User-Directed Screen Reading for Context Menus on Freeform Text

Ka-Ping Yee
Group for User Interface Research
University of California, Berkeley
ping@zesty.ca

ABSTRACT

This paper proposes a variation on existing screen-reading technology to help sighted users automate common operations. When the user wants to perform an operation related to some displayed text, the user can direct the window system to read text near the mouse pointer and offer possible actions. This can be considered an extension of the context menu applied to freeform text instead of GUI objects. The proof-of-concept implementation of this technique helps the user make appointments based on dates and times mentioned in e-mail.

Keywords

Screen reading, context menus, group scheduling, Hotclick, Smart Tags.

INTRODUCTION

A significant part of the work we do on computers consists of transferring information from one place to another. A piece of information is presented on the screen in one form, we interpret what it means, and then we re-enter it somewhere else in a different form. As our example here, we consider the task of making appointments in an electronic calendar based on messages received in e-mail. Other common examples include retyping URLs, e-mail addresses, telephone numbers, or street addresses.

Perhaps we could have computers take care of some of this work for us, if they could only understand what they were themselves displaying. Fortunately, a lot of this work has already been done in the form of screen-reading software, which is designed to extract text from the screen so that it can be converted to speech or displayed in Braille for the blind. A recurring theme in human-computer interaction is the idea that designing and developing technology to improve accessibility for a particular group can lead to improved flexibility and capabilities for everyone. Here we explore one possibility for applying screen-reading technology to assist the sighted.

CHI 2003, April 5–10, 2003, Ft. Lauderdale, Florida, USA.
ACM 1-58113-637-4/03/0004.

CONTEXT MENUS

A widely used and effective user interface technique is the context menu. By clicking on a GUI object on the screen, the user can bring up a menu of commands relevant to the object. This interaction embodies an object-oriented model by enforcing that the noun (object) be selected first, then the verb (method). Among its advantages are the ease with which it lets the user ask "What can I do?".

However, this kind of interaction is typically available only for objects that are discretely identified within the software system, such as icons, hyperlinks, or window regions. It is interesting to consider how we might apply context menus to conceptual objects that do not yet have a distinct representation in the software system, particularly information mentioned in freeform text. Here, we experiment with using string pattern matching to determine the target of the action. We will use the term "text helper" to refer to a software component that recognizes patterns in text in order to offer a specific service; an appointment-maker and a URL-follower would be two examples of text helpers.

CALENDARS AND SCHEDULING

A great deal of past work has been done in this area of group scheduling, including automated negotiation of meeting times, shared online calendars, and integration of calendars with e-mail. The approach we adopt here leaves complete control of the calendar in the user's hands; we merely automate the step of making an appointment based on the content of an e-mail message. Instead of relying on the text of the message to obey a particular format for expressing appointments (like [3], for example), we look for text patterns that indicate times and dates. This includes relative indicators such as "next Saturday" as well as absolute indicators such as "April 5". In the ideal case the task of scheduling an event is reduced to two clicks.

While the general problem of looking at an entire message to determine a meeting time is hard, since the message might mention many date-related or time-related terms, we can afford to match against terms as general as "tomorrow" because we know where the user is pointing.

RELATED WORK

Microsoft's Smart Tags feature [1] aimed to provide similar enhancements. A Web browser with Smart Tags would scan the text on pages for known keywords (such as

company names or stock symbols) and mark each recognized keyword. The user could then click on the marked words to get a context menu. On a company name, for example, the menu might offer to look up a stock quote, get a company report, or search for related news articles. The approach suggested here is different in two ways. First, we insert text-extraction functionality at the system level, not the application level. Thus, a single text helper could offer appointment-scheduling assistance anywhere text appears – in a text file, a Web page, an e-mail message, a dialog box, and so on. Also, text helpers become interchangeable components that users can choose to activate, rather than functionality determined only by the application author. Second, text extraction is initiated and directed by the user, rather than having the application pre-scan the text. This avoids the visual clutter created by marking many words on a page and saves the processing time required for pre-scanning an entire document. More importantly, it gives us much greater flexibility in the patterns we can match, because the pattern matching is directed by the location of the mouse pointer. For instance, it would not make sense for Smart Tags to mark every occurrence of the phrase "the 15th" in every document because it would occur too often in unrelated contexts. However, if the user clicks on "the 15th" we can suggest scheduling an event on the 15th of the month.

The Opera Web browser has a Hotclick feature [2] very similar to what we suggest here: the user can select any piece of text, then right-click on it to get a context menu. The menu offers to look up the selection in a dictionary, translate it into another language, and so on. However, the user is required to select the exact target string, and the context menu is always the same. This work takes a middle ground between Smart Tags and Opera: pattern matching is applied to find relevant text and commands, but text extraction is initiated and directed by the user.

SOFTWARE ARCHITECTURE

We envision the design of a general architecture for supporting user-directed screen reading as follows. It is helpful to establish a clean separation between the screen reader and the applications. Screen reading functionality fits well at the window manager level, since the window manager knows the layout of the screen and can grab mouse events before they get to applications. The window manager reserves a special action to mean "What can I do with this text?" All of the available text helpers pre-register with the window manager.

When the user initiates a screen-reading operation, the window manager first extracts the text in a region around the mouse pointer. It could obtain the text by querying the application under the mouse pointer directly, by communicating with the text-bearing toolkit widgets near the mouse pointer, or by performing optical character recognition on an image of the screen. Then the window manager asks each text helper what it can do with the text. Finally, the window manager gathers the answers from all the text helpers to form the context menu.

PROTOTYPE

The prototype implementation is integrated with the Sawfish window manager and extracts text from Gnome Terminal windows. As it is just a proof of concept, it does not perform OCR; it relies on the Gnome Terminal to get text. Holding down Alt while clicking the right mouse button on a piece of text triggers user-directed screen reading. Since visual proximity does not always correspond to textual proximity, we use the following algorithm for finding "hits" on the text. We define the "hot region" to be a circle with radius 20 pixels around the mouse pointer. For any line of text that intersects the hot region, we extract the entire horizontal line of text. We also extract one additional line of text beyond the first and last lines thus selected. Then we send this entire string to each text helper for pattern matching. For each phrase match returned by a text helper, we then examine the bounding box of the phrase on the screen to see if it intersects the hot region. This method provides plenty of context for text helpers, so that matched phrases can extend far beyond the hot region into the surrounding text, but avoids registering hits too far away from the cursor.

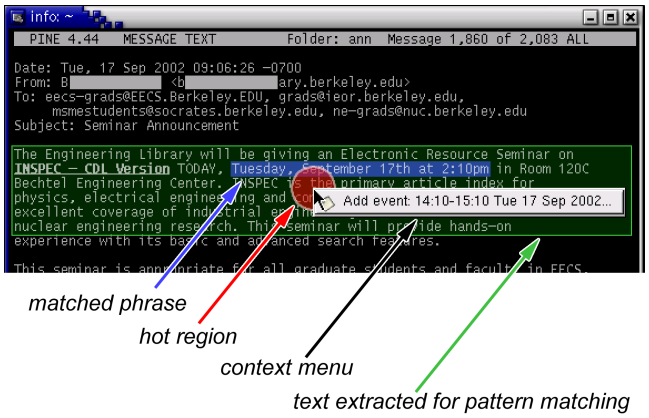

Figure 1. Finding hits in the text near the mouse pointer.

CONCLUSION

We have proposed an idea for combining screen-reading technology with context menus to automate some common user actions. We have implemented a prototype that helps users schedule events based on dates and times mentioned in freeform text as a proof of concept. This is an interesting instance of how assistive technology for the blind can improve interaction for sighted users.

REFERENCES

1. Microsoft. Office XP Tour: Simplifying Productivity. See: http://www.microsoft.com/office/evaluation/tour/page2.asp
2. Opera Software. Built-in Search and "Hotclick" Features. See: http://www.opera.com/privacy/search/index.dml
3. S. Sen, T. Haynes, and N. Arora. Satisfying User Preferences While Negotiating Meetings. In International Journal of Human-Computer Studies, 47:407–427, 1997.