Showing the destination of hypertext links: a new approach for Guide

Maria da Graça Campos Pimentel *

Computing Laboratory, University of Kent at Canterbury, U.K.
SCE–ICMSC, Universidade de São Paulo, Brasil.

May 1, 1992

Abstract

All hypertext systems have a kind of button which can be selected interactively by the user to obtain further information. This paper is concerned with how to show the user the further information when he[1] selects a button. We comment how several hypertext systems show this information, describe in detail how the Guide system currently does it and propose two alternative new approaches, the *previewing operation* and the *highlighted replacement approach*, discussing their implementation.

Key words: Hypertext, User Interface, Getting Lost, Link Following.

1 Introduction

The aim of hypertext technology is to present information to a user in a fast and interactive way. All hypertext systems have a kind of button which can be selected interactively by the user to obtain further information. We say that there is a $link$ between the button and the further information, and we say that when the user $selects$ the button he follows the link. The reference [1] may be consulted to give an initial overview about hypertext systems.

The problem we are concerned with in this paper is how to show the user the further information when he selects a button. Initially, in section 2, we describe how different hypertext systems show this information. After that, we present in section 3 some detail on how the two implementations of the Guide[2] Hypertext system [2] [3] show the user the information related to one type of button, the replace button. Afterwards we propose two alternative ways (sections 4 and 5) for Guide to present to the user the further information, and we discuss the design problems concerned with these proposals. Finally, in section 6, we present the current stage of our work and comment about our future work.

*This research is under supervision of Prof. Peter J. Brown in the Computing Laboratory at UKC and is being supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil) under grant 0813/90-5.

[1]NB: Every time we have writen *he* or *his* in this paper we refer to *the user* who can be either a woman or a man.

[2]GUIDE is trademark of OWL International, Inc.

2 Some current approaches

In this section we describe how several systems show to the user the further information related to the button the user selects. We describe in each sub-section the approach adopted and cite some of the systems that use it.

2.1 A new window in a multiple windows scheme

In the 'a new window in a multiple windows scheme' the user normally has several windows at once and, when the user selectes a button, the system puts on the screen a new window containing the further information. The new window coexists on the screen with the previous ones. One window is opened for each link the user selects. The user can normally see at least part of the information in the earlier windows and may select those windows to see their full content. This is the approach of NoteCards [4], Intermedia [5], for example.

2.2 A new window in an one window scheme, or card scheme

In the 'a new window in a one window scheme', or 'card scheme', the user always has one unique window related to his hyperdocument (other windows may exist, but have other related functions). When the user selects a button, a new window with the further information totally replaces the old one. The user is not able to see the earlier information. This is the approach of Hypercard[3] [6], and the approach of Guide when used to simulate the card approach as in [7].

A similar situation occurs with the KMS system[8]: it permits *one* or *two* windows to be put on the screen simultaneously. Since all the information in the system is kept in *frames* that have full screen size, the following arrangement is necessary: if one window is presented on the screen, it corresponds to a whole frame; if two windows are presented, each one corresponds to the left half portion of a frame. What occurs, actually, is that the frames that are presented in half size have their right half empty. The system authors comment that the most usual size of frame is the half size one, and that the full screen size is used only to keep very complex information.

2.3 A footnote resumé followed by a new window in a one-window scheme

In the 'a footnote resumé followed by a new window in an one-window scheme', when the user selects a link, the system shows, firstly, a resumé of the target information. This resumé is typically put in the bottom of the screen. The user then decides if he wants to follow the related link; if he does, a new window is put on the screen, totally replacing the old one. Hyperties [9] uses this approach.

2.4 Replacement of the button *in situ* by the new information in the same window

or **Go to** *versus* **Come here** ...

In the 'replacement of the button *in situ* by the new information in the same window scheme', the button selected by the user is replaced *in situ* by the related text (we use the word 'text', though in general the information could involve pictures). It works in the following way: the user selects a button using the

[3]Hypercard is a trademark of Apple Computer, Inc.

mouse cursor and presses a mouse button; the system highlights the text of the button (normally one line long); the user releases the mouse button and the system replaces the text of the button, *in situ*, by the related information (perhaps several paragraphs long).

The main difference to the previous approaches is that the new information is brought to the user when he selects the button instead of the user's attention being directed to another window. The Guide system uses this approach.

Actually, this *replace in situ* approach of Guide is more general since any region within the current window, rather than just a button, can be replaced. If the region is the whole window then Guide can simulate the card approach — hence the above reference to Guide under card schemes.

The Guide system has, in fact, several types of button. The situation described above occurs when the selected button is a *replace button*. There are some other types of button in Guide but we do not discuss them in this paper.

In the rest of this paper we refer to figures that are in appendix A. The appendix is annotated so one can read the appendix as an whole section and have a resumé of the discussion in this paper.

3 How Guide works today

The Guide system has currently two distinct versions: the OWL Guide 3.0, the system that runs on PC-like and Macintosh[4] computers and is available from OWL Ltd, and the Unix[5] Guide, which runs on workstations and is available from the Computing Laboratory at the University of Kent at Canterbury. The *replace button* structure is the name in the Unix Guide version; the corresponding structure in the OWL Guide is called *expansion button*.

We focus, in this section, on both Guide versions' approaches to showing the information associated with a replace button, and give a detailed view on how it works.

3.1 A detailed view on Unix Guide

We describe in detail how the replacement is currently done when the user selects a Unix Guide replace button.

In figure 1 we have a Unix Guide window. The top line of the window is the Unix Guide menu, and the rest of the window is occupied by the current document. Within this document, replace buttons are indicated by the use of bold font.

The following steps occur when a user decides to select a replace button:

- the user puts the mouse cursor over the replace button – in this example the user chooses the button **Index** in figure 1;
- the user holds the left mouse button down and the system highlights, using reverse-video, the text of the replace button (figure 2);
- the user releases the left mouse button; the system replaces the text of the button by the new text,

[4]Macintosh is a registered trademark of Apple Computer, Inc.
[5]Unix is a trademark of Bell Labs.

keeping the new text highlighted for a while (figure 3) and, afterwards, showing the new text as ordinary text (figure 4).

The user does not need to notice all these steps all the time. For instance, if he just clicks the left mouse button, he sees only the new text 'blink' on the screen before it becomes ordinary text.

After the replacement text has become part of the document, the user can verify its extent by pointing anywhere within the replacement and holding down the middle mouse button. This operation highlights the portion of the text related to the replacement and, at the same time, shows the user (inside a pop-up-menu) the name of the button that generated the replacement (figure 5).

The operation described above is, in fact, part of the 'undoing the replacement' operation: if the user releases the button when the cursor is on the pop-up-menu, the system replaces the new text by the original button. If the user does not want to undo the replacement he must move the cursor off the pop-up-menu before releasing the middle mouse button. Again, the user does not need to be concerned with such detail if his intention is to undo the replacement: he may just click the middle mouse button and see the text blink and be replaced by the original button.

It is opportune to note that the Unix Guide actually replaces the source of the button by the related destination text. Often, however, it is desirable that the original button name appear at the head of the replacement, e.g. when the button number is a section heading. In the example described, the word Index appears at the start of the replacement. This is because the author has explicitly placed it there. There is an option in Unix Guide to maintain the button in its original position after the replacement has been finished: the replacement text is appended to the button.

3.2 A detailed view on the OWL Guide

We describe in a similar way how the OWL Guide system works when the user selects an expansion button.

In figure 6 we have an OWL Guide window. The top line of the window is the OWL Guide menu — the rest of the window is occupied by the current document. The expansion buttons, in this example, are indicated by the use of bold font. The OWL Guide system works with a one-button mouse and uses the approach of changing the cursor shape to indicate to the user the current context (whether the text under the cursor is ordinary text or an expansion button, for instance) but this feature is not shown in our figures. When the cursor is over ordinary text, its shape is arrow-like $\nwarrow$.

When a user decides to select an expansion button he:

- puts the mouse cursor over the expansion button, in this example **Japanese Consulate** in figure 6 will be selected. The cursor shape changes from arrow-like $\nwarrow$ to crossed-circle-like $\oplus$;
- holds the mouse button and the system highlights, using a light dashed-line, the text of the replace button (figure 7);
- releases the mouse button; the system inserts the new text, showing it highlighted for a while as Unix Guide does, but using the light dashed-line instead of reverse-video (figure 8) and afterwards showing it as ordinary text (figure 9). The cursor changes from crossed-circle-like $\oplus$ to square-like $\square$).

As in Unix Guide, the user does not need to notice all these steps all the time: if he just clicks the mouse button, he sees the new text being inserted in the document as ordinary text and a light dashed-line surrounding the new text blinks on the screen for a while.

To verify the extent of the new text after an expansion button selection, the user points anywhere within the replacement and holds down the mouse button. This operation draws the light dashed-line around the new text (but does not show the button name at same time as Unix Guide does). This operation is, as in Unix Guide, part of the 'undoing the replacement' operation: if the user releases the button the system replaces the new text by the original button; if the user does not want to undo the replacement, he must move the cursor off the highlighted area before releasing the mouse button. Again, the user does not need to be concerned with such detail if his intention is to undo the replacement: he just clicks the mouse button.

As with Unix Guide, OWL Guide actually replaces the source of the button by the related destination text and there is an option to keep the button in its original position after the replacement has been finished.

3.3 One possible problem

We present in this section six pairs of figures that can help us to identify a kind of getting-lost problem that can occur when using Guide. The examples are based on the Unix Guide but similar situations occur when using the OWL Guide.

Each pair of figures corresponds to a button selection. Part (a) of the figure shows the button before the selection and part (b) shows the result after the selection. Table 1 outlines what the text of the buttons and the text of the replacements are.

figure no.	replace button figure part (a)	replacement text figure part (b)
10	**word**	replacement
11	**Example**	Examples . . . Hypercard.
12	**Scheme**	The . . . done.
13	**Guide**	Unix-Guide
14	**new information**	new . . . selection
15	**worst example**	this. . . down

Table 1: Examples of replacements in Guide

In the first three pairs of figures, the replace button was isolated in a line. In the last three pairs, the button was embedded in a paragraph.

We comment now on the difficulty of identifying the replacement text after the selection has been done:

- we can see easily in figure 10 exactly what the replacement text is because it is one isolated word;
- in the figure 11, as the replacement is composed of only few words, its extent is also easy to visualise;
- in figure 12, the insertion of the paragraph caused the window contents under the button to move down, with the result that it is not so easy to identify the extent of the replacement text;

- the situation becomes worse for the next three sets of figures, 13 to 15, because a big replacement is embedded in a long paragraph.

It could be said that it is bad authorship to create big destination regions or to embed buttons within long paragraphs; perhaps this is true in most situations but it does occur particularly if the hyperdocument is created automatically from existing paper-based documents, or even linear computer-based documents — a quite common situation in practice.

4 The 'previewing' operation

To combat the problem of identifying the destination region we have designed and implemented a new approach which we call the 'previewing ' operation: the key to this approach is to show the replacement contents *while* the user is selecting the button. This approach must not affect the normal operation of showing the contents *after* the selection is finished. The previewing operation has some parallels with the operation described in section 2.3 for the Hyperties system. We have studied two alternatives for implementing the previewing operation and present them in the following sections.

4.1 The first alternative

One way of implementing the previewing operation is by showing the contents of the replacement text in a separate window *while* the user selects the replace button and waiting for the user to decide if he actually wants that text inserted in his document [10]. This approach is illustrated below:

- the user puts the mouse cursor over the the button **our idea** in figure 16 and presses the mouse button;
- the system then presents the related information in a separate window as shown in figure 17;
- if, after seeing the destination contents, the user decides to insert it as a normal replacement, he simply releases the mouse button: the system removes the preview window and inserts the replacement in the normal Guide way;
- if, after seeing the destination contents, the user gives up the button selection, he has only to put the mouse cursor out of the destination region and release the mouse button: the system removes the preview window and the original button is maintained in its original place.

4.2 The second alternative

A alternative way is to show the replacement in the ordinary way, but highlighted, *while* the user selects the button. To illustrate the idea, let us suppose that the user is interested in the contents of the button **More** that follows the word 'OPTIONS' in figure 18. What happens is:

- *while* the user presses the mouse button to select the replace button, the system:
    a. presents the destination as ordinary text, in the same position it will be when the selection is finished;
    b. highlights the destination text by putting a thin line arround the region (as in figure 19).

- if, after seeing the destination region contents, the user still wants to select the button, he simply releases the mouse button: the system removes the highlight and the text becomes ordinary;
- if, after seeing the destination contents, the user gives up the button selection, he has only to move the mouse cursor out of the destination region and release the mouse button: the system replaces the highlighted destination by the original source.

4.3 The chosen alternative

We have chosen the second alternative to implement the previewing operation because we think that the operation must be as similar as possible to the ordinary operation. In this case nothing changes if the user does not withdraw from the selection: the result of the previewing operation is the same as the normal operation; what changes is only the *way* in which the button selection is done.

The direct advantages of this proposal are:

- We can highlight the destination region of the link while previewing it, reducing the difficulty of identifying what the new information is.
- This is a better approach than the current one if the user wants to ‘have a look’ at that content before it definitively replaces the original button. Brown mentions in [11] that the user of the Locator system, a Guide application concerned with repair and maintenance, would like to have this option.
- The user can be more curious about navigating in the database: any undesired destination is easily undone.
- This operation can be easily implemented in other hypertext systems other than Guide: as we use the moment of selection to perform the previewing operation, most existing systems can implement this proposal simply by changing the moment of selection to *while* the user presses the mouse button instead of *after* he does it.
- The previewing operation can be made an optional feature of existing systems: this is achieved by performing the operation only if the user holds the button on for a while; if he just clicks the mouse this means that he does not want the previewing operation done.

4.3.1 Problem: replacement does not fit on the window

A problem exists: it is a difficult situation if the replacement does not totally fit in the window. This is an important case since the user is most confused when the replacement is huge, and he should be given an idea of its size.

One option is to perform continuous scrolling directed by moving the cursor up and down in a similar approach to vdiff[6] [12] or Macintosh applications that do scrolling using the ‘hand’ cursor. While the user presses the left mouse button during the previewing operation, the middle mouse button can be used to cause scrolling: the movement of the mouse up and down controls the direction and speed of the scrolling.

[6] Vdiff is a graphic version of the diff Unix utility, available from UKC, which provides continuous scroll of the document contents in a window while the user moves the mouse cursor.

Also, it can be very helpful to indicate some numerical proportion like '10% shown' so the user can estimate the size of the entire replacement.

4.3.2 Problem: replacement smaller than the source

When we described the preview operation, we said that the user has to move the cursor out of the destination region before releasing the mouse if he does not want the replacement to be done. In other words, we assumed that the mouse cursor is initially over the destination region during the preview.

However, the mouse cursor might be initially outside the replacement region. This happens in situations where the resulting replacement is smaller than the source: a case in point is when the selected button is embedded within an enquiry[7] and the associated replacement is smaller than the enquiry source region. The folowing figures illustrates this case.

In figure 20 all the replace buttons, which are the items numbered from **1** to **11**, are *embedded* within an enquiry. According to the definition of the enquiry structure, the result of the selection of an embedded button is the replacement of the enquiry region by the replacement text associated to the selected button. This means that the source of the replace button is not only the button itself, but also all the region defined by the enquiry structure. In the example, the result of the selection of the button **11. Simple animation** is shown in figure 21, the replacement text is presented highlighted so one can compare the size of the enquiry with the size of the replacement. Situations like this are frequently found in the Locator system.

The problem, in situations like the one exemplified, is that the mouse cursor is not over the replacement region when the replacement is highlighted. According to the definition of the previewing operation, the user has to move the mouse cursor out of the replacement region if he wants to undo the replacement. This suposes that the cursor *is* over the replacement area text when the previewing is initiated. As shown by the example, this is not always true.

If we mantain the definition of the operation, the result of releasing the mouse button in the example is to undo the replacement operation, since when the replacement button **11. Simple animation** is selected, the mouse cursor is on the very bottom of the screen and not over the replacement text. Even worse, if the user wants to mantain the replacement he has to move the mouse cursor to a position within the replacement text before releasing the mouse button.

From the above discussion, we feel that, if the cursor is not over the replacement region when the previewing operation initiate, something has to be done. Among the options we have are:

- wait for the user to put the cursor over the replacement before permiting him to do anything else;
- move the cursor to a position over the replacement, warping the mouse cursor;
- modify the way he have defined the previewing operation to eliminate the problem.

We probably are going to try all these alternatives, and evaluate user reaction to each.

5 The 'highlighted replacement' approach

The idea of this approach is to highlight the new information *after* the selection operation has been done. The aim is to help the user to identify the replacement text after the replacement has been finished. This

[7] the enquiry structure receives the name *group* in OWL's Guide.

approach does not conflict with the previewing one. After usability tests have been done, the results may indicate that both alternatives can be used at the same time.

To implement the highlight, we can:

- use a stippled background over all the replacement text so the user can identify more easily the extent of the replacement;
- leave the line that surrounds the replacement text in the previewing operation;
- change that line for a dashed line indicating the distinct context.

The next figures illustrate this idea. Figure 22 presents a Unix Guide window with the replace buttons **OPTIONS** and **SIGNALS**. If no highlight is used after a selection has been done, the result of the selection of the button **SIGNALS** is presented in figure 23. If a stippled background over the new information is used, the result is like that shown in figure 24. Figure 25 presents the result of highlighting the new text using the same line used in the previewing operation.

The main advantage of this approach is that the user is shown the extent of the replacement all the time. We believe this can help not only in a short term navigation but also in a very long one. One disadvantage it that the screen can become 'awful' if there are lots of little replacement texts shown. Another disadvantage is that the screen contents do not reflect the paper contents because the highlights are not intended to be saved.

One question raises: what occurs if the selected button is inside a highlighted region ? We think that the former highlight must be suppressed so that only the last level of replacement is indicated at one time. If the original button was not inside such a region, a new level of replacement is indicated at this time. The result is that the document can have several portions that are highlighted, each portion indicating that there is replacement in that position. It would not indicate, however, how many levels of replacements have occurred.

Among the alternative ways of implementing this approach, we must decide:

- which kind of highlight to use (stippled background, dashed line, continuous line, blinking line, etc.);
- how to indicate that a replacement extends outside the current window;
- whether the user can deactivate the showing of the highlighting of any replacement — permitting optional viewing of each highlight;
- whether the user can deactivate the showing of all highlights;
- whether the user can ask the system to show where the earlier replacements are. For instance, the system can disable the current highlight and show the previous one. This is a similar approach to that existing in the undoing of the replacement operation in Unix Guide;
- whether to perform the highlight of the previous level after executing an undo operation.

6 Current and further work

To evaluate our proposals, we are working with the Unix version of the Guide hypertext system.

We are now working in the implementation of the previewing operation. We have:

- changed the link selection operation to preview the destination region of replace buttons;
- implemented the highlight of the destination region during the previewing operation by drawing a line around the region.

We are now finishing the implementation of the scrolling operation during the previewing operation. The next steps are to:

- implement the 'highlighted replacement' approach;
- evaluate our approaches by tests with users;
- observe users and evaluate the suggestions they give about new alternatives.

We are also interested in surveying the applications that are already using Guide and identifying where the proposed suggestions of this paper are most relevant.

Acknowledgements

We acknowledge Professor Peter J. Brown for supervision, CAPES for sponsorship and the SCE–ICMSC–USP and the Computing Laboratory–UKC for support provided.

A Commented figures

The *current Unix Guide* approach

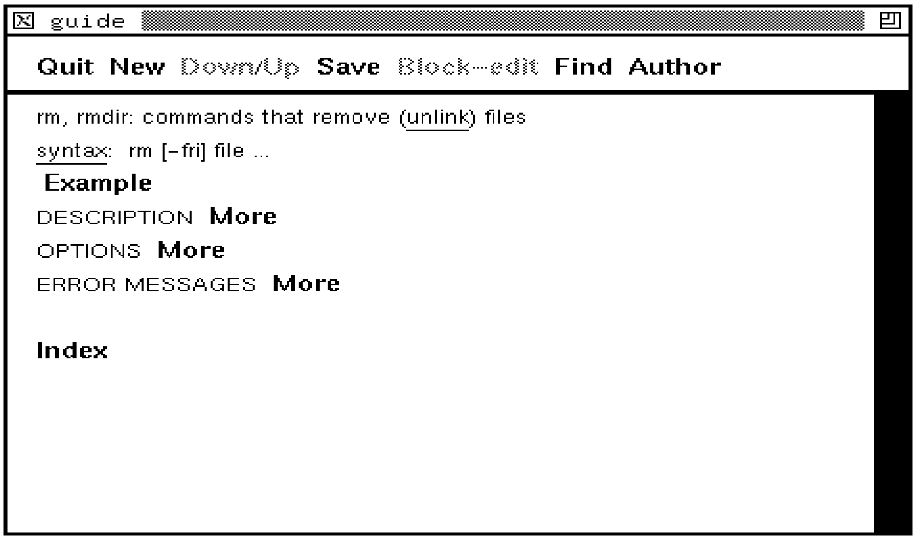


Figure 1: Example of a Unix Guide window

The top line of the window is the Unix Guide menu, and the rest of the screen is occupied by the current document. Within this document, replace buttons are indicated by the use of a bold font.

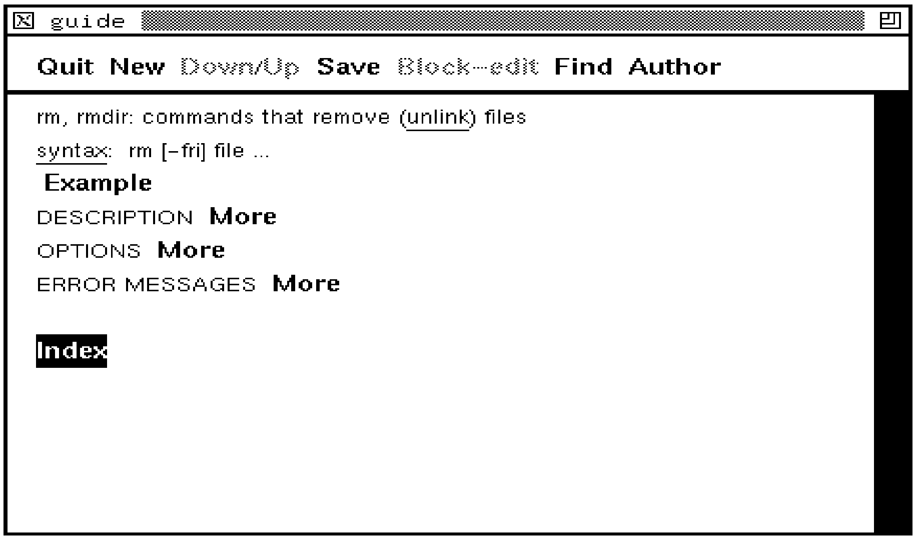


Figure 2: The user selects **Index**

To select a button, the user puts the mouse cursor over the bold font text and clicks the left mouse button: the replace button text is highlighted using reverse video while the user presses the mouse button.

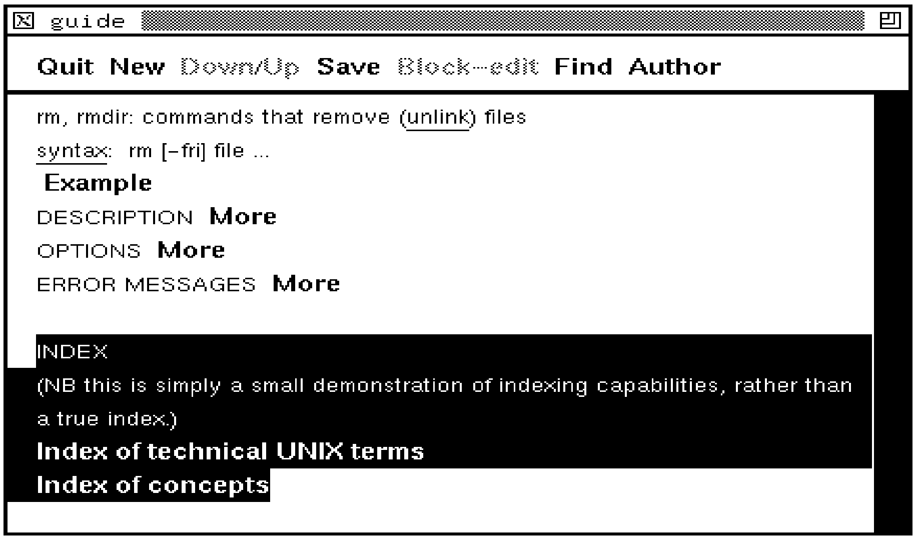


Figure 3: The replacement text is highlighted

When the user releases the mouse button, the system replaces the original button by its associated replacement, keeping the whole replacement highlighted for a while.

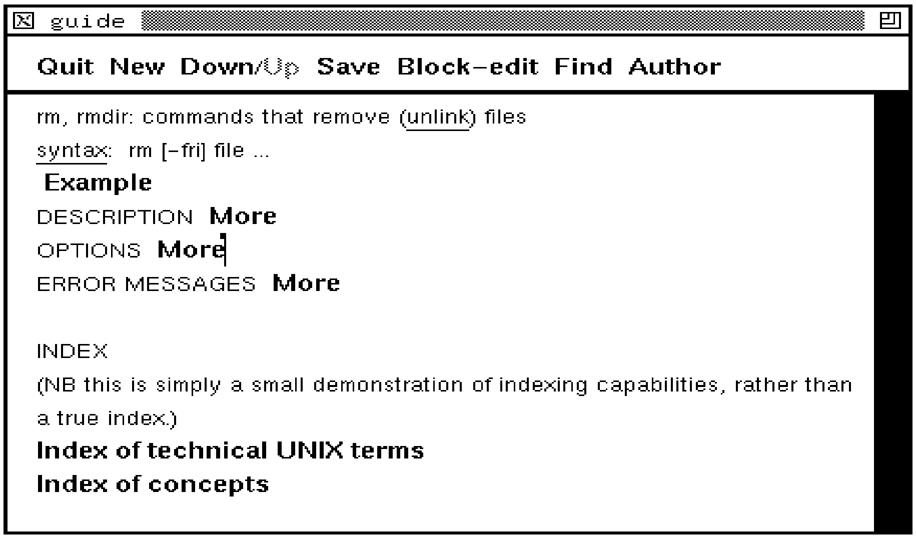


Figure 4: The replacement is finished

Afterwards, the text is shown in the normal way. As we can see, it is not easy to identify, at this point, what exactly has changed in the document information.

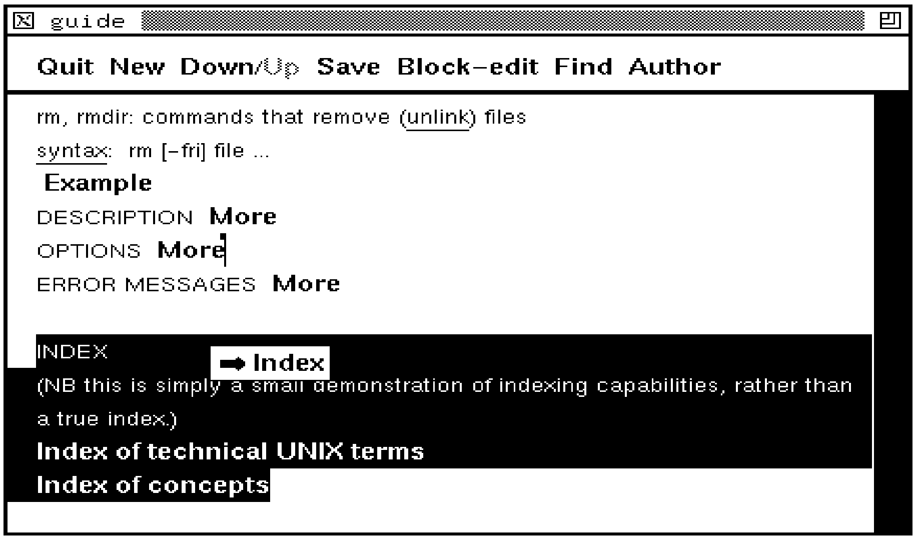


Figure 5: The replacement text is shown

The user may highlight the portion of the document associated with that replacement: he places the cursor anywhere in the replacement and clicks the middle mouse button, causing a highlight as shown above. This is, in fact, part of the undo operation.

The *current OWL Guide* **approach**

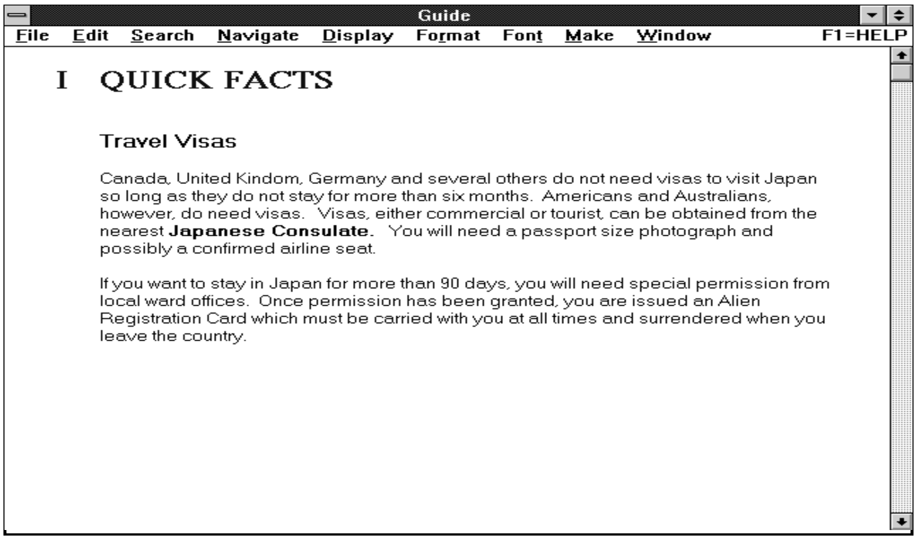


Figure 6: Example of an OWL Guide window

This and the next three figures correspond to the OWL Guide system. We show how the selection of a expansion button occurs. In this case, the user sees the expansion button **Japanese Consulate** in bold font in the document area (when over ordinary text, the cursor shape is arrow-like ↖).

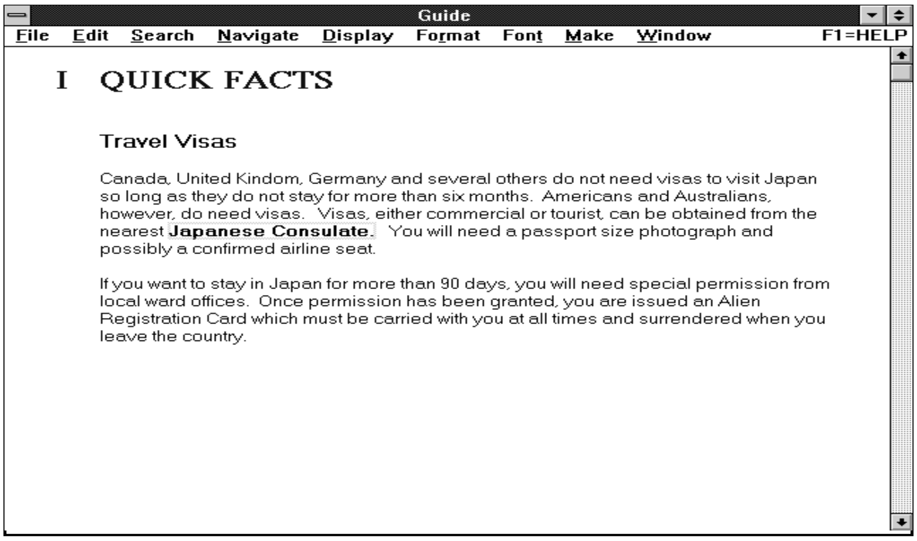


Figure 7: The user selects **Japanese Consulate**

The system highligths the button name using a light dashed-line while the user presses the mouse button over the expansion button name (the cursor shape changes from arrow-like ↖ to crossed-circle-like $\oplus$).

Figure 8: The systems inserts the new text and highlights it for a while using a light dashed-line

When the user releases the mouse button, the system inserts the related text, highligthing it for a while (the cursor changes from crossed-circle like $\oplus$ to a square like $\square$).

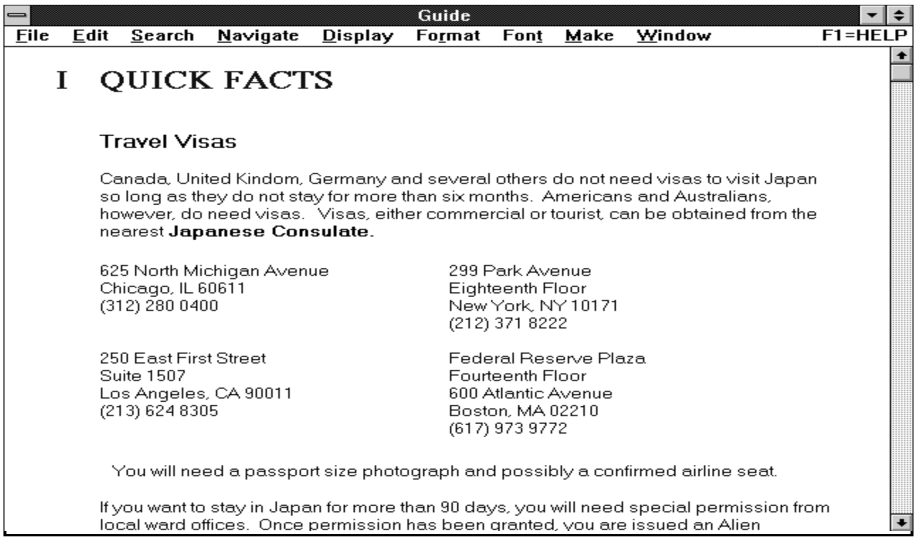
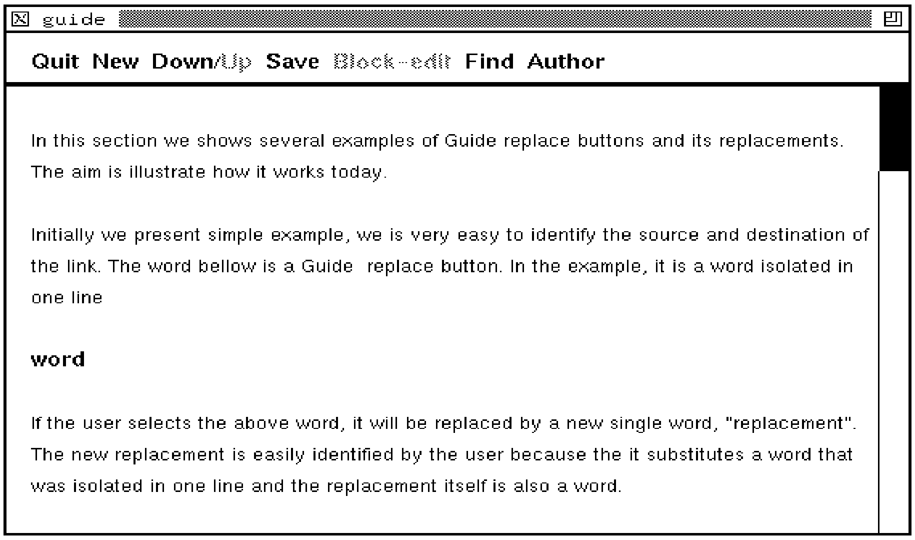


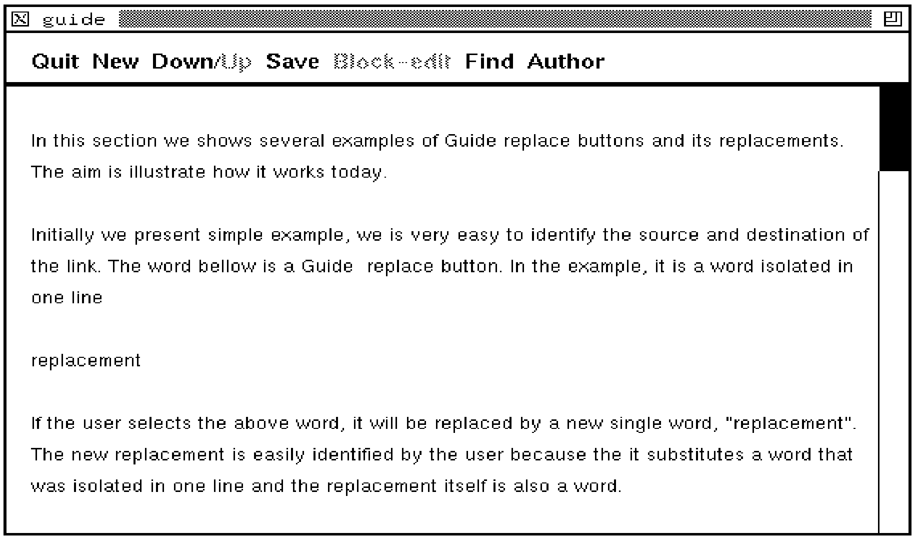
Figure 9: The new text is shown as ordinary text

The new text is shown in the ordinary way at the end of the operation. The next figures present six pairs of windows that show a kind of getting-lost problem that can occur when using Guide. The examples are based on Unix Guide but similar situations occur when using OWL Guide.

Some situations where it can be difficult to identify the destination region



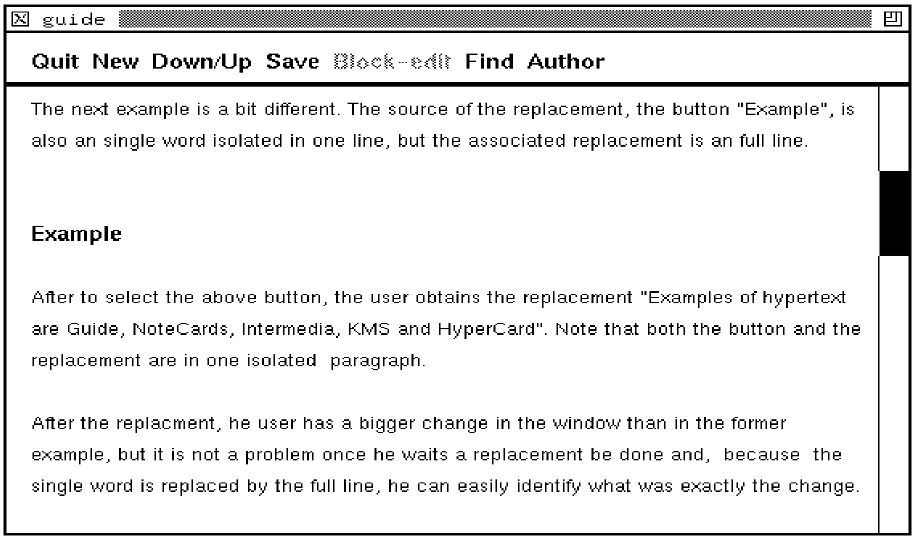
(a)



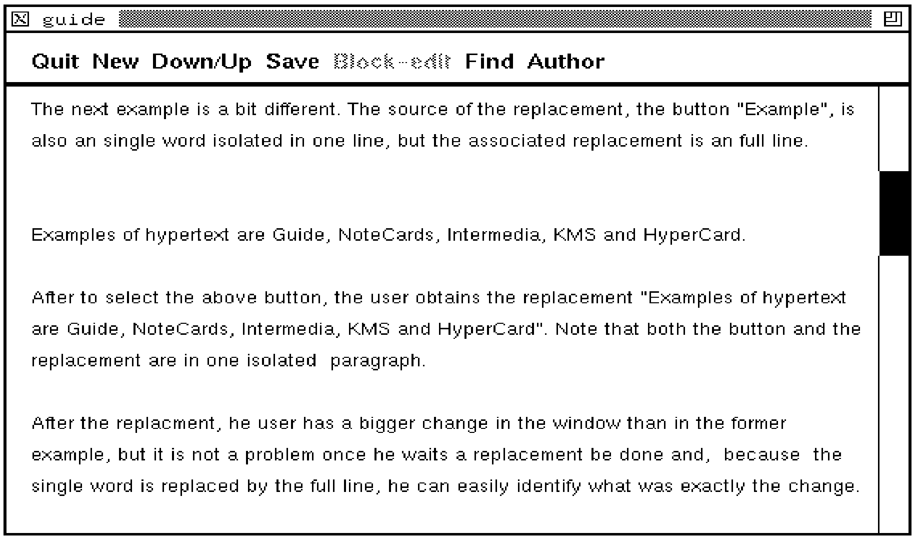
(b)

Figure 10: **word** is replaced by replacement

The button **word** in the top figure was selected and replaced by the word replacement, shown in the bottom figure. We can easily appreciate the change since both source and replacement are nearly of the same size and are, also, isolated in one paragraph.



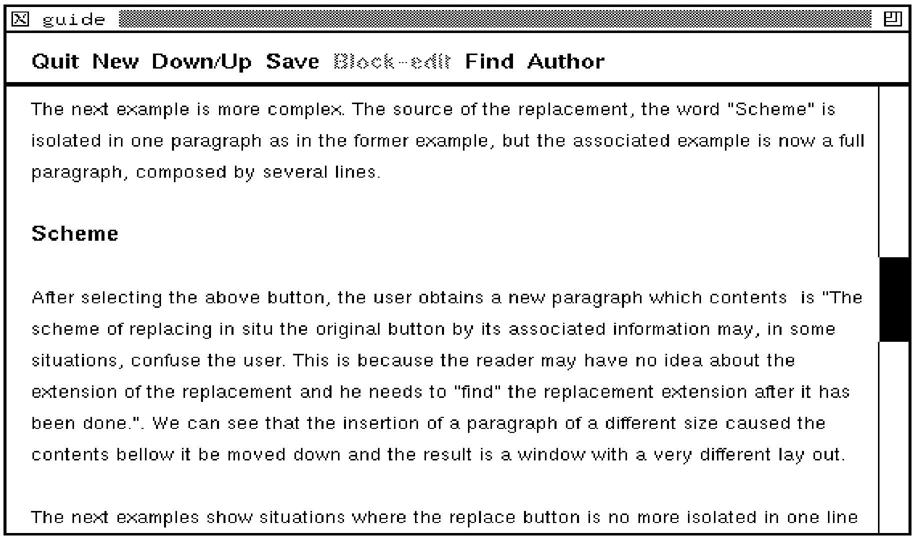
(a)



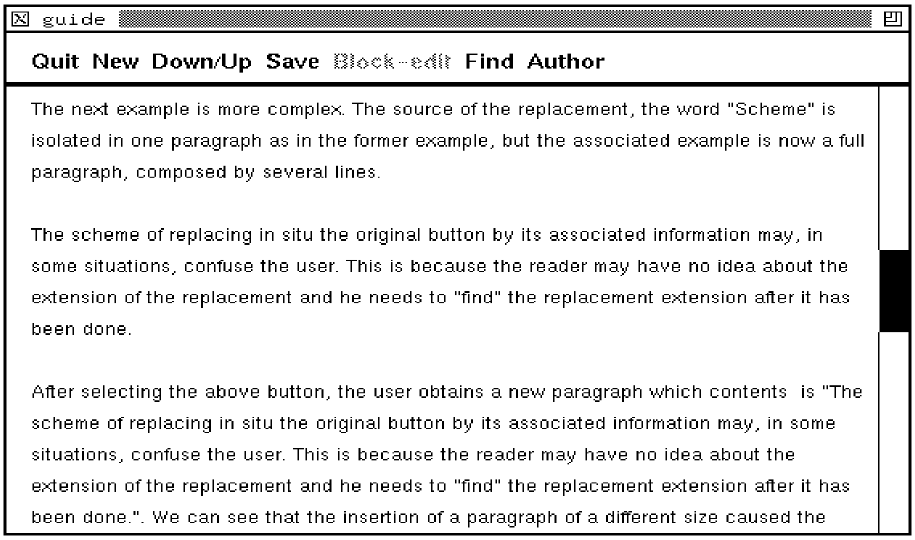
(b)

Figure 11: **Examples** is replaced by Examples . . . Hypercard.

In this case the extension of the replacement is not as easy to identify as before, mainly because the size of replacement is very different. However the result is still clear because both source and replacement occupy an one-line isolated paragraph.



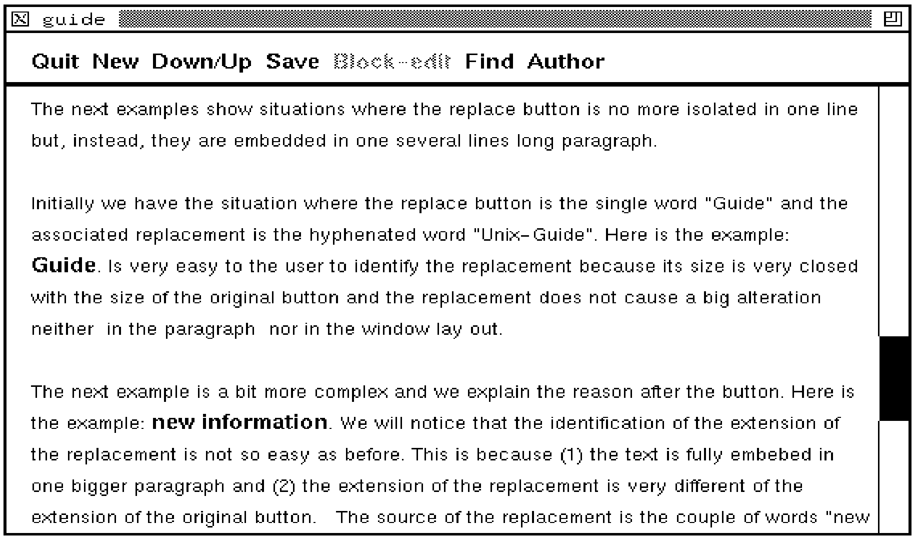
(a)



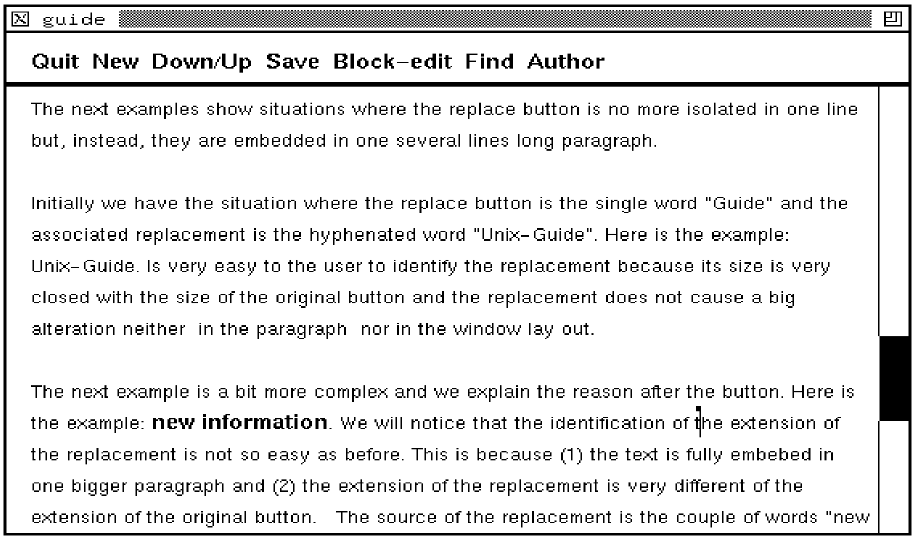
(b)

Figure 12: **Scheme** is replaced by The . . . done.

The situation now is different. The one-word source of the replacement is substituted by a full paragraph. The window contents has changed a lot because some previously existing information has been moved off the screen. However the replacement contents is still identificable because it is formed by a full isolated paragraph.



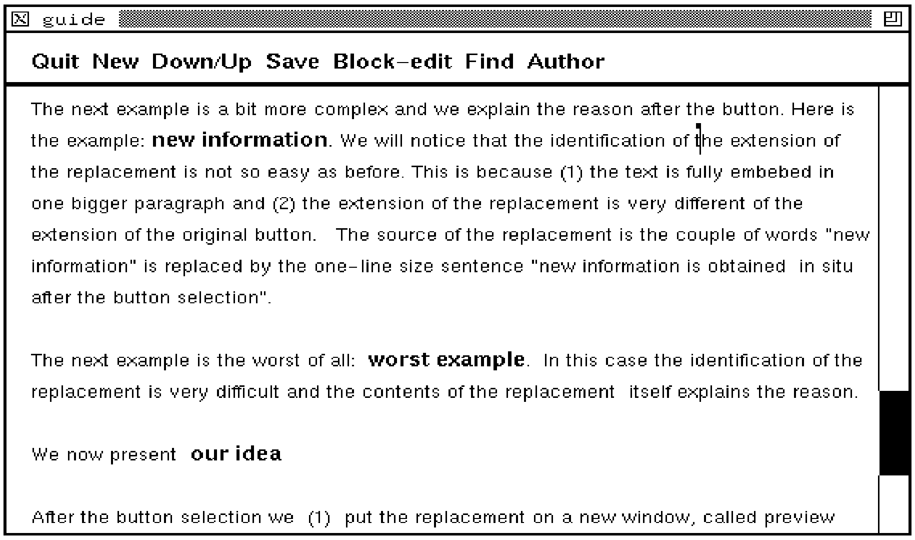
(a)



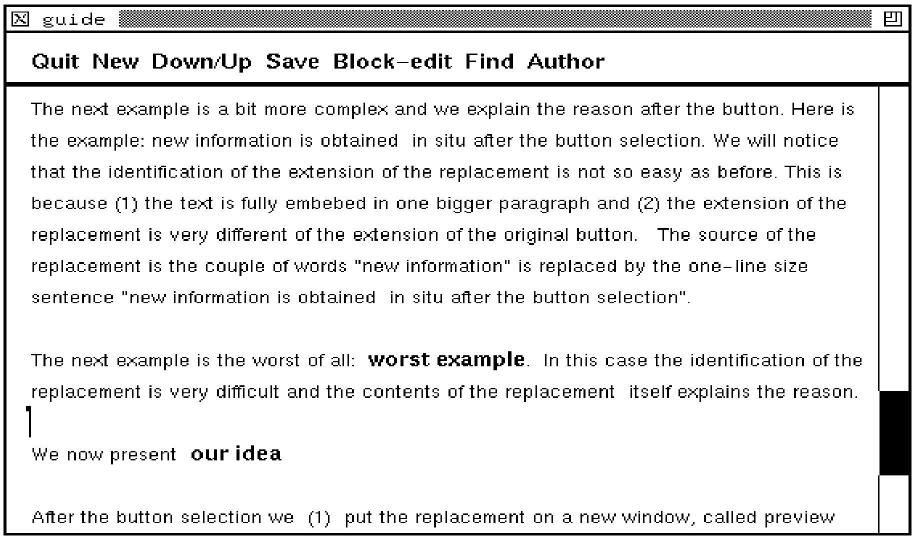
(b)

Figure 13: **Guide** is replaced by Unix-Guide

In this and the next two examples, the source button is embedded within a paragraph. This makes the replacement harder to identify than in the previous examples. In this case, however, as the size of both source and replacement text are nearly the same, the window content has not changed very much, and the user is able to identify the result after some observation.



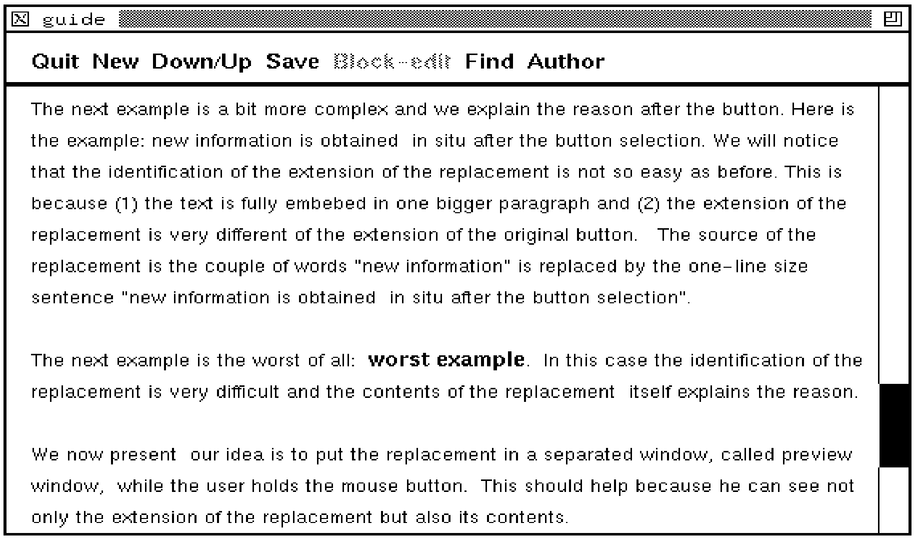
(a)



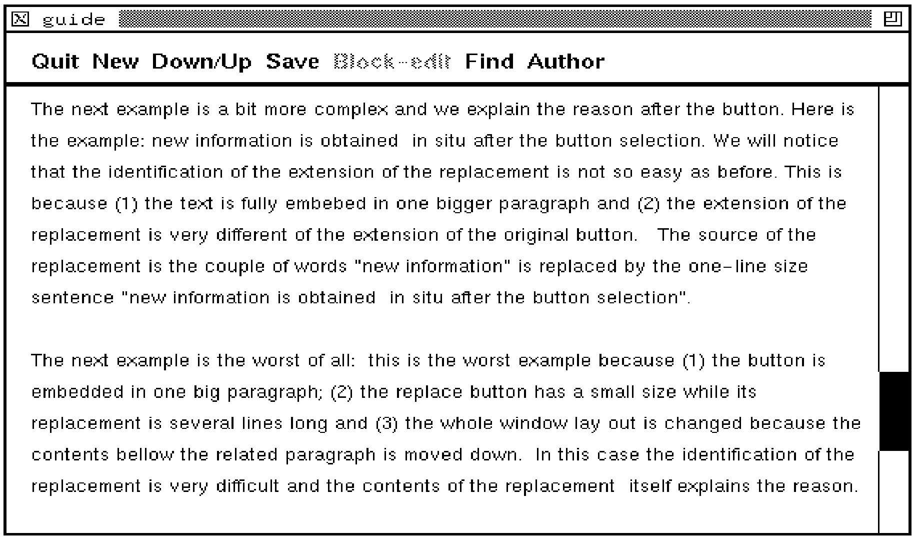
(b)

Figure 14: **new information** is replaced by new . . . selection

The situation in this example is a bit different. The window layout has not changed very much after the replacement has been finished because the replacement is shorter than one line. However, it was very much bigger that the original button (several words instead of only two words), which caused the user to be forced to look very closely to identify the changes.



(a)



(b)

Figure 15: **worst example** is replaced by this . . . down

The source button in this example is two words long while the related replacement is several lines long. Also, the source button is embedded in a paragraph. In this case the whole window information below the selected button has changed. The size of the replacement text was not easy to identify at all.

This can be argued to be poor authorship. However even if the author is oriented to place his replace buttons in isolated paragraphs, the problem of variable size of the replacement still occurs. This problem is a cause of momentary getting lost situations while navigating in a Guide hyperdocument. The next figures show some ideas to minimize the problem.

Our first proposal: the *previewing operation* approach

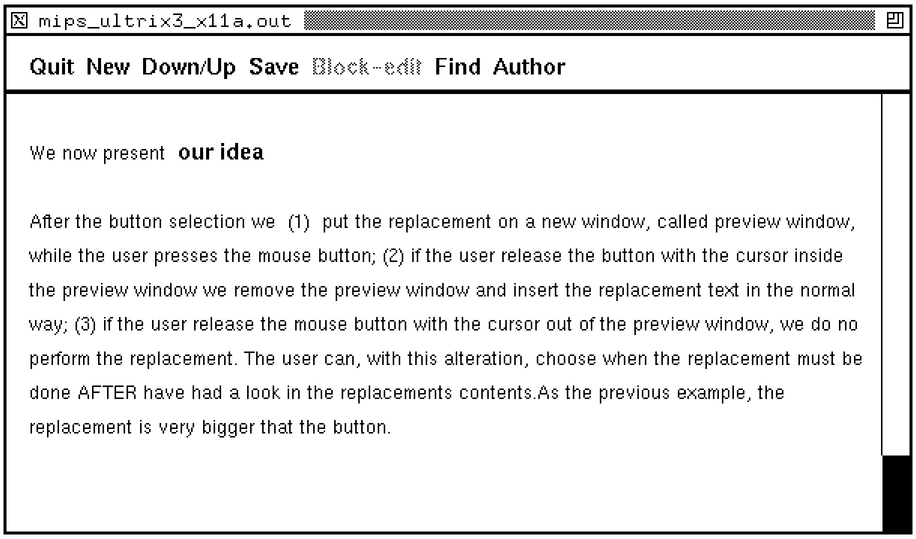


Figure 16: The user sees **our idea** and selects it . . .

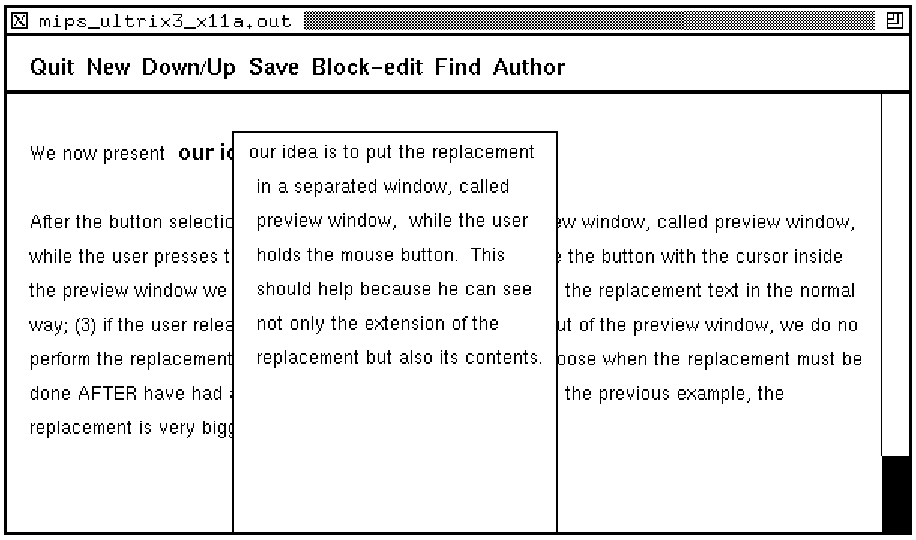


Figure 17: . . . the system shows the replacement in a preview window.

The first idea is to present the associated information in a pop-up window *while* the user presses the mouse button. In this case the user can not only see the information contents but also have idea about the size of the replacement text. Afterwards: if the user releases the mouse button when the cursor is within the preview window, the text is inserted in the ordinary way; if the user moves the cursor out of the preview window before releasing the mouse button, the pop-up window is removed and the replacement is not done.

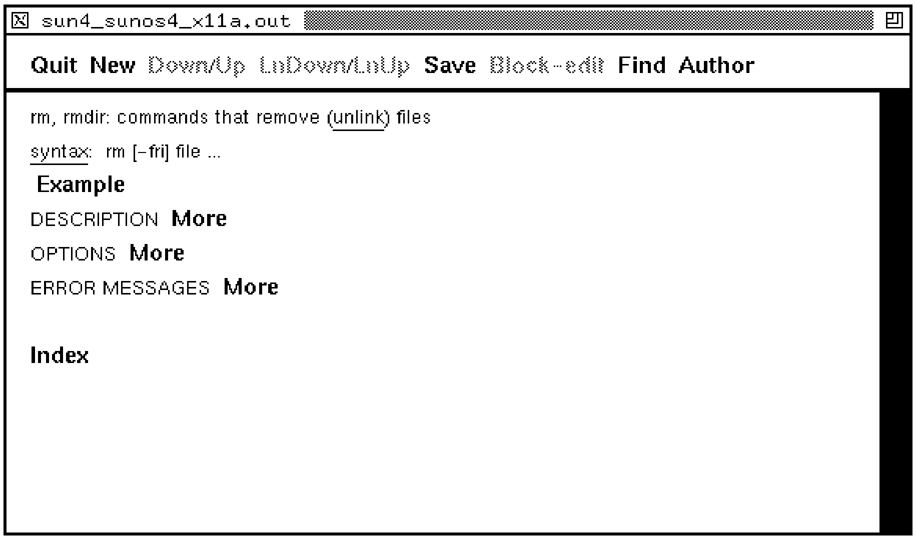


Figure 18: **More** following OPTIONS before the selection

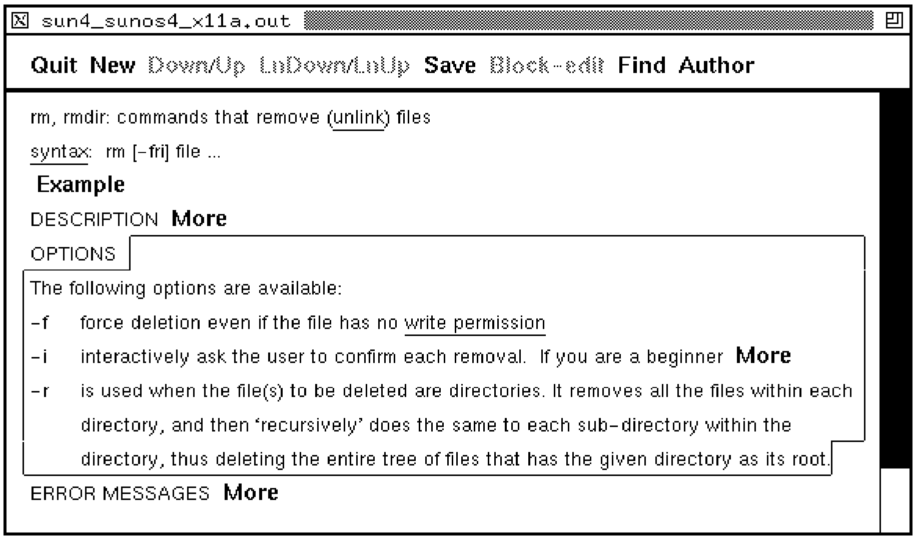


Figure 19: **More** following OPTIONS during the selection

The second idea is to insert the associated replacement in the final position and hightlight it *while* the user presses the mouse button. If the user releases the mouse button when the cursor is within the preview window, the hightlight is removed; if the user moves the cursor out of the highlighted area before releasing the mouse button, the replacement is removed.

The advantage of the second idea, chosen as the previewing operation definition, over the first is that the replacement is in the correct position if the user does not withdraw from the selection.

The next two figures illustrate one problem with the definition of the previewing operation, that always occurs when the replacement area is smaller than the source area.

The buttons in the figure below are embedeed within an enquiry. As a result, the selection of a button results on the replacement of all the enquiry source — in this case all the buttons — by the replacement text of the selected button.

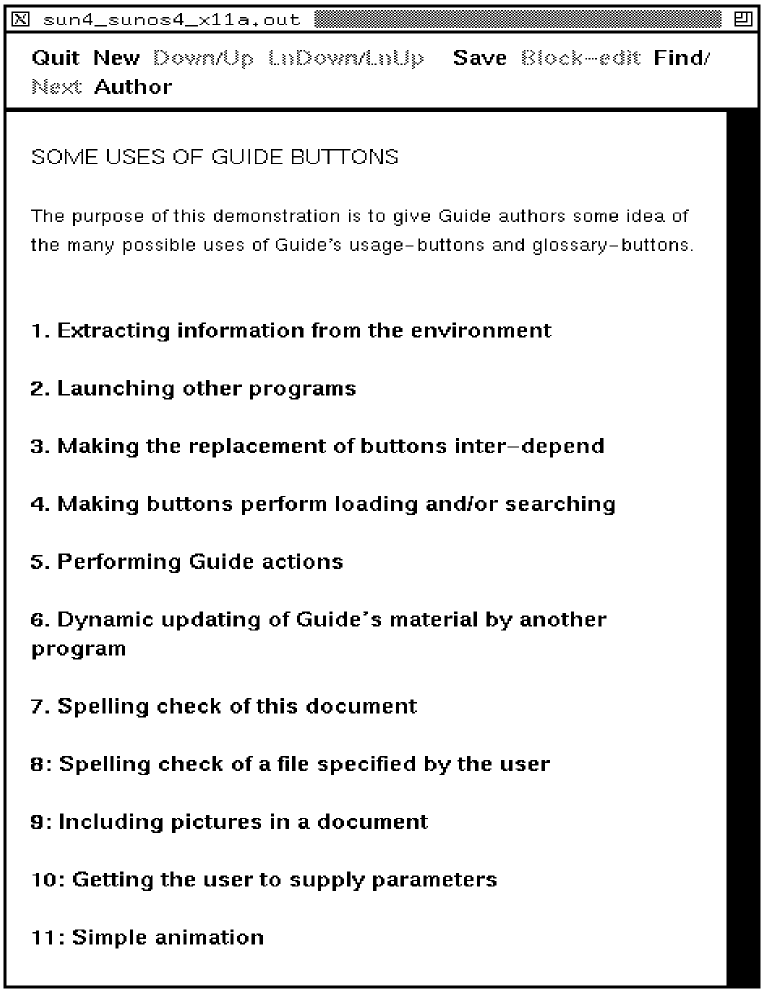


Figure 20: The buttons **11. Simple animation** will be previewed

As an example, when the user selects the button **11. Simple animation**, the mouse cursor is on the bottom of the screen. The replacement text associated, shown highlighted in the next figure, is not long enough to cause the mouse cursor being over the replacement when the user selects the button.

According to the previewing operation definition, if the user wants the replacement of a button to be mantained he simply releases the mouse button, otherwise he has to move the mouse cursor off the replacement area before releasing the button. Also: if the user just clicks the mouse button, the replacement is to be mantained.

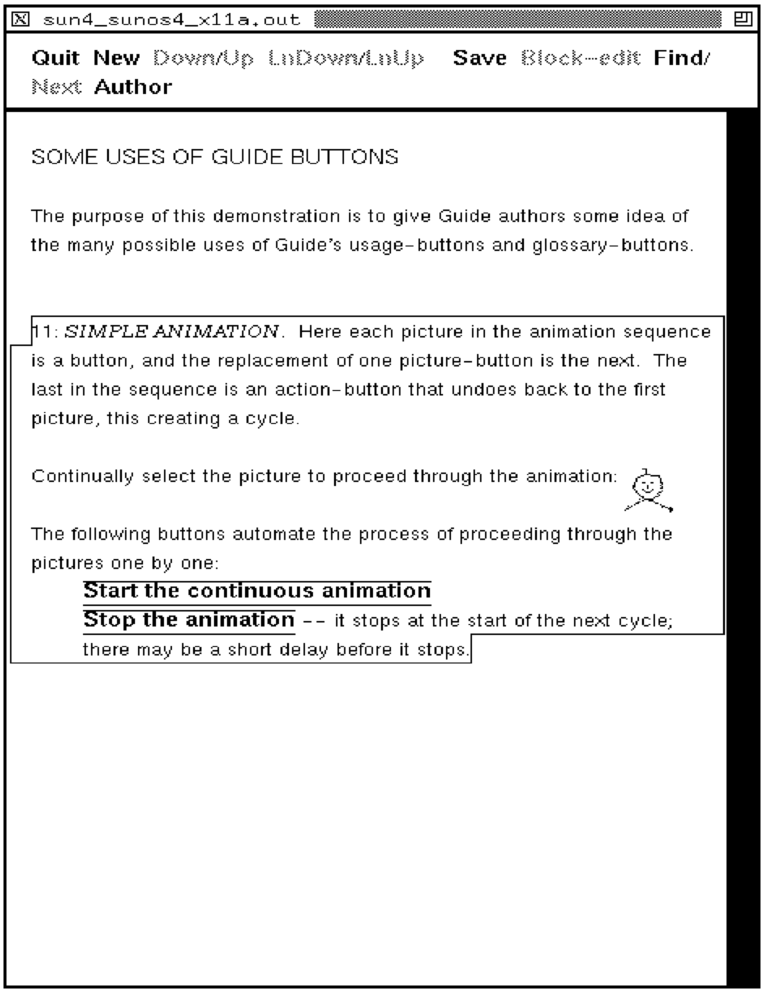


Figure 21: The previewing of the button **11. Simple animation**

As this example illustrates, the mouse cursor may be outside the replacement region when a button is selected, and if the user simply clicks the mouse button, the replacement will be undone, according to the current definition of the previewing operation. A solution has to be found, as for instance to move the mouse automaticaly over the replacement area, grabbing the cursor, or to redefine the previewing operation.

We have finished the description of the previewing operation approach and the next figures describe the *highlighted replacement approach*.

Our second proposal: the *highlighted replacement* approach

The second idea we have to help the user in his navigation is to highlight the new information *after* the selection operation has been done. Before presenting our approach we present the current approach in the figures in this page.

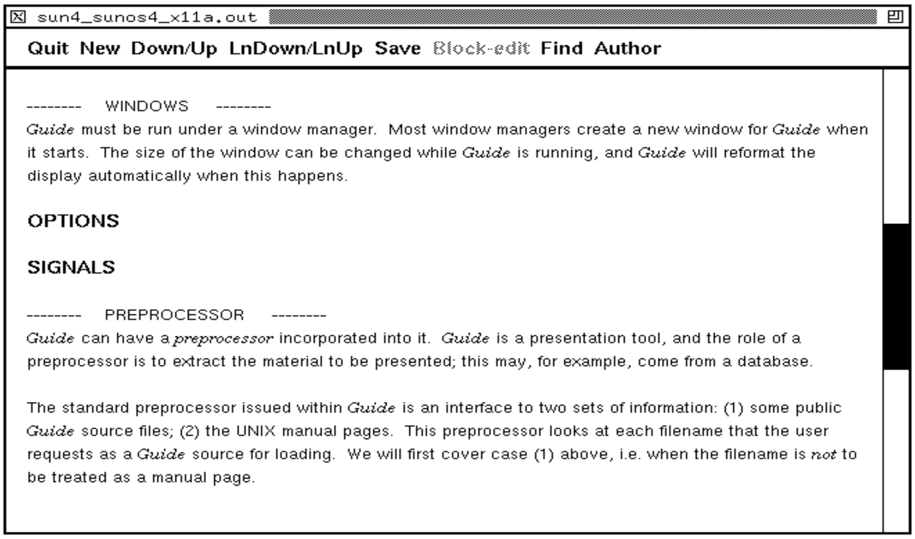


Figure 22: Situation before the selection of the button **SIGNAL**.

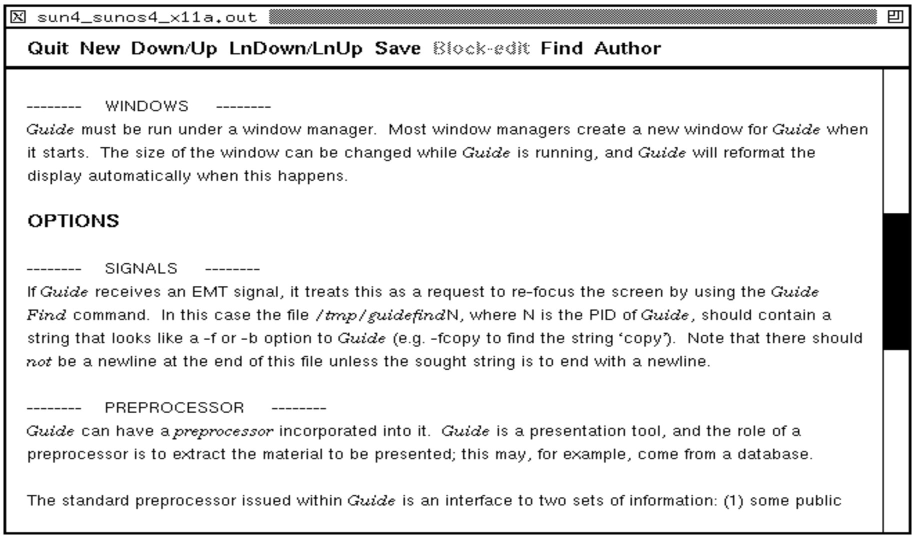


Figure 23: Situation after the selection of **SIGNAL**

The result of the selection of the button **SIGNAL** in figure 22 is shown in figure 23. There is no indication to the user what exactly is the new text. In some applications the highlight of this information could be useful. Two possible ways of highlighting the new information are presented in the next figures.

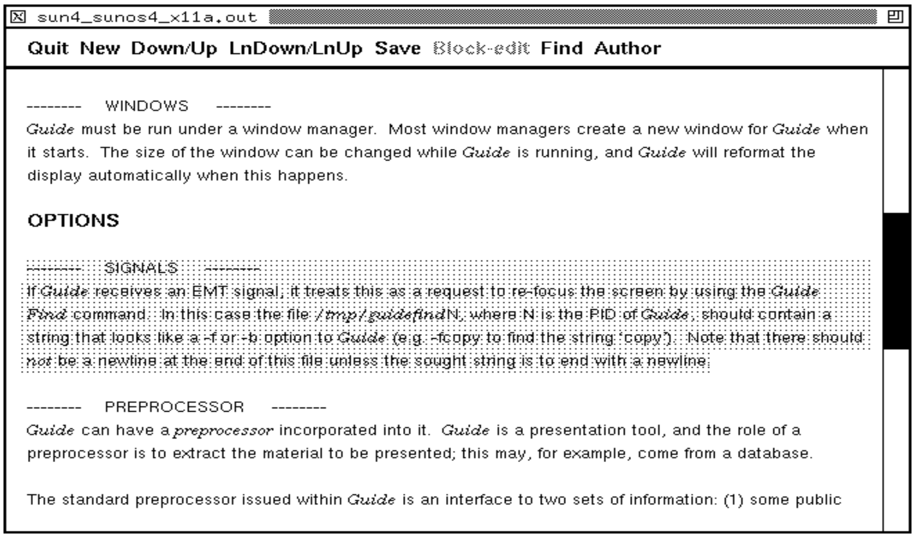


Figure 24: Highlighting the new information using a stippled background.

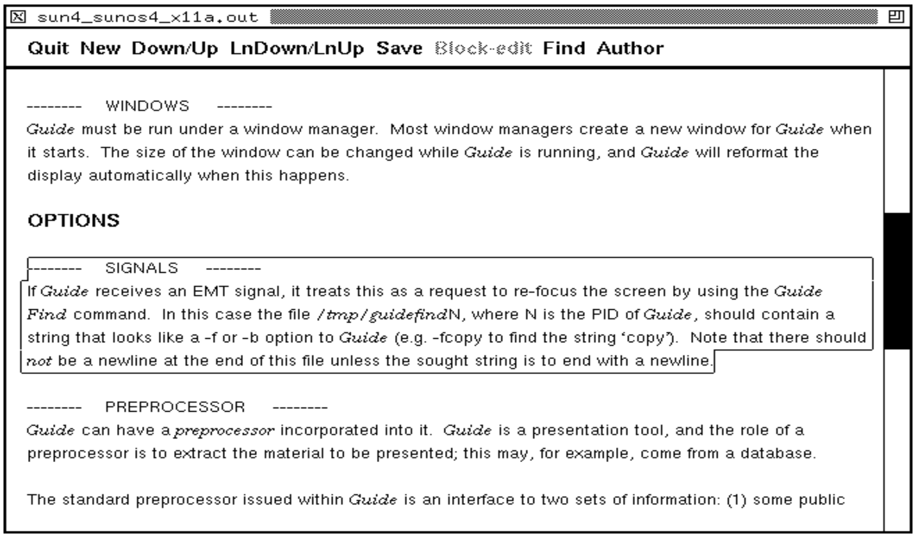


Figure 25: Highlighting the new information using a line around the text.

These are two possible ways of highlighting the new information. We prefer the second one.

References

[1] J. Conklin. Hypertext: an introduction and survey. *IEEE Computer*, 20(9):17–41, Sep 1987. Also a extended version in *A Survey of Hypertext*, MCC, STP-356-86, Austin, Tx, 1987.

[2] P. J. Brown. A Hypertext System for UNIX. *Computing Systems – USENIX*, 2(1):37–53, 1989.

[3] *GUIDE 3.0 user manual*. Office Workstations Ltd, Bellevue, WA, USA, 1988.

[4] F.A. Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31:836–852, Jul 1988.

[5] Nicole Yankelovich, Bernard J. Haan, Norman Meyrowitz, and Steven M. Drucker. Intermedia: The Concept and the Construction of a Seamless Information Environment. *IEEE Computer*, 21(1):81–96, Jan 1988.

[6] *Hypercard User's Guide*. Apple Computer, Inc., Addison-Wesley, Reading, Mass, USA, 1987.

[7] D. M. Edwards and Hardman L. 'Lost in Hyperspace':Cognitive Mapping and Navigation in a Hypertext Enviroment. In *Hypertext '88 Papers*, York, UK, 1988.

[8] R. N. Akscyn, D. M. McCracken, and E. A. Yoder. KMS: A distributed hypermedia System for managing in organizations. *Communications of the ACM*, 7:820–835, Jul 1988.

[9] B. Shneiderman and G. Kearsley. *Hypertext Hands-On!* Addison-Wesley, Reading, MA, USA, 1989.

[10] M. D. G. C. Pimentel. *Current Work on Hypertext*. Seminar presented on 01/02/1991 in the Computing Laboratory, University of Kent at Canterbury, 1991.

[11] P. J. Brown. Hypertext: dreams and reality. In H. Brown, editor, *Hypermedia/Hypertext and Object-oriented Databases*, pages 33–54. Chapman & Hall, London,UK, 1988.

[12] Barnes D., Russel M., and Wheadon M. Developing and Adapting UNIX Tools for Workstations. In *EUUG Autumn '88 Conference Proceedings*, pages 321–333, 1988.