T. V. Raman
Cambridge Research Lab
Digital Equipment Corp.
Bldg 650, One Kendall Square
Cambridge MA 02139
E-mail: ⟨raman@crl.dec.com⟩
Voice-mail: 1 (617) 692-7637
Fax: 1 (617) 692-6650

Abstract

Screen-readers —computer software that enables a visually impaired user to read the contents of a visual display— have been available for more than a decade. Screen-readers are separate from the user application. Consequently, they have little or no contextual information about the contents of the display. The author has used traditional screen-reading applications for the last five years. The design of the speech-enabling approach described here has been implemented in Emacspeak to overcome many of the shortcomings he has encountered with traditional screen-readers.

The approach used by Emacspeak is very different from that of traditional screen-readers. Screen-readers allow the user to listen to the contents appearing in different parts of the display; but the user is entirely responsible for building a mental model of the visual display in order to interpret what an application is trying to convey. Emacspeak, on the other hand, does not speak the screen. Instead, applications provide both visual and speech feedback, and the speech feedback is designed to be sufficient by itself.

This approach reduces cognitive load on the user and is relevant to providing general spoken access to information. Producing spoken output from *within* the application, rather than speaking the visually displayed information, vastly improves the quality of the spoken feedback. Thus, an application can *display* its results in a visually pleasing manner; the *speech-enabling* component renders the *same* in an aurally pleasing way.

Keywords: Speech Interface, Direct Access, Spoken Feedback, Audio Formatting, Speech as a first-class I/O medium.

Introduction

A screen-reader is a computer application designed to provide spoken feedback to a visually impaired user. Screen-readers have been available since the mid-80's. During the 80's, such applications relied on the character representation of the contents of the screen to produce the spoken feedback. The advent of bitmap displays led to a complete breakdown of this approach, since the contents of the screen were now light and dark pixels. A significant amount of research and development has been carried out to overcome this problem and provide speech-access to the Graphical User Interface (GUI).

The best and perhaps the most complete speech access system to the GUI is Screenreader/2 (ScreenReader For OS/2) developed by Dr. Jim Thatcher at the IBM Watson Research Center [Tha94]. This package provides robust spoken access to applications under the OS2 Presentation Manager and Windows 3.1. Commercial packages for Microsoft Windows 3.1 provide varying levels of spoken access to the GUI. The Mercator project [ME92, WKES94, MW94, Myn94] has focused on providing spoken access to the X-Windows system.

A common feature of traditional DOS-based screen-readers and speech access packages to the GUI is their attempt to convey the contents of the visual display via speech. In fact, a significant amount of the development effort required to design speech-access packages to the GUI has concentrated on building up robust *off-screen models* —a data structure that represents the contents of the GUI's visual display. Construction of such an off-screen model helps screen-readers regain the ground they lost due to the advent of graphical displays. However, the nature of spoken feedback provided does not change.

Shortcomings Of Reading The Screen

Screen-readers have helped open up the world of computing to visually impaired users[1]. However, the spoken interface they provide leaves a lot to be desired.

The primary shortcoming with such interfaces is their

[1] The author has used these for the last five years.

inability to convey the structure present in visually displayed information. Since the screen-reading application has only the contents of the visual display to examine, it conveys little or no contextual information about what is being displayed. Put another way:

> A Screen-reader speaks *what* is on the screen without conveying *why* it is there.

As a consequence, accessing applications that display highly structured output in a visually pleasing manner with screen-readers is cumbersome.

Here is a simple example to illustrate the above statement. A typical calendar display is made up of a table showing the days of the week. This information is visually laid out to allow the eye to quickly *see* what day a particular date of the month falls on. Thus, given the display shown in Fig. 1, it is easy to answer the question "What day is it today?". When this same *display* is

Jan 1995

S	M	T	W	Th	F	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figure 1: A Typical Calendar Application

accessed with a screen-reader, the user *hears* the entire contents of the table 1 spoken aloud. This results in the following set of meaningless utterances:

> pipe pipe 1 pipe 2 pipe 3 pipe 4 pipe 5 pipe 6 pipe 7 pipe pipe ... pipe pipe 29 pipe 30 pipe 31 pipe pipe pipe pipe pipe pipe

Alternatively, the characters under the application cursor can be spoken. In the case of Fig. 1, the listener would hear the system say "one". To answer the question "What day is it today?" the user has to first build a mental representation of the visual display, and then navigate around the screen, examining the contents that appear in the same screen column as the **1** in order to infer the fact that the date is Sunday, January 1, 1995.

Screen-readers for both character-cell and graphical displays suffer from this shortcoming. This is a consequence of trying to *read* the screen instead of providing true spoken feedback. The rest of this paper describes Emacspeak, an interface that treats speech as a first-class output medium. Screen-readers speak the screen contents after the application has displayed its results; Emacspeak integrates spoken feedback into the application itself. This tight integration between the spoken output and the user application enables Emacspeak to provide rich, context-sensitive spoken feedback. As a case in point, when using the calendar application, the user hears the current date as Sunday, January 1, 1995. For related work in integrating speech as a first-class I/O medium into general user applications, see [YLM95].

We conclude this introduction by pointing out that visual layout plays an important role in cuing the reader to information structure. Such visual cues reduce cognitive load by allowing the perceptual system to perceive the inherent structure present in the information, thereby freeing the cognitive system to process the information. Spoken feedback produced from the visual layout proves difficult to understand because many of the structural cues are lost; to make things worse, other structural cues turn into *noise* (the "pipe pipe ..." above is a case in point). This results in the listener having to spend a large number of cognitive cycles in trying to parse the spoken utterance, making understanding the information considerably harder. Speaking the information in an aurally pleasing manner alleviates this burden, leading to better aural comprehension.

A Different Approach

We tightly integrate spoken output with the user application. Such tight integration allows the functions providing spoken feedback direct access to the application context. Thus, in the case of the calendar example shown in Fig. 1, the speech feedback routines can access the runtime environment of the calendar application to find out that the current date is Sunday, January 1, 1995 instead of trying to guess this from the visual presentation of the calendar.

Thus, using speech as a first-class output medium provides *direct access* to the information displayed by an application —traditional screen-readers provide what can at best be described as *indirect* access.

Motivation

Every computer application (big or small) can be characterized as having the following structure:

- Accept user input
- Compute on the data
- Display results of the computation

Human computer interaction focuses on the first and third of these stages. Traditional WIMP interfaces[2] have assumed a purely visual interaction; applications designed for such interfaces naturally optimize their displays to this mode of interaction.

However, visual layout is not optimal for spoken interaction as evinced by the calendar application (See Section). By having the user interface (UI) components

[2] Windows, Icons, Menus, and Pointer

of the application communicate directly with the speech subsystem, Emacspeak produces more *usable* output. Contrast this with the screen-reading paradigm, where spoken output is produced by a program that is *unaware of* and *separate from* the user application.

Implementation

We have motivated the design of Emacspeak with the help of the calendar example. However, Emacspeak is much more than a simple talking calendar; it extends all of GNU Emacs to provide full spoken feedback. The author uses Emacspeak on his Alpha AXP[3] workstation running Digital UNIX and on his laptop running Linux. Emacspeak has been made available on the Internet[4] and is currently being used by an increasing number of Digital's customers.

This paper will not go into implementation details —our purpose is to highlight the novel interface provided by Emacspeak. For the sake of completeness, here is a brief sketch of how the system is implemented.

Emacspeak consists of a core speech module that provides basic speech services to the rest of the system, e.g., functions that speak characters, words and lines. The *advice* facility of Emacs Lisp is used to integrate the speech feedback provided by these functions into Emacs. This facility allows us to specify program fragments that are to be run either before, after, or around any function. Since the user interface level of GNU Emacs is implemented entirely in Emacs Lisp, the functions making up this interface can be *advised* to speak. The primary advantage of this approach is that we have been able to speech-enable all of GNU Emacs —a large system— without modifying a single line of source code from the original Emacs distribution.

We conclude this sketch with an example. Function **next-line** implements movement of the editing cursor to the next line in GNU Emacs. Emacspeak provides the following *advice* to this function:

```
(defadvice next-line (after emacspeak )
  "Speak the line you moved to."
  (when (interactive-p)
    (emacspeak-speak-line )))
```

This advice specifies that if function **next-line** is called *interactively* (As the result of the user pressing a key.) then function **emacspeak-speak-line** should be called **after next-line** has done its work.

The next section Section gives examples of the spoken interaction provided by Emacspeak when performing several day-to-day computing tasks. All of the facilities described are implemented using the model described above.

Examples Of Common Computing Tasks

This section describes the user interface provided by Emacspeak when performing common-place computing tasks like editing and proof-reading, surfing the WWW, reading and replying to electronic mail and Usenet news. This paper description suffers from the natural shortcoming of elucidating in print what is essentially aural. Here are some features of the spoken feedback that are common to the different interaction scenarios:

- Speech output is always interruptible. Actions causing new information to be spoken first interrupt any ongoing output.

- Emacspeak provides a *voice-lock* facility that permits association of syntactic units of text with different voices. This is a powerful method of conveying structure succinctly and was first described in [Ram94]. Audio Formatting is used to aurally set apart different syntactic units, for example, highlight regions of text.

- Emacspeak uses auditory icons [SMG90, BGB88, Gav93, BGP93, JSBG86] —short snippets of sounds (under 0.25–0.5 seconds) to cue common events such as *selecting*, *opening* and *closing* an object. Used consistently throughout the interface, these cues speed up user interaction —an experienced user can often continue to the next task when an aural cue[5] is heard without waiting for the spoken confirmation.

Editing Documents

Emacspeak speaks each character as it is typed. Pressing the space-bar causes the previous word to be spoken. Cursoring through a file speaks each line; speech is interrupted if the cursor is moved while a line is being spoken. This allows the user to efficiently browse files. All of the standard Emacs navigation commands, e.g., move to the next paragraph, skip this S-expression, give appropriate auditory feedback.

Emacs' knowledge of the syntax of what is being edited is used to advantage in enabling sophisticated navigation. For instance, the user can move across *statements* when browsing program source code. When navigating through a file of C code, the user gets relevant spoken feedback that conveys the structure of the program. Different syntactic units are spoken in different voices to increase the band-width of aural communication. In addition, the user can have the *semantics* of a line of source code spoken upon request. Thus, when the

[3] For the first time in five years, I can sit in front of a workstation, rather than in front of a DOS PC functioning as a terminal! UNIX is a trademark of Unix Systems Laboratories.
The following are trademarks of Digital Equipment Corporation: Alpha AXP, DEC, DECstation, DECtalk.

[4] *URL http://www.research.digital.com/CRL*

[5] Emacspeak will still produce the spoken confirmation, but continuing to the next task will interrupt this speech.

editing cursor is on the closing brace that ends a function block, Emacspeak says "brace that closes function" and then speaks the opening line of that function. This provides the listener the same kind of feedback that users of traditional visual interfaces have come to expect.

Spell Checking

Emacspeak provides a fluent aural interface to **ispell**, a powerful interactive spell checker. Here is a brief description of the visual interface provided by the spell checker for those unfamiliar with this system.

Typically, a file opened with Emacs can be spell-checked by invoking **ispell**. Errors are visually highlighted, with a separate window showing a list of possible corrections. The user can type a number to pick a choice from the list of corrections; alternatively, a replacement can be directly typed in.

Using this interface with a traditional screen-reader is painful to say the least[6]. A user of a screen-reader needs to query the position of the cursor to find out the erroneous word, then locate the window of corrections on the screen before continuing.

With Emacspeak, the fact that the list of possible corrections appears in a separate window is completely hidden from the listener. When running the spell checker, Emacspeak speaks the line containing the erroneous text with the incorrect word *aurally* highlighted. Next, the list of possible corrections is spoken; the user can pick a choice at any time. Based on the user action, the spell checker inserts the appropriate correction and continues to the next error.

A similar approach is used to provide aural feedback to the common editing task of interactively replacing a string by another. Emacspeak speaks the line containing the instance of the text being replaced, with the instance that will be replaced *aurally* highlighted. This allows the listener to respond correctly when there are multiple occurrences of the text being replaced within a line. Thus, the task of replacing the first occurrence of **foo** with **bar** while leaving the second instance of **foo** intact in the example

> Change this food, but do not touch this fool.

is trivial; the same task using a screen-reader is much harder.

Electronic Mail

Emacspeak provides a fluent spoken interface to electronic mail. Instead of having to listen to verbose utterances consisting of email headers, the listener hears a succinct summary of the form " *sender name* on *subject*.". Emacspeak also infers the dialogue structure present in electronic mail messages based on standard conventions used to cite the contents of previous messages in a conversation thread. When such dialogue structure is detected, the different parts of the dialogue are spoken using different voice characteristics. Hitting any key while a part of the dialogue is being spoken results in the system skipping to the next portion of the dialogue.

Usenet News

Emacspeak provides a fluent spoken extension to GNUS, the GNU Emacs news-reader. The interface permits the user to browse news using the four arrow keys.

We present the user with a simple metaphor of opening and closing objects. The up and down arrows navigate through objects at the current level; the right arrow opens the current object, while the left arrow closes it.

To begin with, the user opens up Usenet news. The up and down arrows navigate through the list of newsgroups, providing a succinct verbal description of the current group and the number of articles that are unread. Opening a group with the right arrow results in the up and down arrow keys moving through the list of unread articles; again, the article is succinctly summarized using utterances of the form "*Sender* on *topic*, 33 lines.". Opening an article by pressing right arrow speaks it; the listener can move to the next article merely by pressing the down arrow, which will interrupt the reading of the current article, and summarize the next article. The auditory icons described earlier are especially useful when browsing news; the aural cues for opening, closing and selecting objects allow the listener to quickly move to the next task in the interface.

All of the features described in the section on reading email are available when reading news; Emacspeak presents the dialogue structure present in news articles using the voice-lock feature described above.

Surfing The WWW

The WWW presents two interesting challenges to a spoken interface.

- Presence of hypertext links.
- Presence of interactive elements, e.g., fill-out forms consisting of UI elements such as input fields, check boxes and radio buttons.

Emacspeak provides a spoken extension to W3, the excellent Emacs-based WWW browser developed and maintained by William Perry.

Browsing A WWW Page. The listener can browse a WWW page just like any other document. Hyperlinks are spoken in a different voice. The listener can interrupt speech at any time and activate the link that was most recently spoken. The listener can also move the

[6] Believe me, I've done it!

application focus between the various links on a page; jumping to a link results in the anchor text being spoken along with an auditory cue indicating a large movement. Activating a link plays the auditory icon for *opening* an object, retrieves the document, and finally announces the title of the newly opened WWW document.

Interactive WWW Documents. The W3 browser parses a WWW document before displaying it. Emacspeak relies on this internal representation to provide the spoken rendering, rather than examining the visually displayed document. This fits well with the overall design of Emacspeak; it also enables Emacspeak to produce spoken feedback that would be impossible to generate by merely examining the screen.

A typical interaction with a form element consists of:

- Moving system focus to the element.
- Changing the state of the form element, e.g., pressing a button or entering a value.
- Obtaining confirmation from the system about the recently performed action.

We illustrate this with examples of what happens when the user interacts with different form elements that are found on WWW documents.

Text Field
- Emacspeak summarizes the element under the focus with an utterance of the form "text field *field name* set to *value*.". The name of the text field and its value if any are retrieved from the internal representation.
- Pressing enter results in the spoken prompt "Enter value for *field name*.".
- After the value has been input, Emacspeak confirms this with the announcement "text field *field name* set to *value*.".

Check Box
- Emacspeak summarizes the check box with an utterance of the form "Check-box *name* is *checked*.", assuming the box has been previously checked.
- Pressing enter produces a button click.
- Emacspeak says "unchecked check box *name*.".

Radio Button The interaction parallels that described above for check boxes. The utterance uses the phrase "is pressed" to distinguish radio buttons from check boxes.

Navigating The File System

Emacs' **dired** mode, which is used to navigate the file system and perform operations such as moving, copying and deleting files, is extended to provide succinct aural feedback. When navigating through the file listing, the user hears the name of the current file or directory spoken; different *file types* e.g., directories, executables and symbolic links are distinguished by speaking their names in different voices. Opening a file plays the auditory icon for opening an object, and then speaks the name of the file just opened. Marking a file for later processing, deleting a file etc. all produce auditory icons. The auditory icons in this context are very useful because typically, performing an action such as deleting a file when using **dired** affects the current object and moves the focus. Visually, the file marked for deletion is set apart and the focus is moved. Combining the sound of a file being deleted with the speaking of the current object introduces the same level of parallelism in the aural interaction.

When navigating the *dired* buffer for the directory containing this paper, a screen-reader would speak a typical line shown below

> -rw-r–r– 1 raman users 11905 Aug 17 16:04 examples.tex

as " dash rw dash r dash dash r dash dash 1 raman users 11905 Aug 17 16:04 examples.tex", an utterance that is hard to parse and comprehend. In contrast, Emacspeak merely speaks the filename; the listener can repeatedly press the *tab* key to hear the various fields of the file listing. Below, we list the utterances produced by each repeated press of the *tab* key.

Permissions rw r r
Links 1
Owner raman
Group users
Size 11905
Last Modified Aug 17 16:04
Filename examples.tex

Figure 2: Tabbing through a file listing.

Notice that Emacspeak infers the meaning of each field in the file listing. Pressing the *tab* key while a field is being described interrupts speech immediately and moves to the next field.

Conclusion

We conclude with a summary of what we have learnt from the work on Emacspeak. Firstly, the design of Emacspeak as a speech interface as opposed to a system that reads the screen is radically different from what has been attempted in the past. The current implementation has achieved a remarkable level of success in providing fluent speech access to day-to-day computing tasks.

The convoluted interfaces provided by screen-readers proved moderately effective in the case of visually impaired users —there was no other choice and consequently, users had the motivation to learn and use these interfaces. However, general users who wish to use speech as an extra modality to enhance their interaction with the computer are unlikely to put up with such interfaces. The *direct* access provided by the speech-enabling approach is likely to produce more acceptable output and make deploying speech interfaces easier.

Finally, our implementation of Emacspeak has provided the first true speech access interface to UNIX workstations. To date, the only available solution for visually impaired users has been to access these using PC's running screen-readers as a talking terminal. Our work provides a viable alternative to accessing the power of UNIX and the wealth of communication and development tools that are commonplace in this environment.

Acknowledgements

We would like to thank the authors of the various Emacs subsystems such as the WWW browser, email and news readers. Without their work, Emacspeak would have remained a speech interface to a text editor; in itself not a very useful artifact. Special thanks go to Hans Chalupsky, author of the *advice* package, without which the implementation of Emacspeak would have been difficult, if not impossible. I would also like to thank Win Treese for drawing my attention to the power of the *advice* facility and Dave Wecker[7] for goading me into writing Emacspeak.

REFERENCES

[BGB88] W. Buxton, W. Gaver, and S. Bly. The use of nonspeech audio at the interface. *Tutorial Notes, CHI '88.*, 1988.

[BGP93] Meera M. Blattner, Ephraim P. Glinert, and Albert L. Papp. *Sonic Enhancements for 2-D Graphic Displays, and Auditory Displays.* To be published by Addison-Wesley in the Santa Fe Institute Series. IEEE, 1993.

[Gav93] William Gaver. Synthesizing auditory icons. *Proceedings of INTERCHI 1993*, pages 228–235, April 1993.

[JSBG86] K. I. Joy, D. A. Sumikawa, M. M. Blattner, and R. M. Greenberg. Guidelines for the syntactic design of audio cues in computer interfaces. *Nineteenth Annual Hawaii International Conference on System Sciences*, 1986.

[ME92] Elizabeth D. Mynatt and W. Keith Edwards. Mapping GUIs to auditory interfaces. *Proceedings ACM UIST92*, pages 61–70, 1992.

[MW94] E.D. Mynatt and G. Weber. Nonvisual presentation of graphical user interfaces: Contrasting two approaches. *Proceedings of the 1994 ACM Conference on Human Factors in Computing Systems (CHI'94)*, April 1994.

[Myn94] E.D. Mynatt. *Auditory Presentation of Graphical User Interfaces.* Santa Fe. Addison-Wesley: Reading MA.., 1994.

[Ram94] T. V. Raman. *Audio System for Technical Readings.* PhD thesis, Cornell University, May 1994. URL *http://www.research.digital.com/CRL* /personal/raman/raman.html.

[SMG90] D. A. Sumikawa, Blattner M. M., and R. M. Greenberg. Earcons and icons: Their structure and common design principles. *Visual Programming Environments*, 1990.

[Tha94] James Thatcher. Screen reader/2: Access to os/2 and the graphical user interface. *Proc. of The First Annual ACM Conference on Assistive Technologies (ASSETS '94)*, pages 39–47, Nov 1994.

[WKES94] E. D. Mynatt W. K. Edwards and K. Stockton. Providing access to graphical user interfaces - not graphical screens. *Proc. Of The First Annual ACM Conference on Assistive Technologies (ASSETS '94)*, pages 47–54, Nov 1994.

[YLM95] Nicole Yankelovich, Gina Anne Levow, and Matt Marx. Designing speechacts: Issues in speech user interfaces. In *Proceedings of CHI95, Human Factors In Computing Systems*, pages 369–376. Sun Micro Systems, May 1995.

[7] He got tired of listening to my complaints about how inadequate screen-readers were.