Emacspeak –Direct Speech Access

T. V. Raman*
Adobe Systems
E-mail: ⟨raman@adobe.com⟩
Voice-mail: 1 (415) 962-3945

Abstract

Emacspeak is a full-fledged speech output interface to Emacs, and is being used to provide direct speech access to a UNIX workstation. The kind of speech access provided by Emacspeak is qualitatively different from what conventional screen-readers provide —emacspeak makes applications speak— as opposed to speaking the screen.

Emacspeak is the first full-fledged speech output system that will allow someone who cannot see to work directly on a UNIX system (Until now, the only option available to visually impaired users has been to use a talking PC as a terminal.) Emacspeak is built on top of Emacs. Once Emacs is started, the user gets complete spoken feedback.

I currently use Emacspeak at work on my SUN SparcStation and have also used it on a DECALPHA workstation under Digital UNIX while at Digital's CRL[1]. I also use Emacspeak as the only speech output system on my laptop running Linux.

Emacspeak is available on the Internet:

FTP ftp://crl.dec.com/pub/digital/emacspeak/

WWW http://www.research.digital.com/CRL

*The work described in this paper was performed at Digital Equipment Corporation's Cambridge Research Lab.

[1]Emacspeak was developed as a spare-time project while I worked at Digital's Cambridge Research Lab (CRL).

Keywords

Direct Speech Access, Access to UNIX workstations.

1 Introduction

Emacspeak is an Emacs subsystem that allows the user to get feedback using synthesized speech. Traditionally, screen reading programs have allowed a visually impaired user to get feedback using synthesized speech. Such programs have been commercially available for well over a decade. Most of them run on PC's under DOS, and there are now a few screen-readers for the Windows platform. Early screen-reading programs relied on the character representation of the contents of the screen to produce the spoken feedback. A significant amount of research and development has been carried out to provide access to Graphical User Interfaces (GUI). These provide spoken access by first constructing an off-screen model (OSM) —a data structure that encapsulates the information displayed visually—and then using this OSM to provide spoken feedback. 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.

As is clear from the above, screen-readers for the UNIX environment have been conspicuous in their absence[2].

Emacspeak is an emacs subsystem that provides complete speech access under UNIX. Emacspeak will always have the shortcoming that it will only work under Emacs. This said, there is very little that cannot be done inside Emacs, so it's not a real shortcoming.

Emacspeak does have a significant advantage: since it runs inside Emacs, a structure-sensitive, fully customizable environment, Emacspeak often has more context-specific information about what it is speaking than its commercial counterparts. In this sense, Emacspeak is not a "screen-reader", it is a subsystem that produces speech output. A traditional screen-reader speaks the content of the screen, leaving it to the user to interpret the visually laid-out information. Emacspeak, on the other hand, treats speech as a first-class output modality; it speaks the information in a manner that is easy to comprehend when listening. The traditional screen-reading paradigm suffers from a severe shortcoming —the user has to interpret the semantics encapsulated in the visual layout in order to arrive at the meaning of the information displayed by an application. In contrast, Emacspeak —a direct speech access system— speech-enables specific user applications to to *speak* the information that is being conveyed to the user. By doing this, Emacspeak has much more contextual knowledge about the information being spoken than does a conventional screen-reading program.

2 Motivation

Emacspeak was motivated by my desire to run a multitasking OS on my laptop. Before Emacspeak, the only way I could access a UNIX workstation was via a PC emulating a talking terminal —a crufty if workable solution on the desktop. However, this was clearly impractical in the mobile environment —I would have had to carry two laptops!

The available options at the time[3] were:

- Run DOS on the laptop and be limited to a highly restricted environment.
- Run Windows 3.1 on the laptop with a commercial screen-access package for Windows —most of which were still flaky to say the least.
- Run Linux on the laptop and get the advantages of a 32-bit OS with full multitasking capabilities.
- Run OS2 with IBM ScreenReader.

At the time I approached the problem, Linux was the most attractive solution —except that there was no speech access system available for Linux (or any other UNIX).

3 Development Of Emacspeak

Initially, I decided to write a speech-enabling extension to Emacs as opposed to writing a conventional screen-reader at the TTY level in order to get a working system in the most expedient manner editting editing possible. The first working prototype of Emacspeak took under a week to design and implement. Once this prototype was working, the advantages of the speech-enabling approach outlined earlier became apparent. I then decided to turn Emacspeak into more than a prototype —Emacspeak turned into my full-time speech access interface.

Using Emacs' power and flexibility, it has proven straightforward to add modules that customize how different applications provide spoken feedback, e.g., depending on the major/minor mode of a given buffer. Note that

[2] This means that most visually impaired computer users face the additional handicap of being DOS-impaired – a far more serious problem!

[3] October 1994

the basic speech functionality provided by Emacspeak is sufficient to use most Emacs packages effectively; adding package-specific customizations makes the interaction much smoother. This is because package-specific extensions can take advantage of the application context to provide appropriate feedback.

Emacs-19's font-locking facilities are extended to the speech output as well; for instance, a user can customize the system to have different types of text spoken using different kinds of voices (speech fonts). Currently, this feature is used to provide "voice locking" for many popular editing modes like c-mode, tcl-mode, perl-mode, emacs-lisp-mode etc.

Emacspeak currently comes with speech extensions for several popular Emacs subsystems and editing modes. I would like to thank their respective authors for their wonderful work which makes Emacs more than a text editor —Emacs is a fully customizable user environment.

Here is a partial list of the various editing modes and applications supported by Emacspeak:

W3 A powerful Emacs-based WWW browser.

HTML-HELPER Publishing on the WWW.

VM A mail reader.

GNUS A USENET news reader.

BBDB The Insidious Big Brother Data Base. This is a powerful rolodex system that can be used to maintain and automatically update a rolodex.

CALENDAR A tool for maintaining appointments etc.

HYPERBOLE A powerful hypertext and hyperbutton system that allows the user to organize and find information.

ROLO Yet another rolodex system that comes with Hyperbole.

KOUTL An outlining editor for maintaining structured documents.

AUCTEX Editing (LA)TEX documents.

DIRED Emacs' file manager.

OOBR A powerful code viewer for browsing object oriented code. The interface provided is similar to the one provided in the Small talk world.

GDB A powerful interface for debugging C and C^{++} code.

CC-Mode C and C^{++} editing extensions.

INFO Browsing online documentation.

MAN Browsing online UNIX MAN pages.

Folding Using Emacs as a structured folding editor.

TEMPO A package that allows for editing templates.

ISPELL A powerful interactive spell checker.

CALC A powerful symbolic algebra calculator.

ETERM Launching a terminal inside Emacs. This extension enables you to login to another system and get spoken feedback, as well as running programs that can only be run from the shell. With this extension, Emacspeak can do everything that a screen-reader written at the TTY level would achieve. For instance, you can run a VI[4] inside the terminal emulator and get complete spoken feedback from Emacspeak.

BUFFER-MENU Navigating through the list of currently open buffers.

Comint Interacting with command interpreters running in an inferior process. This allows the user to run inferior UNIX shells, Lisp or TCL processes etc.

EDIFF A powerful interface that allows the user to compare files, apply patches etc.

[4] VI is the default editor on most UNIX systems

TCL Supports editing and interactive debugging of TCL scripts.

PERL Supports editing of PERL scripts.

4 Advantages

Emacspeak is a speech output system, and as such describing the output from Emacspeak in print is clearly impossible. Suffice it to say that the spoken feedback provided by Emacspeak is qualitatively superior to that of the traditional screen-reading approach[5]. In this section, we point out some of the features of the spoken feedback provided by Emacspeak.

The Main Differentiator

The primary difference between Emacspeak and conventional screen-readers can be summarized by saying that Emacspeak extends individual applications to speak as opposed to speaking the final display produced by individual applications. On the surface, the approach of extending each individual application to speak might seem intractable. On closer examination however, it becomes obvious that speech-enabling applications is in fact the only way to provide appropriate spoken feedback to the user. Further, the implementation strategy used, providing appropriate hooks to key functions in an application, is a powerful technique for speech-enabling applications. We justify this assertion in the following paragraphs.

Every computing application can be viewed as having three distinct components:

- Obtain user input —get the data.
- Perform the computation.
- Display the result —generate output.

Conventional software assumes that the only mode of providing output is via a visual display. The screen-reading approach retrofits spoken output to this design in order to provide access to these applications. Clearly, the screen-reading approach has the advantage that providing spoken access does not require direct cooperation from the underlying application —but this is also its primary shortcoming.

The speech-enabling approach forces specific applications to treat speech as a first-class I/O medium. This means that the speech output modules do not have to wait until the information is finally displayed to the screen before speaking it —the speech module can access the application context and generate the spoken output using all the information that was available to the visual output routines.

The design used in Emacspeak can also be described as follows. Dr. Jim Thatcher describes his IBM ScreenReader as an interpreter that executes specific scripts to provide application specific feedback. This makes IBM ScreenReader[6] one of the most powerful screen access systems. In the case of IBM ScreenReader, the application specific scripts (ScreenReader Profiles) as well as the ScreenReader interpreter that runs these scripts both run at global scope (top-level). Emacspeak goes one step further —the application specific scripts as well as the core speech output routines all run within the application context.

[5] I have used screen-readers for the last 5 years.

[6] I used this screenreader exclusively for 5 years until I wrote Emacspeak.

References

[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.

[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.