. In *Proceedings of the Third Darpa Speech and Natural Language Workshop* (Hidden Valley, June 1990), Morgan Kaufmann, San Mateo, CA, 1990, p. in press.
- [3] HON, H.-W., AND LEE, K.-F. On vocabulary-independent speech modeling. In *Proceedings of the ICASSP* (1990), IEEE, 1990, pp. 725-728.
- [4] KNUTh, D. E. *Sorting and searching*, vol. 3 of *The art of computer programming*. Addison-Wesley, Reading, MA, 1973.
- [5] LEE, K.-F. *Automatic Speech Recognition: The Development of the SPHINX System*. Kluwer Academic Publishers, Boston, 1989.
- [6] LUNATI, J.-M. A parallel implementation of fast beam search. Carnegie Mellon Speech Group research memo, September 1990.
- [7] RUDNICKY, A. System response delay and user strategy selection in a spreadsheet task. CHI'90, invited poster, April 1990.
- [8] RUDNICKY, A. I., AND QUIRIN, J. L. Subjective reaction to system response delay: A pilot study. Carnegie Mellon Speech Group research memo, January 1990.
- [9] RUDNICKY, A. I., SAKAMOTO, M. H., AND POLIFRONI, J. H. Evaluation of spoken language interaction. In *Proceedings of the Second Darpa Speech and Natural Language Workshop* (Cape Cod, October 1989), Morgan Kaufmann, San Mateo, CA, 1989, pp. 150-159.
- [10] WARD, W. H. The CMU air travel information service service: Understanding spontaneous speech. In *Proceedings of the Third Darpa Speech and Natural Language Workshop* (Hidden Valley, 1990), Morgan Kaufmann, San Mateo, CA, 1990, pp. 127-129.