Working Paper Series
ISSN 1170-487X

Facilitating Multiple Copy/Paste Operations

by Mark Apperley, Jay Baker, Dale Fletcher & Bill Rogers

Working Paper 99/6
May 1999

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Facilitating Multiple Copy/Paste Operations

Mark Apperley, Jay Baker, Dale Fletcher, Bill Rogers

Computer Science Department
Waikato University
Hamilton, New Zealand
www.cs.waikato.ac.nz

Abstract

Copy and paste, or cut and paste, using a clipboard or paste buffer has long been the principle facility provided to users for transferring data between and within GUI applications. We argue that this mechanism can be clumsy in circumstances where several pieces of information must be moved systematically. In two situations—extraction of data fields from unstructured data found in a directed search process, and reorganisation of computer program source text—we present alternative, more natural, user interface facilities to make the task less onerous, and to provide improved visual feedback during the operation.

For the data extraction task we introduce the Stretchable Selection Tool, a semi-transparent overlay augmenting the mouse pointer to automate paste operations and provide information to prompt the user. We describe a prototype implementation that functions in a collaborative software environment, allowing users to cooperate on a multiple copy/paste operation. For text reorganisation, we present an extension to Emacs, providing similar functionality, but without the collaborative features.

Keywords: Copy and Paste, Cut and Paste, Paste, Multiple Selection, Collaborative, Transparent Overlay, Augmented Pointer.

1. Introduction

A copy (or cut) and paste facility has been a part of graphic user interfaces since their inception, providing the principal mechanism by which the user can move data within and between documents. When there is only a single piece of data to move, such as a bitmap or a segment of text, copy and paste works reasonably well. The user selects the information; issues the copy command; moves focus to the required destination application and document; and issues the paste command. The only drawback is the fact that the clipboard (paste buffer) is usually invisible. If the entire action can be completed quickly, this is not important (except perhaps for novice users). If there is some delay between copy and paste, say, a need to edit the destination document in readiness for receiving the cutting, then difficulties can arise. It is often all too easy to inadvertently issue another copy command and overwrite the (invisible) information waiting in the buffer.

When there is a need to copy several related pieces of data, however, the conventional copy/paste sequence of actions can require a great deal of mouse movement. This is illustrated in Figure 1. The fields of the dialog box on the left are to be filled with information selected from the web page in the browser window on the right. Two of the pieces of data required are on immediately adjacent lines, and the other two are accessible with a simple scroll action. Yet a user must repeatedly shift between source and destination windows to move items one by one. They are unable to take advantage of the proximity of source data items. The movement required is greatly increased when copying data between overlapped windows, or from one place to another in a long document.

The conventional copy and paste model assumes that a user should first select an object, and only then think about where it might be placed. Sometimes it would seem more natural to issue the copy and paste commands in the reverse order. In Figure 1, the user starts with the need to fill the dialog box and then goes to the web page. If the paste command can be issued first, the system has enough information to keep a prompt before the user during their search for data, and also possibly to allow parsing or validation as the information is copied. Where there are several fields to fill, there is the possibility of starting with a multiple 'paste'.

In this paper we look at mechanisms for multiple copy/paste in two contexts. The first is a project, Collaborative Information Gathering [2] in which we are developing software to facilitate collection of data from the World Wide Web and other sources. The second is editing the text of computer programs using the Emacs editor. One demonstrates a between document, between application software, copy/paste problem. The other demonstrates the problem in the context of a single application, holding one or more, possibly large, documents. We have developed different prototype solutions for the two situations, both providing for

multiple data transfers. They will be described in the following sections.

2. Related Work

Selection, copy and paste operations are a fundamental part of the graphic style of user interface. In the simplest form of selection a user can select an object by clicking on it. Most systems also make provision for selecting several items at a time. Either the user holds down a modifier key while clicking on objects to add them to the selected set, or outlines an area to select everything inside. These capabilities are illustrated by the file selector used in Windows 95 applications [6].

This kind of multiple selection does not however, meet our need for multiple copy and paste. The intention is always to treat the selected items as a group. They will be pasted on an all or none basis. The actual cut and paste commands are usually implemented so as to provide quick and convenient use. Generally a menu form is available for novice users, with keyboard shortcuts or mouse gestures providing quick operation for experienced users.

A copy and paste system that comes close to meeting our needs is that provided by the GNU Emacs text editor [7]. The paste buffer holds a history of copied text selections. The 'select and paste' menu option displays a sub-menu of the most recent entries. Any can be pasted into the document. This makes it possible to copy a series of fragments and then perform a series of pastes in some different document or area of the same document, without the need for focus flipping. Because of the limited space available when displaying the sub-menu however, only a tiny amount of text can be displayed, and it can be difficult to recognise fragments. Further, Emacs stores all deleted text in the paste buffer. There is no distinction made between fragments deliberately cut or copied, and text that is simply being removed. As a result the paste list tends to be somewhat unpredictable, containing apparently spurious entries.

In our Collaborative Information Gathering context we wanted to provide good graphic feedback to the user during copy/paste operations, without using a great deal of screen area. This meant that some kind of screen overlay method would be required. Many other systems have used overlay techniques to add information to displays. The mouse pointer itself is an example of an overlay. In Groupkit [3], Greenberg implements semantic telepointers in a CSCW environment. Here, multiple mouse pointers are augmented with collaboration awareness indicators. This is an excellent way of providing information with minimal occlusion of important screen contents. The mouse pointer is usually beneath and to the right of the point at which the user's attention is focused. If it does cover something important, the user can move it immediately.

Transparent overlays have been used in a number of systems. In "See-through Interface Software" [1], tools for graphic editing are depicted lying on a transparent 'surface' overlaying the image being edited. A user can move the tool surface using their non-dominant hand, aligning a tool over the area of the underlying document to which it is to be applied. The dominant hand is then used to select the precise point of application. The problem of keeping the tools from obscuring the document is addressed by drawing the tools quite large, in outline form, leaving useful areas of the underlying image visible within their borders. Enough of the image is visible to provide context for final alignment. The lines used are thin, to minimise the area obscured.

Kramer addresses the issue of distinguishing the information on an overlay from that in the underlying document in his Translucent Patches [4]. Difference in appearance is used, for example hand drawn annotations over a typeset document. He points out that this is just the distinction that we use to recognise editing markup on a manuscript. The same issue is addressed by Lyons [5], who uses slight transparency and high saturation colour on overlaid menus to enhance the impression of their 'being in front'.

Finally, in our environment, we have the problem of tool interaction. We wish to extract information from, and will later want to put information into, existing application programs for which we have no access to source code. There are several standard systems used to interface tools; CORBA, COM, Java Beans, etc. The programs we are using have COM interfaces [6]. This kind of interface allows dynamic connection to a tool and whatever level of control the tool designers chose to export, usually access to the information held and notification of changes made. The available interfaces don't provide detailed information about user interface events. They are designed to provide an independent way of operating the application, but leave the application in control of display appearance and user interaction. Our system requires something more like the kind of software interface proposed by Bier to allow See-through Tools to operate in an application independent way.

3. Collaborative Information Gathering

The Collaborative Information Gathering project [2] is developing software to facilitate the work of people collaborating to build and use documents from information found and extracted from the world wide web, databases or other electronic information sources.

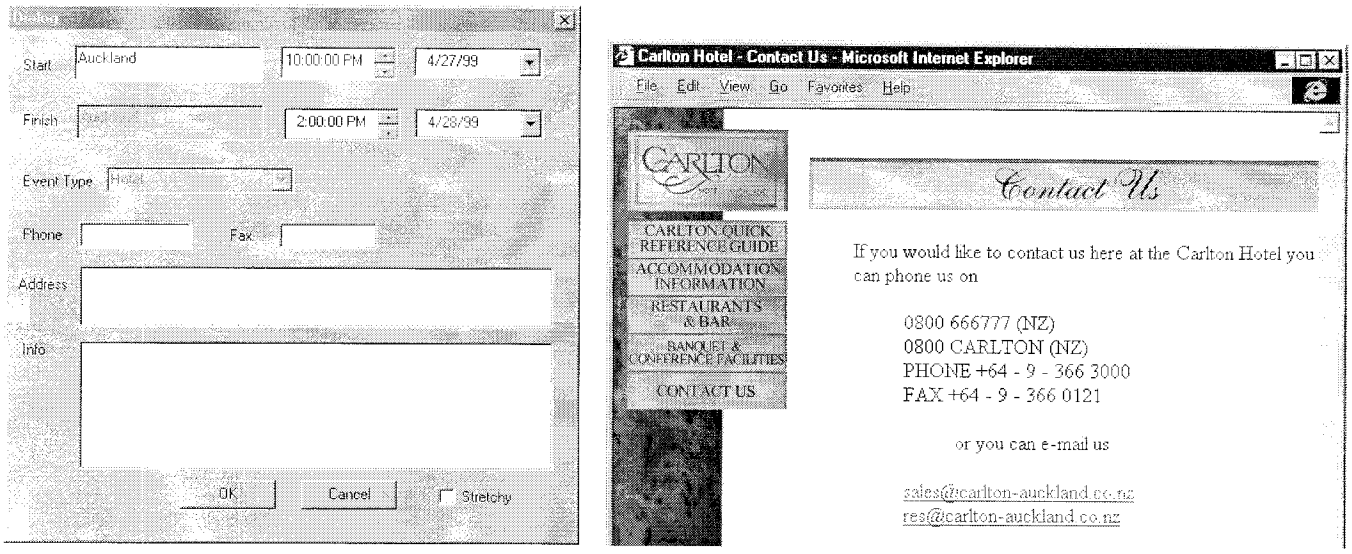

Fig 1: Dialog box and web page holding required data

In our first prototype application, the document developed is a travel itinerary, generated in collaboration by a consultant and their client(s).

Our software can display an itinerary in a number of forms, some text and some graphical. We also envision it taking programmed actions, like making airline or hotel bookings. A crucial part of the information gathering process is therefore the extraction of data in structured form, on which to base visualisations and actions, from the free form text found in sources such as web pages. We are interested in developing ways of automating some information extraction, but the ill structured nature of the web and the huge variety of ways in which information can be presented on web pages strongly suggest to us that manual extraction of data will often be the only possibility. Certainly, the simplest way of structuring data is to have one or more of the users of the program manually extract fields. Hence the multiple copy/paste operation.

Consider the following scenario. An itinerary being developed includes a visit to a city (Auckland, New Zealand), and a hotel booking is required. The consultant enters a 'hotel stay' event and the system pops up a dialog box requesting details on the hotel. In the completed system it will be possible to satisfy the information need in a number of ways. The system might already have data on a number of hotels, and the client might simply select one. The case of interest though, is where the system has no access to suitable structured information.

We revert to searching an unstructured source, for example, the World Wide Web. The task of locating a hotel falls to the consultant and client, who must find the information required to fill the dialog box. They are setting about a directed search process. The system 'knows' what they are looking for, but neither where to find it, nor how to extract it in a useful form from the unstructured documents that will be the initial result of the search. We assume that appropriate web resources are used, and a web page or pages describing a suitable hotel are found. The system reaches the situation shown in figure 1. Using traditional copy and paste, the user must:

For each field in the dialog box
  Focus on the web page
  Find and select appropriate text
  Issue a copy command
  Focus on dialog box destination field
  Issue a paste command

At best this involves repeated long mouse movements, at worst it may involve switching between overlapped windows. If the dialog box cannot be kept visible during web browsing, the user must also remember which datum is required on each transfer. Although the system 'knows' about the data required, it does not prompt the user, nor does it automatically accept the data.

3.1 The Stretchable Selection Tool

Our solution is the Stretchable Selection Tool (SST). It is associated with a dialog box or other information sink. When an SST is activated, a paste operation into the first field of the information sink is 'armed', ready to happen automatically as soon as the user identifies the source data and triggers the SST. If unfilled fields remain, the SST will 'rearm' after firing, ready to paste data into the second field of the sink, and so on. Because this is a modal facility, it is vital that the user interface makes it clear to the user just what action is pending. We do this by connecting a visible band from the dialog box field to the mouse cursor, and placing a short text prompt below the end of the band, near the cursor (fig 2).

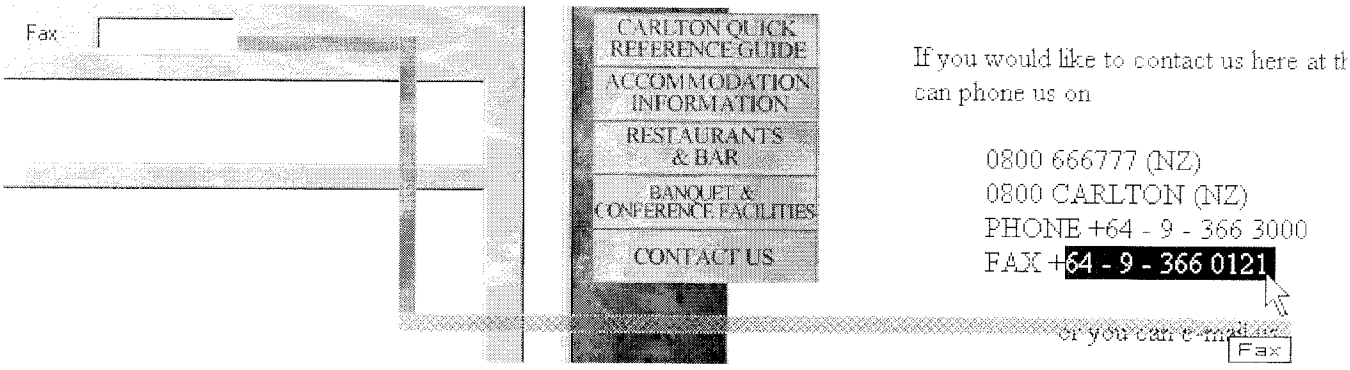

Fig 2: Stretchable Selection Tool connecting 'Fax' field to cursor.

The visual metaphor is of a flexible pipe, one end connected to the sink field, the other dragged about with the cursor until connected to a suitable 'tank' from which data can be drawn. Ideally the 'pipe' would snake about like a hose pipe lying on the ground, but implementation constraints have forced us to implement it with horizontal and vertical segments.

In the trial implementation the SST can be switched on and off using the tick box at the lower right of the dialog box (Fig 1). When the SST is active, the pipe and prompt serve to remind the user that they should be trying to find a particular piece of data. In effect activating the SST constitutes the issue of a multiple paste command. It is appropriate to issue the paste before finding the data, because that is the order natural order in which the user addresses the problem.

The SST should not cause any change in the way in which applications behave, except that one command will have been stolen to 'fire' the SST. In the prototype we have used the right mouse click for this purpose. Otherwise the user is free to operate the web browser or other application as normal to find and select the required information.

The crucial point is that the user does not need to shift their mouse pointer, or their attention, between windows. If, as in the example shown, the required data is close to that first selected, there is a very considerable reduction in total mouse movement. The user just left-clicks, drags, and right-clicks on successive pieces of data.

It is possible that the data required to fill a subsequent field lies some distance away from the first data set, or on a different web page, or even somewhere accessed by a different application. The SST permits access to the normal browser controls. The user can scroll, or link to another page before selecting data. In such cases the gain in terms of reduced mouse movement may not be so dramatic, but in our view the user still benefits considerably by not having to switch attention back and forth to the dialog box. They are also continually reminded of their goal.

In our screen shots we have shown the case where the dialog box and the web browser windows fit side by side, without overlap, on the screen. This is the case for which our prototype has been developed. However, the system is also workable when the dialog box is out of sight. In such cases the 'pipe' comes in from the left edge of the screen, and the prompt field identifies the required information.

It is possible that a user may be unable to fill some field, may want to fill fields in a different order to that specified in the dialog box's natural tab order, or may want to make corrections. The SST destination can be relocated by clicking in a dialog box field, but that involves mouse movement to and from the dialog box. The alternative is to use non-dominant hand keyboard actions to cycle through the fields.

The travel itinerary program supports concurrent collaborating users, so that copies of the dialog box with SST attached may be open simultaneously on two or more users' screens. We permit each user to activate the SST feature, and to use it to add data to the dialog box. For the example shown it is unlikely that this would be a useful, but in a larger example, with more fields to be filled, it would make sense for people to share the task of data collection. We have adopted a weak synchronisation model in that fields are not locked against simultaneous access. Users will usually have an audio connection, and can resolve conflicts themselves.

We do provide some visual indication of competing activity, however. When there is simultaneous activation by several users, it is desirable that each user should know which of the fields each of the others is focused on, whether it be for text entry or as an SST destination. The system associates a colour with each collaborating user. All active fields are outlined on each view with the colour identifying the user accessing the field. The system displays only one colour on each field, but this is chosen to be one other than the colour of the user on the screen concerned. The user is therefore warned that they are competing over a field by not seeing their own colour

surrounding their active field. Competing data fill requests are actioned in their order of arrival at a user interface server. The server immediately dispatches updates to each display, so a user will quickly see if their data gets overwritten.

3.2 Design Issues

The band representing the data conduit is displayed on the screen overlaying the application from which the data is being extracted, and possibly parts of other windows as well. This presents implementation difficulties that are discussed in the following section. It also presents design difficulties. There are conflicting visibility requirements. The band must be easy to see and clearly distinguishable from the material below, but it must not obscure the underlying display excessively.

Following Greenberg and Bier, we attach the band to the mouse cursor and space its graphics sufficiently below the pointer to leave the context of the parts of the underlying document near the cursor clearly visible. The pointer itself lies mostly to the right of the active spot. Good spacing becomes impossible only when the cursor reaches the bottom of the screen, when we allow it to overlap the band in order to keep the prompt visible. This is not normally a problem as the source document is rarely extends to the bottom of the screen.

To make the band easily distinguishable from the background text, we began with strategies of Lyons and Kramer. First, that it should be drawn with high saturation colour. This idea wasn't appropriate in our situation. High saturation colour focuses attention, and usually serves to identify high priority items. We wanted the band to be distinctive, so that it was separable from the underlying document, but not especially attention grabbing. Instead, we found that primary colours in low saturation, pink and pale blue, served our purpose better. These were colours that didn't occur on our test documents, and so were distinguishable, yet had similar brightness to the document and were not obtrusive. The effect is like using pale highlighter, different enough to ignore when reading a document, but still easy to see. The only disadvantage is that the lifting effect of high saturation colours were lost.

Kramer's principle of using a different style was followed by making the bands wide, again in a highlighter style. The underlying material is usually small text with fine rectilinear lines. We would have liked to draw our band with curves, to give a hand drawn appearance, but implementation difficulty restricted us to horizontal and vertical bars.

A number of options for painting the bands were explored. Our first idea was to simulate highlighting exactly, by blending a colour with the underlying text. This proved to be technically infeasible. It would also have had the disadvantage of visually blending the band into the document, where we really wanted it to look elevated. We then experimented with three different overlay styles (fig 3). They were solid colour, hollow outline, and pixel checkerboard. Solid colour, of course, obscured the underlying text. The hollow outline was satisfactory and gave the best view of the underlying document. The checkerboard arrangement gave the best visibility, and a satisfactory view of the background. Whilst it might not always be clear enough to read through the band, visual continuity of the underlying text was maintained.

We also experimented with animating the passage of text along the band. It looked cute, reinforcing the pipe interpretation, but introduced irritating delays.

3.3 Implementation

Our implementation is in C++, using Microsoft Foundation Classes, under Windows NT. The choice of platform was dictated by the application. Had we just been experimenting with SST software, it would have been more convenient to use a system that gave us the freedom to reprogram its window manager. Without that option, we found ourselves trying to make the windowing software do things it was not designed to allow.

There were two technical problems to solve. One was to be able to track the mouse while other application software was running. The reason was to monitor the event stream of the application that was being used as a data source, in order to know when selection occurred and to capture the SST fire command. The second problem was to draw the SST band over other application windows without disrupting their screen update process.

Event monitoring was achieved using the 'transparent window' facility. The Windows NT library allows a window to be declared transparent. It receives user interface events over its screen area, but passes through

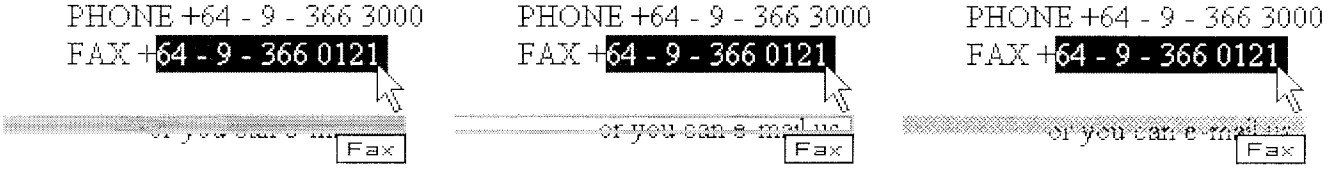

Fig 3: Options for painting the bands (solid colour, hollow outline, pixel checkerboard).

repaint events to the windows beneath, allowing those windows to remain visible. Our system uses such a window to catch events and pass filtered versions to the application below. This was not as simple as we had expected because the window manager doesn't just pass raw mouse events to windows. It checks to see what part of the window is being accessed, and in many circumstances it sends a more specific message. For example, some scroll bars are operated in this way. The application window doesn't see a mouse event, it just sees a scroll bar event. The effect was that objects could be selected, but that many application controls would not function. We resolved the problem by carefully fitting the transparent window over just the client area of the application window, leaving control areas free for direct operation. This means that scroll bars and menus work correctly, but leaves difficulties when windows are moved or resized, or when the user shifts between applications. There is also come confusion with window focus. The first click of a click and drag selection may be interpreted as moving focus to the window, and ignored as a click. These problems have not been fully resolved, but the current system is moderately robust.

Drawing bands over the screen also proved challenging. Our first attempt was again based on using a transparent window. We were unable to make that work, and instead turned to actually building the band from windows. Windows NT provides only rectangular windows, but it does have a facility for putting rectangular holes in a window, through which underlying windows can be seen. Using this 'region' facility, a rectangular window can in fact be cut to any required shape. For our application a window large enough to encompass the whole band proved too slow, so we use three separate windows for the three parts of the band (fig 2). The part on the dialog box changes only rarely. The vertical segment just grows and shrinks in height. Only the segment leading to the mouse cursor moves substantially.

We used the region facility to achieve different degrees of transparency. Cutting out the entire interior generates the hollow bands. The checkerboard band is made by clipping out alternating one-by-one pixel regions. The result is very demanding on processor time during update, but provides the best appearance. The demand on processing resource seems to be as much due to problems with redrawing priority as it is to do with intrinsic computational complexity. Normally, when a window is being moved, the user's attention is primarily on the window, and on the background at its leading edge. The screen will most rapidly provide the needed information if redrawing the window in motion is the first priority. Some slowly repaired mess at the trailing edges will not be a problem. In our application the user is not focusing on the moving window at all. Rather they are looking at the underlying windows. Unfortunately the areas of interest are those that are updated with lowest priority. The effect is amplified by the use of regions. The window with holes in seems to be updated quite efficiently. Underlying applications can be quite slow in contributing the image areas that should be visible through the holes. Again, the implementation is far from perfect, but works well enough to demonstrate the concepts.

There were many options for implementing the 'copy and paste' command. The application we have experimented with most often, Internet Explorer, has a COM interface through which selected data might be extracted. In fact, we have chosen to simulate a 'copy' menu selection on the grounds that the event used is common to a large number of applications. The SST software then takes the clipped text from the system clipboard and puts it into the dialog box field. There are also a number of alternative ways in which we might have implemented the 'fire' command. One possibility would have been to simply allow the user to clip using the normal command of the source application and monitor the system clipboard for incoming data. We rejected this option for two reasons. One was that applications are not uniform in providing short cut keys or mouse gestures for cutting and we didn't want the user to have to use a menu, with its attendant large mouse movements. The other, and more important reason was that we couldn't assume that all copy events were intended to deliver data to our system. The user might, for example, want to copy and paste a URL as part of their searching process.

4. Multiple Copy and Paste in Emacs

Modern programming involves not so much knowing how to work with a programming language, but how to work with a variety of libraries and application programmer interfaces. A typical programming session alternates between searching through documentation and reference material on the web, and writing program instructions. It is frequently the case that documentation is, or seems, ambiguous. The only resort in such cases is to write test code and see just what does happen. It is often best to write isolated test programs, rather than directly try a new feature in the context of the program under development. Sometimes documentation comes directly in the form of a demonstration program. Once the test or demonstration program is understood and functioning, there is a program reorganisation task. Bits of the test program - declarations, initialisation, event handling - must be extracted and merged into appropriate places in the target program.

Pasting is rarely a simple process. It normally involves making some changes to the incoming text. If sections are being moved one at a time, it is almost impossible to

maintain any continuity of attention to the source program, and the likelihood of sections being missed is very high. As an editing operation we consider that it is most convenient and reliable to work through the source program extracting the required sections, and then to work through the destination program pasting them into place. The programmer will work best if they are not continually changing focus, and if movement commands are kept to a minimum.

Of course there is always a trade-off between learning new features and simply using less efficient sequences of commands. We argue that the program reorganisation task is so common, and in our experience so poorly supported by existing editors, that we are justified in providing special commands.

Program reorganisation is another example of multiple copy and paste. It differs from the CIG example in that there is no advance knowledge of the data segments required. The conventional copy then paste order is appropriate, the user wants to collect some pieces of data, and then paste them by hand later. The SST mechanism would do half of the task. It would allow us to collect pieces of data. It does not provide a way of pasting.

Rather than an elaborate graphic solution, we built a multiple copy and paste mechanism using the macro capabilities of Emacs. Our system provides four paste buffers. They can be left invisible as is usual for paste buffers, or they can be displayed as subwindows (fig 4).

In normal operation a right mouse click will cut selected text from a document. In our system, an extra click will put the text into the next free paste buffer. Note that ordinary deletion of text does not put information into the special buffers. Note also that ordinary operation of the editor is not changed. The four paste buffers are associated with the function keys F5 through F8 and are named to remind the user of this fact (fig 4). These keys serve as paste commands.

The system allows multiple copy and paste (limited to four buffers). We have used it successfully for program modification and find considerable benefit from the reduced need for context swapping and navigation.

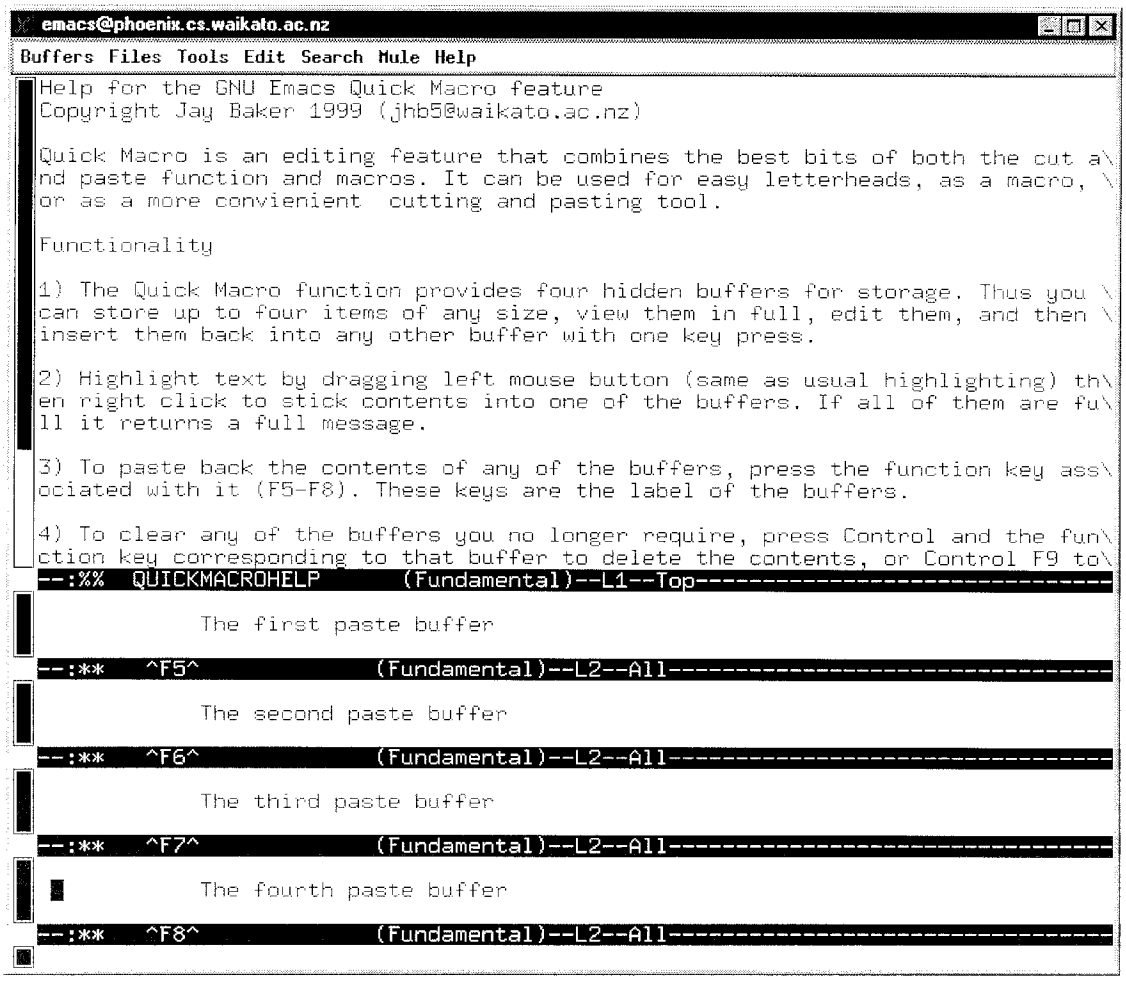

Fig 4: Emacs screen showing paste buffers as subwindows.

5. Conclusions and Future Directions

We have argued that multiple copy and paste operations tend to be poorly supported in graphic user interfaces and that this is of importance to users because it can interpose annoying and time consuming navigation tasks between naturally related operations. We have the identified the problem and prototyped solutions in two different domains.

In the directed collection of data domain we provide a graphic user interface widget, the Stretchable Selection Tool, based on the visual metaphor of an 'information pipe'. The prototype implementation forms part of a larger project and as such is not yet ready for user testing. Even the SST code needs further work to be made robust enough for user testing. Nevertheless, our experience in putting sample data into our application has been positive. The reduction in navigation overhead when collecting data seems worthwhile.

For our larger project, further development of the SST is required. The intention of the project is to be able to put data into as well as take data out of weakly structured formats. To do this, the SST receiver will need to take the form of an overlay table on the receiving document. In fact it is because of this requirement that we persevered with the 'pipe' graphics. The existing system would be workable without a 'pipe', having the text prompt near the cursor to indicate the data requirement. As the system develops, there will be situations in which no suitable prompt text will be available, and the graphics connection from source to destination will be all that the user has for guidance.

In the text editing domain we provide a set of macros for Emacs to allow copy and paste of up to four items at a time. These macros are available for downloading from our web site (www.cs.waikato.ac.nz) and have also been posted to the Emacs news group.

References

1. Eric A. Bier, Maureen C. Stone, Ken Fishkin, William Buxton, Thomas Baudel. A Taxonomy of See-Through Tools. *Proceedings of CHI 94*, pp 358-364. (1994)
2. Collaborative Information Gathering. http://www.cs.waikato.ac.nz/cs/Research/cig/
3. Saul Greenberg, Carl Gutwin, Mark Roseman. Semantic Telepointers for Groupware. *OzCHI 96 Proceedings*, pp 54-61. (1996)
4. Axel Kramer. Translucent Patches – Dissolving Windows. *UIST 94 Proceedings*, pp 121-130. (1994)
5. P. J. Lyons, M. Pitchforth, D. Page, T. Given, M. D.Apperley. The Oval Menu – Evolution and Evaluation of a Widget, *OzCHI 96 Proceedings*, pp 252-259. (1996)
6. Microsoft Corporation, Windows 95 and NT Programmers Reference, *Online documentation with Visual Studio compiler*. (1995...)
7. Richard Stallman, The GNU Emacs editor, http://www.gnu.org