THE IBM PERSONAL SPEECH ASSISTANT

Liam Comerford, David Frank, Ponani Gopalakrishnan, Ramesh Gopinath, Jan Sedivy

IBM Watson Research Center
Yorktown Heights, NY 10598
`psg@us.ibm.com`

ABSTRACT

In this paper, we describe technology and experience with an experimental personal information manager, which interacts with the user primarily but not exclusively through speech recognition and synthesis. This device, which controls a client PDA, is known as the Personal Speech Assistant (PSA). The PSA contains complete speech recognition, speech synthesis and dialog management systems. Packaged in a hand-sized enclosure, of size and physical design to mate with the popular Palm III personal digital assistant, the PSA includes its own battery, microphone, speaker, audio input and output amplifiers, processor and memory. The PSA supports speaker-independent English speech recognition using a 500-word vocabulary, and English speech synthesis on an arbitrary vocabulary. We survey the technical issues we encountered in building the hardware and software for this device, and the solutions we implemented, including audio system design, power and space budget, speech recognition in adverse acoustic environments with constrained processing resources, dialog management, appealing applications, and overall system architecture.

1. INTRODUCTION

The thrust of the Personal Speech Assistant project has been to create an early example of a collaborative tool. By this we mean an inanimate instrument capable of collaboration in the same way that humans collaborate with one another. In doing so, we have explored and extended the capabilities of the existing speech recognition, synthesis and understanding technology, and the constellation of related elements (user interface, developer's tools, utilities, hardware requirements, industrial design).

We begin in Section 2 with a discussion of the hardware. In Section 3 we review the spoken language software stack, by which we mean services associated with managing and maintaining a dialog with the user. Then in Section 4 we treat the characteristics and performance of the speech recognition and synthesis systems. Section 5 is a brief discussion of flow of control to applications, and of the applications themselves. Section 6 is a summary.

2. HARDWARE

At the inception of the PSA project, speech recognition and synthesis systems required approximately 100 MIPS to operate with acceptably short latency. The latency may be considered short if a question equivalent to "Was I heard?" doesn't arise in the user's mind. This avoids disruptive efforts to repair the dialog by repeating the command. The processing requirements were met by a single board system designed around NEC MIPS core processors (Vr41xx), operated at 133 MHz. The board, now in its fourth revision, measures 60 mm by 100 mm.

The card supports 8 MB of DRAM and 8MB of on-card flash memory. Memory size requirements were guestimated early in the project, before any portion of the software stack had actually been written. Flash memory, used to store recordings, user interface description files and the PSA software stack, could be increased to advantage.

Early in development, we found that the power consumed by computing and driving audio output through the speaker exceeded the capabilities of most small batteries. The current design uses an MnO_2/Li non-rechargeable battery made for photographic applications. Battery life depends upon usage. With a Palm usage model, the PSA's battery life is measured in days. When operated in continuous audio-out, this falls to approximately 3 hours. The unit, not including the Palm III, weighs 143 grams, of which 17 grams are battery weight.

Two serial ports are provided. One is used as a communication path to a Palm III. The other duplicates the Palm cradle connection to support communication with peripherals and PC synchronization. Audio input is provided via a built in microphone (Lucent DM 1000 model M118HC), microphone amplifier, and an STLC7550 CODEC, sampling input at 11.025 KHz. The audio input filter is essentially flat from approximately 90 Hz to 5 KHz. A 25 mm speaker driven from the CODEC through a power amplifier provides audio output. A headset jack for users desiring "private" audio output is also provided.

The industrial design of the package plays a significant role in its usability. It is made to clamp firmly to a Palm III, so that the impression is created of a single, mechanically robust package. The speaker is mounted in a base-ported enclosure in order to maximize its efficiency in producing voice band signals. The microphone is modified and located to support voice operation at the distance and orientation typical of reading or writing on the Palm. This design was chosen in order to support a physical usage model that allowed normal PDA operation.

Three LEDs are visible on the front face of the PSA. These serve as microphone state (green = listening) indicator, a recording state indicator (green = ready to record) and a user speech volume indicator (yellow = too soft, green = OK, red = clipping). These indicators are provided to make status information immediately available to the user without using precious screen space, or requiring the user to search the screen for it.

3. THE SPOKEN LANGUAGE SOFTWARE STACK

The PSA software stack comprises an operating system (VxWorks version 1.01), a collection of "engines" providing spoken language, communication and other services, and a dialog manager. The role of the dialog manager is primarily construction of the user interface by coordinating the operation of the service engines and utilizing data from the user interface file set.

Two engines are central to the operation of the software stack. These are the speech recognizer and the text-to-speech encoder; these engines are currently available as the IBM Embedded ViaVoice product. Other engines were written specifically to meet project requirements. All project code components are designed to be portable and can be adapted to new hardware or operating systems by modifying a hardware portability layer and recompiling.

3.1. The Embedded Dialog Manager: Philosophy

The Embedded Dialog Manager (EDM) is successful to the extent that it makes the user comfortable with the spoken language interface. The conditions for this are easily recognized from personal experience with conversations that have failed. Human parties to a conversation must feel that they are being paid attention to, that they are understood and elicit a response when they speak, that there is meaning in what is spoken to them, that they can express the same request in several ways, that if understanding fails the other party will cooperate in a mutual effort to restore the conversation, and that the conversation itself may be discussed and its rules changed dynamically. Moreover, the behavior of the conversation must be varied and take into account both recent and long-term conversational history. These properties are supported in the EDM by means of its built-in properties and its collection of user interface data.

Either party, in the course of a conversation, may create utterances in four domains of discourse. The first three are (1) the content of subject of the conversation ("Tom pitched a curve"), (2) the subject of the conversation ("Let's stop talking about baseball"), and (3) the conversational conditions ("Please speak a little louder, I'm having trouble hearing you over the game"). In a conversation between a person and a device, these correspond to addressing an application, addressing the operating system under the application (navigation), and addressing the dialog manager itself. Utterances in the fourth domain occur when two people converse and a third listens; in this case, some portion of the utterances can be intended to influence the listener. In a conversation between a person and a device, these utterances correspond addressing or launching background applications.

3.2. The Embedded Dialog Manager: Design

The user interface (UI) data collection is structured around these four domains. All applications addressed by the spoken language interface software stack provide UI data files for the EDM. UI files can be created and manipulated with any text editor. Specifying behaviors through data files that govern the dialog permits developers to build applications with a conversational interface without any specific knowledge of the APIs of the supporting engines.

A minimal UI file set contains a VOC (vocabulary) file; a typical set of such files will include at least one VOC file, a PMT (prompt) file and a PRF (profile) file. Multiple files of each kind can be provided for each application in order to provide both default and application-state-specific vocabularies, prompts and hardware properties.

VOC files map user utterances into events that may be processed in the event loops of the VOC target. When the EDM accepts a spoken utterance for processing, the decoded string returned by the recognition engine is used as a search key in the set of active VOC files. The search order (dialog manager VOC, target platform VOC, application VOC, background VOC), prevents any application developer from overriding the default vocabularies and damaging default functionality such as inter-application navigation.

PMT (prompt) files provide a set of useful system responses for programmed events such as error conditions or acknowledgments. Well-designed prompts play an important role in creating the illusion of conversation. A PMT file is a list of sets of two elements comprising a prompt key and a prompt string. Prompt strings, which may contain references to environment variables (such as the name of the user), and to sound files, are played after composition. Any process capable of sending a message to the EDM can issue a command to play a prompt, so the same mechanism can be used by non-application functions such as low battery warnings or appointment notifications.

In a minimal prompt file, only one string is specified for each index. If one prompt is spoken to the user several times in a row, the user will perceive this as "mechanical" and grow frustrated. Three conditions arise in which more than one prompt should be specified. If the user needs to be alerted to the same condition several times, a collection of prompts of similar content may be used. A set of prompts of the same information value is called a "rotation." The EDM can also select prompt complexity on the basis of conversational history. Repeated errors cause prompts of successively increased content (taper up) [2]. Growing user experience reduces feature prompts, down to "Ready" or a sound icon (taper down).

Properties such as voicing, power management and button properties also play a role in the user experience. These aspects of the interface are controlled by environment variables. A PRF (profile) file contains a list of variable names and values. When active, these are treated as environment variables and are used to control services, settings, substitutions in prompts, and to pass values between applications. Values in these files may be defaults or application-specific.

Any application can change these values while running, but the defaults will be restored before a new application environment is loaded.

4. SPEECH RECOGNITION AND SYNTHESIS

The Embedded Speech Engine (ESE) is the smallest engine in the ViaVoice product line. The ESE is designed for medium vocabulary (up to 500 active words) speech recognition with finite-state grammars (FSG). The ESE uses the same application development tools as the ViaVoice dication/telephony engine; however, it uses substantially less processor bandwidth and memory. The ESE is highly portable and scalable (small to medium vocabulary) and can run on any suitable 32 bit general-purpose CPU. The speech synthesis module in the ESE is a low-resource version of the one in ViaVoice desktop/telephony.

4.1. ESE Architecture and Toolkit

The ESE uses the abstraction of *services* and *queues*. Each service (or execution unit) reads from an input queue, processes the data, and writes results to an ouput queue. At the heart of the ESE is a *scheduler* that examines the queues and schedules the appropriate service(s) to process data. Typically all the services run synchronously. However, the architecture also supports buffered services. For example, the *ABS* service, described below, works on accumulated cepstrum coefficients.

The recognition part of the ESE is built mainly from three services (*FrontEnd*, *Labeler* and *Decoder*), and three queues (*PcmQueue*, *CepstralQueue*, and *RankQueue*). The *FrontEnd* service reads from *PcmQueue* (which buffers the incoming audio), computes cepstral coefficients and writes results to *CepstralQueue* (which buffers the computed cepstra). The *Labeler* reads cepstra from *CepstralQueue*, computes the ranks (described below) and writes results to *RankQueue*. The *Decoder* reads ranks from *RankQueue*, finds the most likely word string permitted by the FSG and writes it out. The order of service invocation is controlled by a scheduler. The maximum size of each queue depends upon the CPU speed and expected maximum length of an utterance; the queues have to be big enough to buffer sufficient data in case the CPU cannot keep up with the incoming PCM. For the 7/11 digit string recognition task described below, with an acoustic model (AM) of 7000 Gaussians, the CPU requirements of the three services are similar. For larger grammars the *Decoder* takes more CPU. The speed of the *Labeler* is nearly independent of the size of the AM.

The *FrontEnd* service computes standard 13-dim MFCC coefficients from 16 bit PCM, sampled at 11.025 KHz. All computations are in integer arithmetic. The *FrontEnd* also labels each cepstral vector as speech or silence. This decision is based on simple Gaussian mixture models for speech and silence in cepstral space. Due to the latency of speech-silence detector the *FrontEnd* also uses a buffer that holds a few milliseconds of cepstral data. The *FrontEnd* also performs adaptive mean and energy normalization.

The *Labeler* service calculates the delta and the delta-delta coefficients, yielding a 39 dimensional feature vector. The *Labeler* then ranks the Gaussians in the acoustic model (AM) according to their log-likelihoods as computed on the given vector. The *Labeler* represents the acoustic model (typically about 7-10K Gaussians) efficiently in memory, and its algorithm is highly optimized for small footprint and high speed on RISC CPUs. The output from the *Labeler* is the top one hundred ranked HMM states, where each state is modeled by a Gaussian mixture.

The *Decoder* service implements a simple synchronous Viterbi search, based on the rank likelihoods supplied to it [3]. The associated trellis is represented in memory by arrays specifically built for the currently active grammar.

The ESE architecture allows multiple services to run in parallel on the same queue. For example multiple *Labeler*s and *Decoder*s (with different grammars, acoustic models and so on) can run on the same cepstral data simultaneously. Moreover services like *SpeakerID*, *SpeakerVerification*, *CepstralCompression*, and *AcousticBaseformGeneration* can all read from the same *CepstralQueue*; thus the *FrontEnd* service is shared.

An important service that is provided by the recognizer is *AcousticBaseformGeneration* or *ABS*. *ABS* reads in the cepstral vectors that correspond to an uttered word and generates a pronunciation for it. It searches through the space of phone strings for the most likely phone string (as scored by both the recognition acoustic model, and a language model on phone strings) that could have generated the observed cepstral vectors [1]. This service permits on-the-fly creation of acoustic baseforms, for dynamic enlargement of the vocabulary.

The ESE toolkit has two parts: runtime code and offline utilities. The offline utilities enable users to design sets of grammars for applications, and to create the final relocatable binary image. The image can be located in either ROM or RAM, and at any place in memory. The ViaVoice grammar compiler compiles common BNF input source and generates the HMM search graph associated with the input FSG. In addition to the compiled FSG, the binary image includes the acoustic model data, initial mean normalization vector, the speech/silence detection model data and other initialization data.

4.2. ESE Performance

This section describes the application of the ESE to a difficult digit string recognition task in a car. The task is to recognize 7 or 11 digit phone numbers in a car.

A total of 120 hours of speech data were recorded from 300 speakers. Each recording consisted of 100 sentences of read speech from one speaker at one of three speeds: 0/30/60mph. The recorded scripts contained a mix of digit strings, command-and-control strings and general English. Over twenty cars and minivans were used as data collection sites. The data were recorded simultaneously on AKG-

Q400 microphones placed on the driver side seatbelt and sun-visor (or rear-view mirror). The split of data was 80/20/20 hours each at 0/30/60mph respectively.

The training data were augmented with data generated by digitally adding car noise to the 0mph part of the training corpus. The augmented training set (200 hours) was used to build the AM which has about 700 word-internal triphone HMM states, each modeled with a Gaussian mixture. The AM had a total of about 7000 Gaussians in a 39-dimensional feature space.

The test data consisted of recordings of 100 sentences of 7/11 digit strings from 14 speakers (7 male, 7 female) at each of three speeds: 0/30/60mph. The test data were recorded on an AKG-Q400 mounted on the visor.

Operating at a 15 ms frame rate, the ESE requires about 60 MIPS of CPU for this recognition task, which has an unusually broad grammar. Recognizing utterances in a simpler grammar would require less computation. The memory requirements for the digit string task are 387KB ROM and 92KB RAM. Table 4.2 shows the string error rate (SER) of the ESE on this task.

Speed	0mph	30mph	60mph	Avg
SER (%)	2.0	6.0	9.3	5.8

Table 1. String Error Rate (SER) of the ESE on the 7/11 Digit String Recognition Task

5. APPLICATIONS

The EDM was designed to be application neutral: application domain expertise is expected to reside in the application, and conversational expertise is expected to reside in the EDM and the UI data files. The EDM provides its expertise by responding to messages. The external sources of messages are the user, the application platform and the current application. When the EDM captures a user utterance, it sends the audio to the ESE, and prepares to receive back an error or a recognized phrase. On receiving the response from the ESE, if it is not an error, the decoding is found in the active VOC file collection, and its associated event and data are sent to the target application.

By these means, we have speech-enabled the Palm PIM applications (date book, address book, to do list, memo pad), and a simple phrase-to-phrase language translator. The PIM applications were made the owners of their user interface files, and events in these applications were aliased or added to respond to messages from the PSA. Provisions were also added to drive the text-to-speech and PCM playback systems from the application.

The speech-enabled date book functions include navigation in the calendar by day (for instance "yesterday," "today," "last Friday, "next Sunday") and time ("8 am," "5 pm"), readout of appointments ("next appointment," "list appointments"), and creation of new appointments at a specified date and time. New appointments drawn from a short list of place names (for example "auditorium") can be transcribed directly into the calendar as text, or can be created in audio format and stored there as PCM. Future versions will transfer such audio files to the desktop at synchronization time, where they will be decoded by a large vocabulary recognizer, and reinserted as text into the calendar. A similar set of functions are enabled for the other PIM applications. All these applications may also be controlled by the familiar stylus interface, in case the user prefers silent interaction.

The phrase-to-phrase translator recognizes a small vocabulary of complete phrases judged useful to the business traveler (for instance "Good morning," "I am lost," "I'd like a single room," "Please take me to this address"). These are then rendered into one of a selected list of languages, currently French, German, Italian, Japanese and Spanish. The rendering is accomplished either through the text-to-speech system, or by playback of prerecorded utterances of native speakers. The translation is a simple mapping from one known phrase to another.

6. SUMMARY

This paper has described the hardware and software characteristics of the IBM Personal Speech Assistant (PSA), a handheld conversational personal information manager. The PSA is a both platform for experimentation and a demanding proving ground. In addition to the obvious challenge of performing speech recognition in the varied and often highly adverse acoustic environments where handheld devices are expected to operate, we have contended with the constraints imposed by the unit's desired size and weight, the esthetics of creating a device that is pleasing to look at and hold, the creation of an environment that is easily modified and extended by others, and the synthesis of a complete user interface from all these elements of design and technology. Whether we have succeeded must be judged by those who use the result of our labors.

7. ACKNOWLEDGEMENTS

We wish to thank Sabine Deligne, Ellen Eide, Paul Fernhout, Dimitri Kanevsky, Benoit Maison, Harry Printz, Roberto Sicconi, Karl Zeeso, IBM Vimercate (now Celestica Vimercate) and IBM Voice Systems for their contributions.

8. REFERENCES

[1] S. Deligne, B. Maison and R. A. Gopinath. Automatic Generation and Selection of Baseforms for Dynamic Vocabularies *Proc. ICASSP 2001.*

[2] N. Yankelovich. Designing SpeechActs: Issues in Speech User Interfaces *CHI'95 Proceedings.*

[3] L. R. Bahl, P. V. de Souza, P. S. Gopalakrishnan, D. Nahamoo, M. A. Picheny. Robust Methods for Using Context-Dependent Features and Models in a Continuous Speech Recognizer *Proc. ICASSP 1994.*