The design of a spoken language interface

Jean-Michel Lunati **Alexander I. Rudnicky**

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Fast and accurate speech recognition systems systems bring with them the possibility of designing effective voice driven applications. Efforts to this date have involved the construction of monolithic systems, necessitating repetition of effort as each new system is implemented. In this paper, we describe an initial implementation of a general spoken language interface, the Carnegie Mellon Spoken Language Shell (CM-SLS) which provides voice interface services to a variable number of applications running on the same computer. We also present a system built using CM-SLS, the Office Manager, which provides the user with voice access to facilities such as an appointment calendar, a personal database, and voice mail.

Speech interfaces need to provide services that are particular to speech, either due to the intrinsic properties of speech or due to the characteristics of current recognition technology. We are interested in identifying these services and in understanding how they should be integrated into the computer interface. Ultimately, our goal is to understand how to make speech be a conventional form of input to a computer, well integrated into a multimodal interface.

A well-designed speech interface must respect four fundamental rules of computer-human interface design:

1. *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.

2. *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.

3. *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.

4. *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 is another.

To facilitate the exploration of these and other issues, we have developed a system that provides a core of spoken language interface services. In this paper, we describe the design of this system and provide motivation for the various design choices that it incorporates.

Interface components

A good interface design embodies a clear functional decomposition which in turn simplifies system implementation and allows for independent development of different components. The particular design we have arrived at has so far proven to be quite useful, in that we have been able to implement straightforwardly a number of different recognition systems with it, while maintaining its modularity.

The design presented here decomposes a recognition system into what we believe are functionally independent units, each corresponding to a necessary function in the speech interface. It should be noted that we have not created novel elements. Each of these functions are implicit in all existing recognition systems but typically have not been explicitly identified or recognized as separable components of the interface. The present decomposition provides an explicit identification of these functions, thereby 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 of each component.

The 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. Spoken language systems need to approximate this ability in order to relieve users of the burden of monitoring system input on their own. The current system isolates this function as a separate module and permits its independent development.

The signal processing component of the spoken language system produces a constant stream of coded speech (in the form of vector codebook values). The Attention Manager segments utterance-sized units from this stream and routes these utterances to the recognizer. The implementation of the Attention Manager can span a range of complexity. At

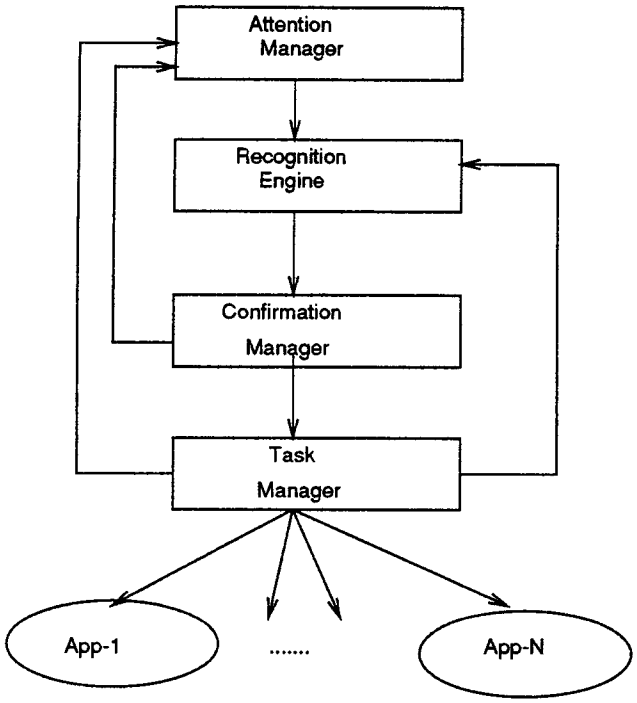

Figure 1: The Spoken language interface

one extreme, the user explicitly controls the signal acquisition process, by indicating to the system the start and the end of an utterance. At a more complex level of function, the system determines these points on its own (through automatic end-point detection). Ideally, the Attention Manager should be capable of determining whether the user is addressing the computer (as opposed to another agent in the environment) using suitable cues in the speech stream or from the environment.

The current version of the AM implements a selection of attention modes, allowing individual users to select the one they prefer or the one best suited to a particular activity. Visually, the AM offers the user a state indicator, in the form of a clickable button, and a sound level indicator, in the form of a VU meter. The following modes are supported:

- **Push to Talk.** The user clicks the talk icon before speaking and clicks again when done speaking. This mode gives the user complete control over the interaction and over delimiting an utterance. The disadvantages include the need to perform two separate acts for each utterance and the use of the mouse, increasing the user's cognitive load with actions that are not application-related (the user may forget to click at the end, or even at the beginning).
- **Push to Start.** The user clicks the talk icon when ready to speak, and the system decides when the utterance has ended, by performing endpoint detection. Only one act needs to be performed, but the user cannot include extended pauses into an utterance. The user must also coordinate the click and the beginning of the utterance.
- **Continuous Listening.** The system uses endpoint detection to delimit each utterance. The advantage is hands-free operation; the disadvantage is lack of control of the extent of an utterance and over the inclusion of extraneous speech.

Recognition Engine (RE)

The Recognition Engine transforms the coded utterance stream into an ASCII string corresponding to the decoding of the input. In our present implementation, the RE functions as a dedicated server and allows multiple clients to share the same recognition facilities. 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. Ideally, the recognition engine would be implemented as a specialized co-processor within the computer.

In its current implementation, the RE maintains separate knowledge bases for each speech application. Control signals communicated by the Attention Manager (which obtains this information from the Task Manager) allows 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, of course, the use of contextual constraint provided by a particular application, based on its individual history. In the current design, the intent is to communicate such information on an utterance by utterance basis.

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) is necessary to maintain the rhythm of interaction. Slower response times force users into devoting resources to monitoring system availability instead of concentrating on the task at hand [4]. We believe that "times real time" is no longer an appropriate metric for characterizing system performance. Figure 2 displays a histogram of "times real time" performance for a four-processor parallel implementation of the SPHINX algorithm, calculated over a standard set of 150 Resource Management utterances, using a perplexity 20 grammar [3]. 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 other delays, due to signal acquisition and processing). Since delays have a clear impact on what users do [1, 5, 6], we believe that a "*percent real-time*" measure is more relevant in characterizing spoken language system performance than a simple mean response time. Our current recognizer is 85 percent real-time for the Resource Management task.

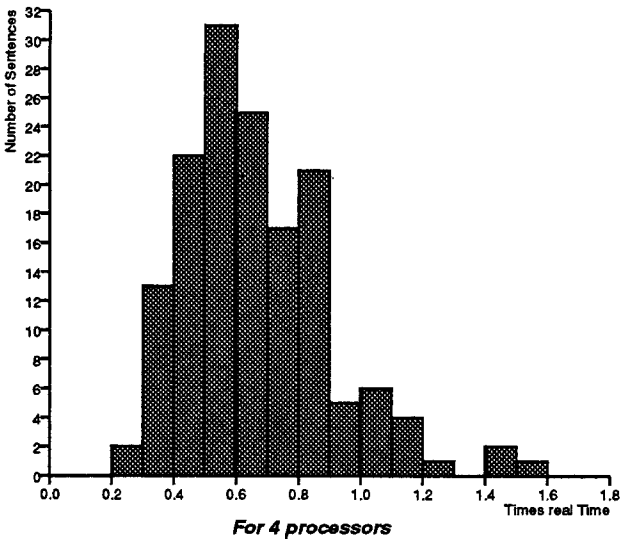

Figure 2: Histogram of system response time for a four-processor recognition system.

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). This component 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. In more complex implementations, the system can allow for editing (either by voice or by keyboard) the input and for the generation of "*undo*" signals for the benefit of the application. The latter facilities are available in our current interface.

A more sophisticated system would be able to, by modeling the interaction and by integrating application-state information, detect utterances that might be, with high probability, incorrectly recognized and on that basis engage the user in a clarification dialog. Currently, we include a clarification dialog component as part of the Mapper within a particular application. In the latter case, the dialog is triggered by inconsistencies in the results of (say) database queries and represents a different class of resolution. On the other hand, if a system is capable of providing high recognition accuracy, can respond rapidly (i.e., in real-time), and is essentially stateless, then simple repetition until correct input is achieved can be a reasonable alternative (see [2]).

Task Manager (TM)

In our previous experience, speech recognition systems have been built as monolithic processes. While this approach is adequate for a computer that runs one or at most a few speech applications it is inefficient on a computer that is meant to support a variety of speech-enabled applications. In this case it becomes more efficient to centralize speech resources (all the more if they are not preemptable once assigned to an application) and to allocate them 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 its communication to the recognition engine.

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 system actually has the possibility of two parallel input channels, vocal and manual. Ideally, a single Manager would perform this function, though in our current implementation, these are handled separately, theoretically allowing for parallel input to the computer, allowing the use to talk to one application while typing to another one.

Applications

Our goal in providing an interface to individual applications was to minimize the changes that need to be made in order to incorporate speech into an application, while enforcing a common approach to our system and accordingly a coherence between applications. The diagram in Figure 3 shows the components of a voice-enabled application.

Each application incorporates a frame-based parser, described elsewhere [7]. The frames produced by the parser are passed to a Mapper which translates each command to the application into suitable method invocations (see next section). Two styles of interface are possible for existing applications, either the Mapper can emulate an existing interface, generating a stream of keyboard and mouse events for each utterance that correspond to the equivalent input for those modalities or it can access functions within the application directly. A previous system [5] was implemented using the former strategy. In the present case, we chose to have the Mapper access application functionality directly. The availability of spoken language disposes the user to express requests in terms of goals and other abstractions, essentially what the advantage of an SL interface should be: freeing the user from the need to explicitly specify command sequences for the application and hiding this specification process within the natural language component of the system. As a result the implementation of an interface between speech and application at this more abstract level can be done quite efficiently.

In some cases, the Mapper encounters situations where a user request is either underspecified or contains am-

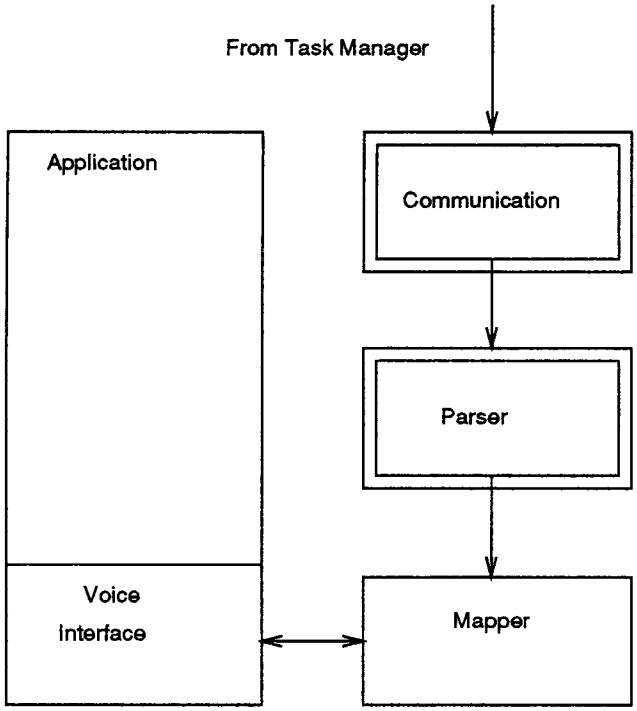

Figure 3: Components of the application interface. The double framed boxes are black boxes for the programmer.

biguities. Some of these situations can be dealt with through the invocation of mechanisms for, e.g., 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.

Some implementation notes

The Carnegie Mellon Spoken Language Shell (CM-SLS) was intentionally designed to have easily modifiable components and to allow the incorporation of different applications with minimum difficulty. Some of the features in our implementation that make this possible include the following:

- The CM-SLS is implemented within an object-oriented paradigm that encourages modularity as well as code reusability, thus making it easy to add new voice driven applications into the system.
- The CM-SLS incorporates a fast speaker-independent continuous-speech multiple knowledge-base recognition system. General English models are used to speed up task development, avoiding task-dependent training.

application	vocabulary	perplexity (estimated)
OM	36	15
Calendar	157	54
PID	367 (111+43)	58
Voice Mail	246 (111+43)	44
Calculator	58	54
total	864	

Table 1: OM task characteristics as of June 1990, PID and Voice Mail have 111 names and 43 nicknames in their vocabulary.

- The CM-SLS offers a high level user interface on a NeXT Machine for efficient end-user access to the applications. We provide an application framework that provides coherence across applications, allows conciseness inside an application, offers an appropriate feedback, and presents a natural structuring of activities allowing a fast and effective access to applications for both casual and expert users.
- The CM-SLS uses external tools to quickly build new applications. Tools include a case frame grammar compiler, a case frame grammar parser, and a semi-automatic speech knowledge base (used by the RE) generator.

The Office Manager (OM)

To demonstrate our approach to speech interface design, we have implemented the Office Manager system, a system which is meant to provide the user with voice access to a number of common computer-based office applications. The Office Management domain has several interesting properties that make it an ideal instrument for exploring issues in spoken language system design. The critical attributes of this task domain are the following:

- It provides a range of interaction requirements, from tight-loop (e.g., calculation) to open-ended (e.g., database retrieval).
- It focuses on a realistic task domain that supports meaningful problem-solving activity (e.g., scheduling, information search).
- It's a domain in which it would be reasonable to expect daily interaction through spoken language. Since the tasks it encompasses are performed regularly, it creates the opportunity to study spoken language interaction on an ongoing basis, under natural conditions.

The Office Manager at present includes the applications listed in Table 1. In addition to the applications themselves, the OM understands a 36 word vocabulary, which is used to execute a variety of control functions, such as creating tasks, switching between them, invoking help, etc. The current (June 1990) implementation of the system in-

cludes a database of addresses for the 111 official participants in the October 1989 Darpa Speech and Natural Language Workshop. This database is used by the Voice Mail and PID (Personal Information Directory) components of OM. Our plan is to make available additional databases to users in our environment (for example, a list of department members) and to pursue the development of tools for user-customizable databases. Customization is of two types: the addition or modification of entries in existing databases, and the creation of new, arbitrary databases by the user. Both forms of customization introduce interesting problems for spoken-language systems: the modification of existing recognition knowledge bases (as might be occasioned by the introduction of a new person name), and the creation (by a presumably naive user) of an access language for a new database.

Conclusion

This paper has described a number of innovations in the design of spoken language interfaces. We have advanced a particular functional decomposition for the interface and have argued that it identifies key areas in which advances are needed. We have proposed what we believe to be a meaningful metric for system response characteristics. We have also 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 the use of 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, Eric Thayer, Bob Weide, Paul Arceneaux, and Alex Franz.

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-C-0163. 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. and Wiesen, R.A. and Yntema, D.B. "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 and typed input. In *Proceedings of the June 1990 Darpa Workshop on Speech and Natural Language*, Morgan Kaufmann, San Mateo, 1990.

3. Lunati, J.-M. A parallel implementation of FBS, Results and Comments (September 1989).

4. A. I. Rudnicky and J. L. Quirin. Subjective reaction to system response delay: A pilot study (January 1990).

5. Rudnicky, A., Sakamoto M. A. and Polifroni, J. Evaluation of spoken language interaction. In *Proceedings of the October Darpa Speech and Natural Language Workshop*, Morgan Kaufmann, San Mateo, 1989, pp. 150-159.

6. Rudnicky, A. System response delay and user strategy selection in a spreadsheet task. CHI'90 invited poster, April, 1990.

7. Ward, W. The CMU Air Travel Information Service: Understanding spontaneous speech. In *Proceedings of the June 1990 Darpa Workshop on Speech and Natural Language*, Morgan Kaufmann, San Mateo, 1990.