Architecture and requirements for a Windows screen reader

Paul Blenkhorn & Gareth Evans
CRESST (Centre for Rehabilitation Engineering, Speech and Sensory Technology)
Department of Computation, UMIST, PO Box 88, Manchester, M601QD
p.blenkhorn@co.umist.ac.uk

1 Introduction

For nearly twenty years blind people have been successfully accessing standard applications on desktop computer systems using screen readers [1, 2]. These programs can echo user input and provide screen information to the user by using synthetic speech and/or a Braille display.

This paper describes two aspects of screen readers. Firstly, it describes the mechanisms currently available for capturing the activities of the user and the user's application(s) that normally result in updates to the screen display. Secondly, the interface for presenting that information in an appropriate form to blind users is discussed. In both cases these are illustrated by reference to an ongoing project to produce a Windows (95, 98, NT4 & Windows 2000) screen reader for blind people, called Lookout. In the first instance this project only uses synthetic speech and, consequently, we will not discuss Braille output in this paper.

2 Capturing user input and application activity

In earlier operating systems, e.g. MS DOS, the screen was memory mapped in character/attribute pairs (a Character User Interface (CUI)) [5]. Accessing this information was relatively straightforward once the address of the screen buffer and the current cursor position were known. Relatively simple 'hooking' mechanisms[1] enabled a screen reader to capture keyboard input and read appropriate information from the screen. As desktop applications became more sophisticated, the screen readers developed more facilities, for example, to enable them to monitor parts of the screen to detect if, for example, a highlight bar moved, or if important information changed on a status bar.

When desktop systems moved from a CUIs to a Graphics User Interfaces (GUIs), the relatively simple mechanisms used for CUIs were no longer appropriate. This was because the screen display moved from being memory-mapped characters to a pixel based system, requiring far more sophistication to determine information about the screen's contents. On a more positive note there has been strong tendency for applications (at least on the same operating system) to have a similar user interface, e.g. on English systems ALT-F S almost always performs a save operation.

In this paper, we will focus on the GUI provided by Microsoft Windows. Whilst the details are specific to Windows machines, the basic principles should apply to other GUIs.

2.1 The Mouse

Most blind users readily solve the difficulty of using the mouse by the simple expedient of relying on keyboard shortcuts [3]. This poses few difficulties except for a few rogue applications that do not have keyboard shortcuts. Screen readers provide mechanisms to deal with such applications (see Section 3). Essentially they allow the user to control the mouse cursor from the keyboard.

2.2 The Keyboard

In a similar manner to the strategy adopted for the CUI interface, the keyboard activity of users can be filtered[2] by Windows specific 'keyboard hooks' [6]. In most cases the filtering is used to speak characters or words typed, or detect other keyboard operations, such as cursor up, and respond appropriately. In addition, sometimes the key hooks are use to detect commands specifically for the screen reader, e.g. *read contents of foreground window*, in which case the keystrokes are removed from the input buffer before any applications can receive them.

---

[1] We use the term 'hook' to means a piece of software that intercepts calls made on another piece of software. The hook passes the calls on the original destination, but also makes information about these calls available to a further piece of software, such as a screen reader.

[2] We use the term 'filter' to indicate that the information gathered by a 'hook' may be used by the screen reader in different ways depending upon the values of the input and the current state of the system and the screen reader.

2.3 *Application activity*

One of the difficulties with Windows applications is that the screen information is presented in pixel format on the user's screen with no textual representation accessible via the operating system. The traditional method used by GUI screen readers [6] to address this problem is to hook Windows messages to determine screen updates. The Windows messages from the Graphical Display Interface (GDI) are monitored by a display 'hook'[3] to maintain an *off-screen model* that is essentially a data structure and associated Application Programming Interface (API) for screen information. The screen reader queries the off-screen model to retrieve information about the contents of the screen. For example, a query on a particular pixel can be used to determine whether that pixel is part of a character and, if so, the value, font and screen dimensions of the character will be returned. In addition to queries, the off-screen model can also generate events, for example, if the blinking caret in Word moves. These events are used by the screen reader to determine that a change has occurred and the screen reader, will, when necessary, inform the user. So, for example, if a user cursors up in MS Word then a screen reader event will fire when the caret moves, the screen reader will have kept a record that the last keystroke was Cursor Up, the screen reader will query the off-screen model to determine the text on the new line, and will then normally speak the line.

Figure 1, shows the architecture of a typical screen reader.

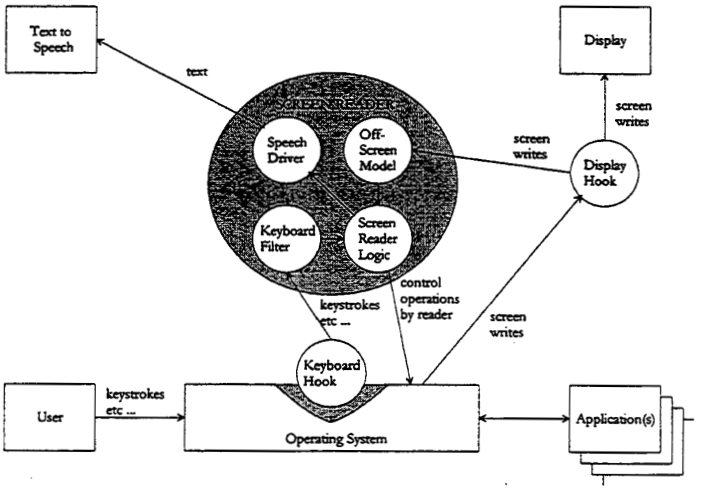

Figure 1: The Architecture of Simple Screen Reader

In Figure 1, the screen reader obtains most of its information from the off-screen model. Using this technique relatively sophisticated heuristics are required to determine the current focus. For example, when the user 'cursors up' on a menu there are several updates to the screen that clear the background of the previously selected item and highlight the new item. Another example concerns as user using the Tab key to move around selectable objects, when the focus changes to a button, there are a number of screen writes that need to be interpreted in a sensible way to determine that a button has been selected. A further problem with using the off-screen model is that, in the client areas of many applications, sophisticated heuristics are required to determine the current selection. For example, currently selected cell in Excel is marked by a bounding rectangle with a small group of pixels in the bottom right of the cell. Detecting this requires considerable processing.

The Look Out screen reader uses to additional sources of information to increase reliability and efficiency. These are discussed below.

2.3.1 Microsoft Active Accessibility (MSAA)

"Microsoft Active Accessibility is a suite of technologies for the software developer that lets applications and objects communicate and automate more effectively. MSAA allows applications to actively cooperative with accessibility aids, providing information about what they are doing and the

[3] In this description the off-screen model obtains its information by 'hooking' the calls from the GDI. However, this is not the only means of capturing information suitable for the off-screen model. Another approach, which is more suited to NT than Windows 9x, is to develop a custom device driver.

contents of the screen" [4]. Applications that support Active Accessibility follow a common standard to generate events and expose their contents to other applications. Currently many applications do not fully support Active Accessibility - including popular office applications, such as Microsoft Word. However almost all applications support Active Accessibility to some degree, for example the selection of options from pull-down menus and the information in dialog boxes. When the focus changes in an application, in many cases active accessibility can generate an event to inform a screen reader that the focus has changed. The screen reader can then send an Active Accessibility query to determine characteristics, e.g. the name of the new object with the focus. Unfortunately the current situation with some popular applications, for example Excel and Word, is that the focus event only indicates the fact that focus is in the client area and not the exact position in it. There, the off-screen model must be used to obtain the information.

2.3.2 OLE Automation

OLE automation is a technology from Microsoft that enables different applications to work together. It enables one application to access the Object Model of another application (provided that the application supports OLE automation). One application can enquire about the properties and can even execute commands within another application, e.g. a Visual basic program can be written to launch Word, load a document into Word and then change text, font, tables, etc. within that Word document. This technology can be of particular value for determining focus information in difficult applications, e.g. the focus in Excel or the position in a table in Word.

Figure 2 shows the architecture of the Look Out screen reader. A discussion of which components are used in particular circumstances is given in Section 3.

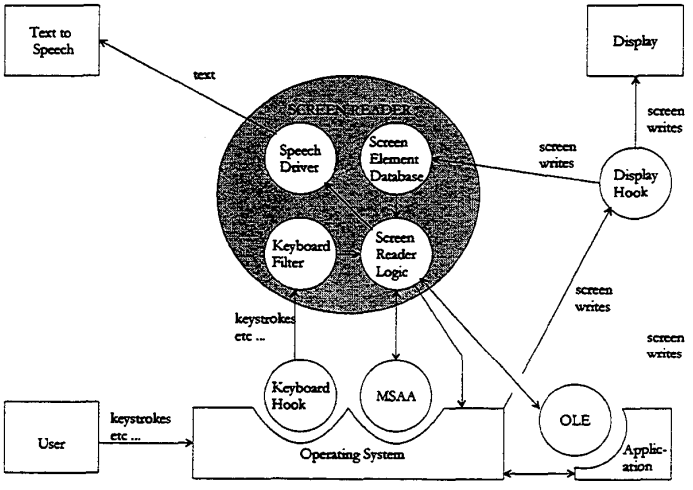

Figure 2: A simplified view of the Look Out Architecture

3 The Architecture of Screen Readers

A screen reader can use the information provided by the off-screen model, MSAA and OLE to vocalise user interactions such as key presses, the presentation of dialog boxes, menus etc. An important user requirement is that the screen reader responds in an appropriate and meaningful manner. A blind user will typically navigate through an application using keyboard commands. CTRL-ESC to bring up the start menu; ALT-TAB to step between applications; the Cursor Keys, Tab and Enter to move between screen objects; and so on. As the focus changes the new focus is voiced.

However, determining what should be voiced in a particular situation can be a complex decision. This can be illustrated by what, at first, appears to be a simple example. What would be a meaningful vocalisation following a user pressing the 'cursor up' key? The answer depends on the context. If the 'cursor up' results in a new menu item being highlighted, that item should be spoken; with additional information being spoken to indicate the case of an item that has a sub menu. If the 'cursor up' results in the cursor in a word processor moving up a line, the new line should be spoken. If the 'cursor up' results in a change on the screen, that change should be spoken, e.g. if the focus is on a spin-button and the 'cursor up' results in an associated numerical value in a text box being incremented, the new value in the text box will be spoken.

One strategy for addressing the subtleties in operation is to build the special cases into the screen reader. This is problematic, as blind people may wish to access any Windows application, and of course, the most popular applications are changed on a fairly regular basis. The approach taken in the Lookout screen reader is to:

- Take Active Accessibility as the primary reliable source of changes in applications and the focus.
- Use the off-screen model as a fallback position for applications where the focus does not support Active Accessibility. Look Out uses an off-screen model called the Screen Element Database (SED) – see Section 5.
- Where the strategy above fails to give satisfactory results there is the option to have application specific code (written in VBScript) loaded when the focus changes to a new application. The application specific script code can access the information provided by both MSAA and the SED, but can also access the object model (if there is one for that application). This is the method that is used, for example, to vocalise the spell check window in Word and to provide details of the focus in Excel.

There are some additional difficulties that are worthy of mention. What can the blind user do if an application requires a user to click on objects which cannot take the focus (for example on multimedia CDs where 'links' are represented as bitmaps) or where the screen layout/tab order is not particularly friendly to navigate? Two options are available. Screen readers always provide the user with the option to roam around the screen using a *reading cursor* that also normally moves the mouse pointer. Keyboard options are available to simulate clicking of mouse buttons. The other option is for the screen reader to drive the application directly. Users can give commands to the screen reader, using the filtered keystroke method discussed above, which can then cause a script to run which may perform a number of actions such as moving the mouse pointer 126 pixels to the right and 25 pixels down from the top left hand corner of the foreground window – effectively clicking a specific button for the user. Look Out uses this technique to control the Media Player but effectively replacing the user interface with three keys on the numeric keypad: 4, 5, and 6 which correspond respectively to Pause, Play and Stop.

4 Discussion

The screen reader discussed here is still under development and user trials are underway. The results of that evaluation will be provided at a later date.

5 Acknowledgements

The SED and other low-level components used by the Lookout system are the joint work of O.N.C.E., Baum Products, euroBraille S.A. and Microsoft. Baum Products GmbH licensed the work to Stone Soup Software. The SED is available as 'open source'; further details can be obtained from Stone Soup Software www.stonesoupsoftware.org.

6 References

1. Allen S. I., Songco D. C., Plexico P. S. and Morford R. A. "A voice output module developed for a blind programmer" *Journal of Visual Impairment and Blindness*, 75(4), 157-161, 1981
2. Blenkhorn, P. and Caulderwood, D. "Access to personal computers using speech synthesis: A review of the past decade", *The New Beacon - The Journal of Blind Welfare,* LXXVI(898), 185-188, 1992
3. Blenkhorn, P. "Requirements for Screen Access Software using Synthetic Speech" *Journal of Microcomputer Applications*, 16, 243-248, 1993
4. Microsoft, http://www.microsoft.com/Windows/platform/Innovate/Marketbulls/msaamkybull.asp, current December 1999
5. Schreier, E.M., DeWitt, J. C., Goldberg, A.M., and Leventhal, J. D., "An evaluation of synthetic speech software programs", *Journal of Visual Impairment and Blindness*, February, 70-74, 1987
6. Swerdtfeger, R. S., "Making the GUI Talk." *Byte*, December, 118-128, 1991