For: jan
Printed on: Tue, Nov 17, 1992 15:56:50

Document: xmfm2
Last saved on: Tue, Nov 17, 1992 15:56:40

XmFm – An X/Motif File Manager

Jan Newmarch
Faculty of Information Sciences
and Engineering
University of Canberra

Introduction

`xmfm` is a file manager for Unix workstations that uses the Motif graphical user interface. Like other file managers it displays directory contents, using icons for files. The display organises files into groups of directories, executable files (programs) and other files (data files) to aid in finding appropriate files quickly. It can display multiple directories, each appearing as an identical display without requirement of a ‘‘root’’ directory display (unlike the OpenLook file manager). It supports drag and drop actions using the features of Motif 1.2, and can interact with other applications using the same protocol. Its major differences with other file managers are that it supports an object–oriented approach to the actions that can be performed upon files, and the high degree of user–level configurability of the system. This paper describes `xmfm` and compares it to some other file managers.

Display

Like most other file managers, `xmfm` gives a graphical display fo the files in a directory. The display of files for a directory is divided into three areas (see Figure 1). This gives an area for directories, an area for executable files (programs) and an area

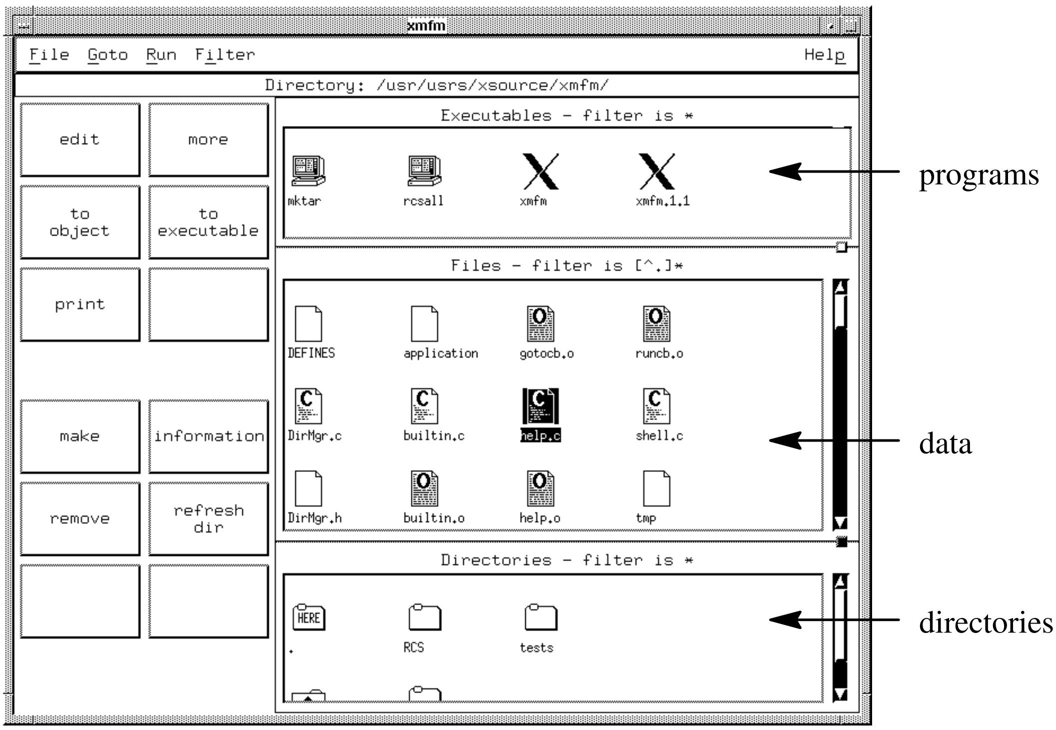

Figure 1: xmfm

for data files. These three categories are functionally quite distinct: programs are normally run (or sometimes debugged!), while directories are opened or moved into, and other files generally have no common type of action. It is also easy in Unix to distinguish these three types (directories have the directory bit set, programs have the executable bit set and the data files have neither). Unix also distinguishes other file categories (such as devices) but these are less common and probably not worth distinction at this level. In addition to using three areas, the areas are actually resizable panes and if a pane is too small to show all of its contents it adds a scroll–bar to allow movement through each file list. This display allows the user to quickly find files of each type without having to disturb their location within the other panes.

A practical illustration of this is the edit/compile/execute cycle of a program. A source file is edited, and will be compiled to produce an executable file often with a very different name to the source (the `cc` compiler will, by default, create the executable as `a.out`, no matter what the source file. `xmfm` makes the task of switching between source files and programs just a mouse click between separate panes.

By comparison, other file managers do not give this separation. The manager `xdtm`, (see Figure 2) shows just one area, so that moving between different file types is not easy. To move between data files and programs may require much use of the scroll bar. Looking Glass (Figure 3) shows two areas only, one for directories and one for files. It also shows the directories in the upper pane. This is where the user would focus attention most often, but the directories probably need attention less often than other files. `xmfm` places the programs at the top, other files in the middle, and directories at the bottom, to allow the focus on the most important objects to be near the top. The OpenLook `filemgr` (Figure 4) places all icons in the same area, like

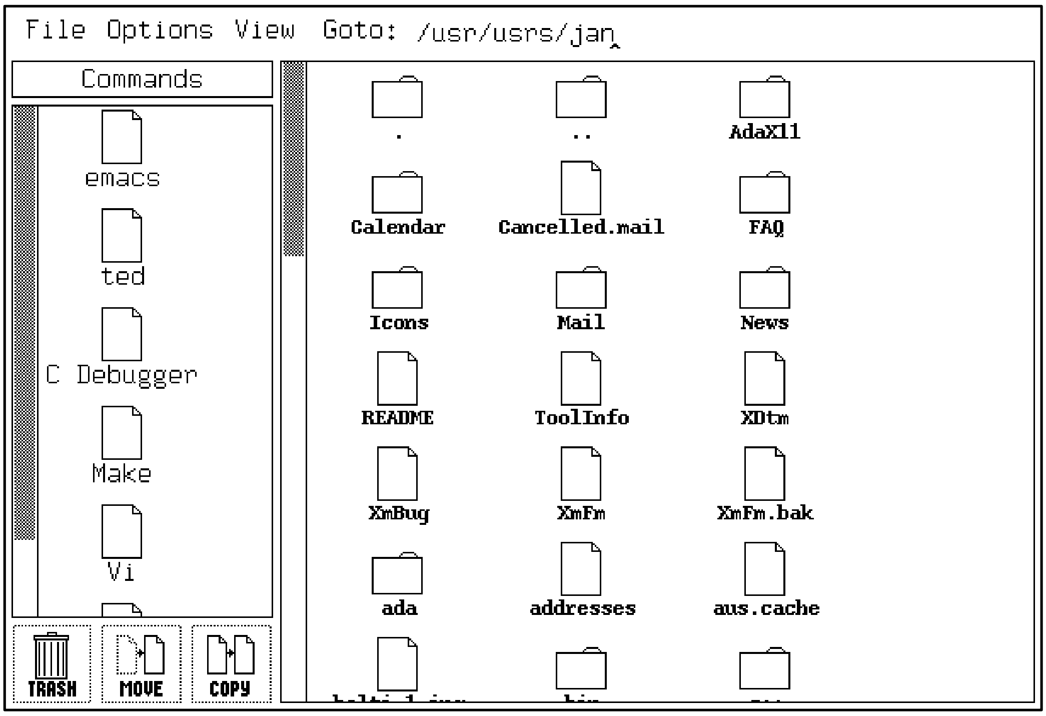

Figure 2: xdtm

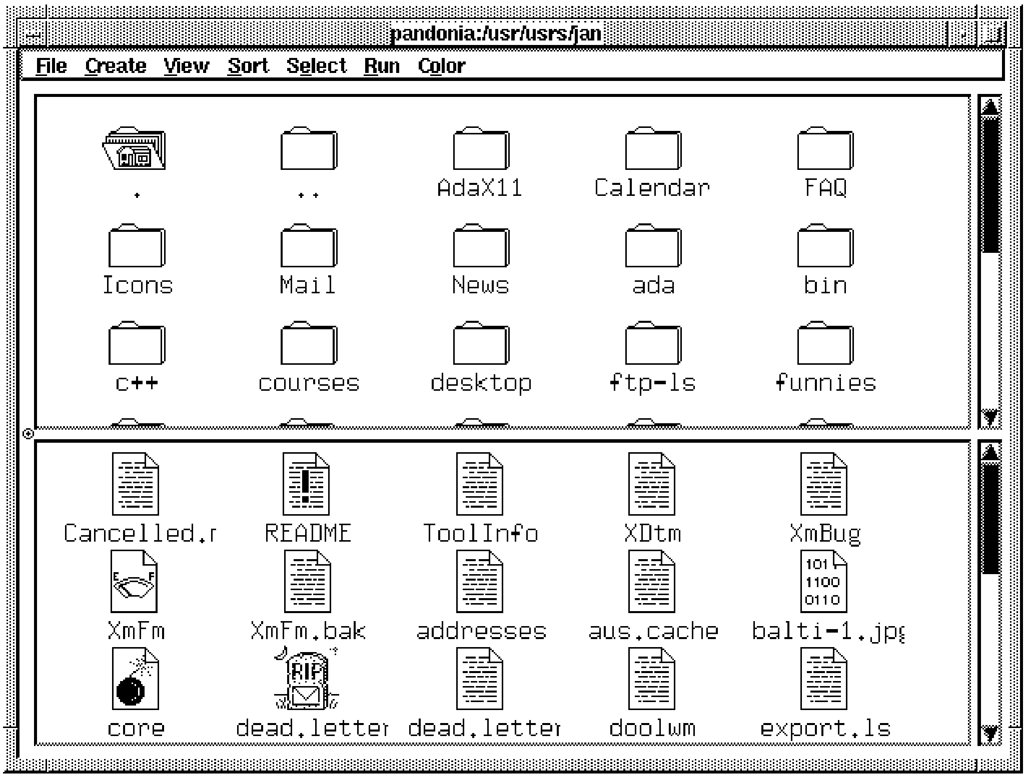

Figure 3: Looking Glass

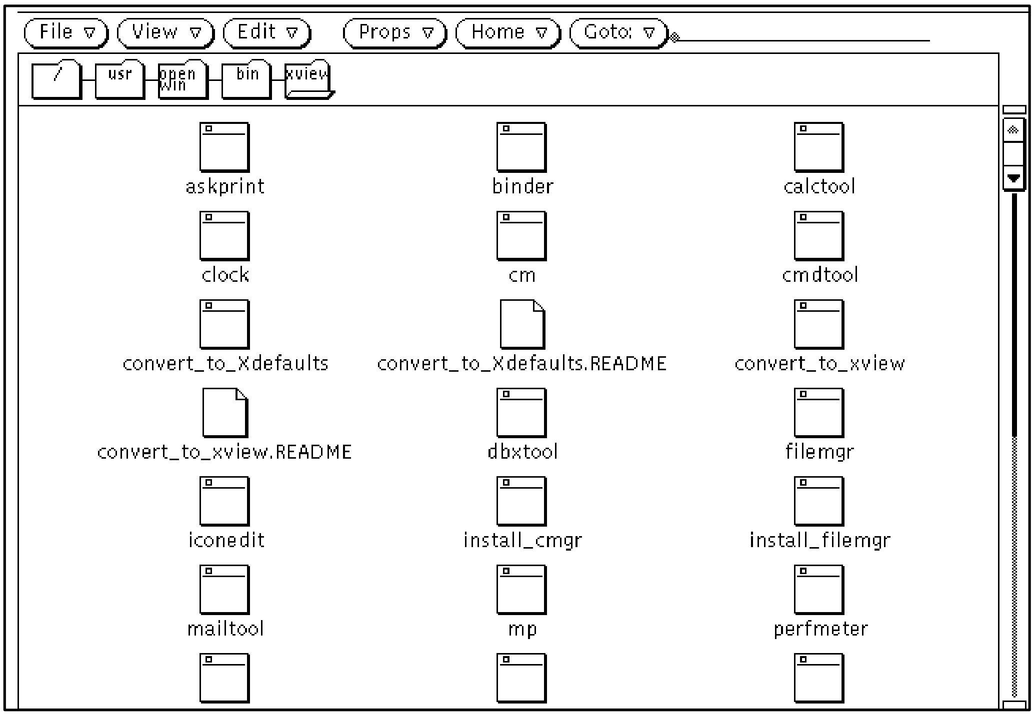

Figure 4: filemgr

xdtm. It has an option to sort this display according to programs, directories and data files, but they still show in the one area causing the same navigation problems as `xdtm`.

Filtering views

In addition to this, `xmfm` allows the contents of each pane to be filtered by a shell pattern. The filters for each pane can be set independantly of each other at any time by the user, so that for example the data–files pane can be set to show all `.c` and `.h` files while the programs pane can be showing all programs. The patterns used include the Bourne shell patterns of * and ?, but in addition also use some of the extended patterns of the Korn shell and `zsh` and allow subranges of characters and excluded ranges. For example, the default pattern for files is `[^.]*` which acts like the default behaviour of the directory list command ls in not showing files beginning with a ".". This is of course accompanied by online help to show how to use these patterns.

The OpenLook `filemgr` also has a filter mechanism, but it is far less useful. It applies the filter to all file types, so it is impossible to set a pattern, say, that shows all .c data files, all programs beginning with x and all directories

Multiple panes

The Openlook file manager allows the user to pick up a subdirectory and drop it onto the desktop, where it opens up a new display on this directory. The disadvantages of this are threefold: firstly, the original directory display is the "master", and all others are slaves to it. The original display contains all of the menus, so that for example, the filter applies to all directories. Secondly, this display cannot be removed without terminating the entire application. Thirdly, new directories can only be opened that are direct parents or children of one that is curently showing.

The manager `xdtm` only has one display area and so cannot show multiple directories.

For Looking Glass, the default double–click action on a directory opens up a new display for that directory with separate menus. The difficult thing to do is to *replace* the current display. This requires pressing the control key at the same time as double–clicking the mouse, which is hard to do if the mouse is used by the same hand that presses the control key (this aspect of Looking Glass cannot apparently be configured to anything better).

In `xmfm`, almost independant new versions of the application can be created for arbitrary directories by a suitable action on visible directories or a menu selection for non–visible ones. Each of these has its own menus, its own set of filters and its own command selection dialog boxes (to be described later). Each of these can be terminated or iconified independantly without affecting the others. There is one point of contact between them: the exit button on any menu will terminate the entire application. Of course, this is protected against accidental use by a dialog box!

Object oriented actions

The popular trend in many branches of computing is to adopt an object–oriented approach to system design and resultant interaction. Neither Unix nor MSDOS have object oriented file systems, and so their limitations have to be worked around. A typical mechanism in MSDOS is to "type" a file by its extension, and then to have a single application that can respond to this file type and perform all possible actions on it. For example, a `.DOC` file may be interpreted by Microsoft Word as a document file. The application—in this case Microsoft Word—will then be responsible for all possible actions to be performed on the file. For many actions this is justified, where the actions are explicitly word–processing actions. However, there are many actions that the user may wish to perform which strictly transcend the word–processing function, such as printer setup, and the requirement to supply every possible action leads to extremely complex software.

In contrast, the Unix approach has always been to supply a large number of tools that can act on many file types. The difference can be illustrated by an example that happened recently: a colleague used to using MSDOS wanted to sort a file: he imported it into Word and used the sort function within Word to do this. The Unix user just runs the utility `sort` on the file. The Unix philosophy has generally been to avoid overly complex tools and use tools that do "just enough". Even some of the more complex applications, such as the Interleaf document publishing system, often rely on small tools making up a whole.

`xmfm` tries to use this mechanism. Instead of attaching one large monolithic application to a file type where the application performs all actions, it allows many smaller actions to be attached directly to a file type. In `xmfm`, each time a file is selected, the set of *all* possible actions is displayed immediately. These appear in the set of "file action" buttons to the left of the file panes. The action can be selected by then clicking on the action button, or by dragging the file onto the button. Often, of course, there is a major action for a file type, that should be accessible as a default action. In xmfm, this is the first action in the set of buttons, and the default can be activated by double–clicking on the file.

The definitions of file types, and the actions available on each, may be left to system configuration or set by individual users. This is considered in more detail later.

This file–typing mechanism is not sufficient for many Unix users. The directory structure of Unix is often exploited by collecting "projects" or certain file types within certain directories. This leads to the concept of directory–based actions. For example, any directory containing a `Makefile` should have the `make` command available, and any directory with an `Imakefile` should have `xmkmf` available. In addition to this, there are system wide actions, applicable to many files whatever their file types. A typical action of this type is "remove" or "copy". Lacking an inheritance mechanism, the simplest way to do this is by a global action section. At present, xmfm combines both directory–based and global actions into a single set of further buttons. The default for these buttons gives both "make" and "remove" within this set of buttons.

This approach is also used by Looking Glass from Visix. However, in Looking Glass you normally only see one action—the default action—and it is hard to access the full range of actions (this also requires the control key plus double click). The OpenLook `filemgr` only allows one action to be attached to a file type, and xdtm has none.

xdtm instead uses a task–oriented mechanism. The tasks may be chosen interactively from a list, such as the compile task or the games task. This shows a set of programs related to the task. For example, the compile task may show an editor, a compiler and a debugger. When a file is selected the appropriate action is chosen by the user. While it brings the appropriate set of actions to the front, xdtm does not prevent the user from chosing the wrong action for a file (such as editing an executable file).

System access

Unix is still heavily command–line based. Indeed, many Unix users (the author of this system included) often find the command–line interface faster and more convenient. Like Looking Glass, xmfm allows command line processing to take place from a dialog box of commands with a history mechanism. Each directory display has its own separate command history list. Looking Glass has one history list for all displays.

In the OpenLook filemgr, runing system commands is quite cumbersome: first a command has to be defined so that it can be added to a menu, and only then can it be run. xdtm has no mechanism at all for this.

Configuration

The xmfm file manager is built from Motif, which in turn is built above the Xt toolkit. Because of this it is fully configurable using the Xt resource database. This allows such things as the colour of any component and the labels used in menu items to be set on a global basis or on a per–user basis. The widget hierarchy of xmfm is documented in the manual entry so that those familiar with the Xt resource management system can reconfigure many aspects of xmfm. The user manual gives more information for those less experienced.

In addition to this, other aspects of xmfm may be configured. These are: file type definitions, icons associated to types, and actions belonging to file types. The user can use their own configuration file or leave it to a system supplied configuration file. The configuration file defines file types based solely on shell patterns. It has not yet proved neccessary to use the more sophisticated mechanisms of Looking Glass, which can also examine the contents of files for type information. Any user can configure xmfm.

For each file type an icon is defined. This icon is searched for in standard Motif directories, depending on various environment variables that the user can set. These directories include system wide places as well as those defined by the user. The icons are just bitmaps as created by the program bitmap, so that a user can define any icons that they desire for their own file types.

The actions for each file type are defined in the configuration file. There are a few built–in actions which must be performed internally by xmfm, such as changing directory and the file information dialog. Many others are ordinary Unix calls such as cc. Parameters control whether or not they are X–Window applications or have to be run in an xterm, and whether or not they need a "pause" statement to retain their xterm display. Further parameters control the arguments to the Unix command, and there is even a dialog mechanism to prompt interactively for parameters. Shell variable patterns are used in these files so that for example ${0%.c} means "the zero–th argument with the trailing `.c` removed from the filename''. Although the syntax of patterns may seem obscure, it will be familiar to many Unix users and allows a large amount of control of actions. (The particular pattern described would be used in creating a file from its corresponding .c file.)

This shows a typical entry for a file type:

```
{       type      f
        pattern   *.c
        pixmap    ccode.xbm
        description           ``C source code''
        {
                  label        edit
                  action       ``ted $0''
        }
        {
                  label        more
                  run_in_xterm
                  pause_after_exec
                  action       ``more $0"
        }
        {
                  label        ``to object"
                  run_in_xterm
                  pause_after_exec
                  action       ``gcc -c $0"
        }
        {
                  label        ``to executable"
                  run_in_xterm
                  pause_after_exec
                  action       ``gcc -o ${0%.c} $0"
        }
        {         label    print
                  action       ``lpr $0"
        }
}
```

While Looking Glass also allows has configuration files, access is needed to their ''resource compiler'' for this. This cannot be done on a per user basis, but instead sets the configuration for all users.

Drag and Drop

`xmfm` uses the Motif 1.2 drag and drop mechanism to allow files to be dragged by the middle button and dropped elsewhere. The only drop action currently supported is the drop of a file on a button, which triggers the action on the file. Design decisions on the effect of a drop elsewhere still have to be made. The drop data (the file name) is a simple string, and so other clients could also act as drop clients. For example, a wastebasket or a printer could be written to accept dropped files. At the moment there are some speed problems in this area that need to be resolved.

Interface standards

`xmfm` is written using the Motif toolkit, and every effort has been taken to follow the Motif style guide. It is thus consistent in action and appearance with other Motif applications. Looking Glass is written using a proprietary system, and although it generally follows the Motif style there are places in which its appearance diverges (such as in resize buttons for panes). The OpenWindows `filemgr` follows the OpenWindows style and is hence compatable with other OpenWindows applications, but it seems unlikely that this will become a major interface style. `xdtm` is written using the Athena widgets for which there is no real style guide. It seems to have many similar elements to other Athena applications.

Availability

The file manager `xmfm` is available for anonymous ftp access from csc.canberra.edu.au under `/pub/motif/xmfm.tar.Z`. It has been run under Motif 1.1 and Motif 1.2 on a Sun Sparcstation.

Conclusion

`xmfm` is still in development. All the features described in this paper work, but `xmfm` is missing many of the features of other file managers such as multiple types of file display (by icon, by name, long listing, etc). Nevertheless, it has enough functionality to be in everyday use by students at the University of Canberra. It differs from other file managers in being tuned to make some tasks as easy as possible by using an object–oriented approach.

Acknowledgements

`xmfm` uses the icon bitmaps supplied with `xdtm`. It also uses the Directory Manager routines from the Free Widget Foundation.