Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference

San Antonio, Texas, USA
June 9-14, 2003

Matchbox: Window Management Not for the Desktop

Matthew Allum
OpenedHand Ltd.
London, England
matthew@openedhand.com

Abstract

Matchbox is a set of X11 utilities for managing applications. Matchbox is intended primarily for use on embedded X Window System devices with low display resolution, limited available input mechanisms, limited available storage and/or slow CPUs.

The core of Matchbox is an X window manager which aims to ameliorate shortcomings of existing desktop window managers on constrained platforms.

Matchbox approaches window management in a unique restricted way, bene£ting the user, while striving to adhere to relevant standards such as the ICCCM and EWMH. Included applications include a PDA style application launcher, a panel and numerous panel applications. Matchbox also hopes to support emerging devices with limited input mechanisms such as Tablet PCs, HUD based devices and "wrist tops".

1 Introduction

The past few years have heralded the wide availability of high-powered handheld computers such as the HP Ipaq. The processing power of such machines is equal to that of the desktop computers of just a few years ago. A typical device will have a 200MHz ARM processor, 32 Megabytes of RAM, and a 320×240 pixel touchscreen display.

During this same period, several factors have contributed to the emergence of open Unix-like software distributions for handhelds. These include the ¤exibility and platform independence of the software, the cooperation of hardware manufacturers, and the hard work of bedroom reverse engineers.

The standard windowing system for Unix-like machines is the X Window System [15], a powerful and ¤exible network-transparent window system. Client applications connect to the server, which performs the actual rendering of windows. A single client, known as the "window manager", performs the task of managing other clients' windows. The tasks of the window manager include framing windows with decorations, providing controls for common actions and managing window placement.

Unfortunately, many window managers do not cope well with a small display and limited input mechanisms such as a touchscreen. Application window positioning and layout are often inappropriate, making the user interface awkward and unfriendly. In addition, specialized handheld computer input mechanisms, such as software keyboards, impose new requirements that often violate assumptions of conventional window managers.

With storage space at a premium, the selection of a window manager is greatly in¤uenced by the size of its installed binary. Unfortunately, this constraint leads to use of simple window managers that lack features, visual ¤air and perhaps most importantly support for standards. Standards support is important so that the myriad of applications can expect a uniform interface for interacting with the window manager.

Matchbox includes a new window manager designed speci£cally for limited platforms. However, the Matchbox window manager also supports pre-existing and emerging standards. As much as possible, the Matchbox window manager attempts to manage existing applications in such a way as to make them more usable on small screens. It also provides features to enhance new applications that are speci£cally developed for handheld and embedded X platforms.

Matchbox also optionally offers features found in high-end desktop window managers including XML-based ¤exible theming, support for modern X infrastructure such as Xft [12] and the RandR extension [7] and support for utility libraries used by KDE [3] and GNOME [4] such as startup noti£cation [13] and XSETTINGS [16].

Matchbox has grown to be more than just a window manager. It now includes a suite of tools for managing X11 applications. This paper will focus on the the window manager, but will also discuss the included panel, desktop, utility library and panel applications.

2 X Window Management

Most window systems place responsibility for managing windows either within each application or within the window system itself. The X Window System [15] diverges from convention by supporting external window management: the geometry and stacking of windows on the screen is managed by a separate application known as the *window manager*. X Version 10 and earlier also used external window management, but were unable to provide decorations and necessary event management to allow the implementation of friendly interfaces.

X Version 11 added several new mechanisms to support sophisticated external window management. Application requests for window placement can be *redirected* to the window manager: the window manager can then augment or amend the request as desired. Application windows can be *reparented*—placed within a window manager frame which also serves to contain window management user interface elements.

The basic X protocol avoids enforcing policy on applications. However, some level of cooperation is required for applications to successfully interoperate with a wide variety of window managers. The ICCCM sets the rules of engagement, so that applications know how to successfully interoperate with arbitrary window managers, session managers and peer applications.

The existing ICCCM standard may have been suf£cient to describe operations in legacy X environments like CDE. New environments have included additional functionality not even conceived of in that era. A new organisation, freedesktop.org, was formed to set standards that extend the ICCCM. Standards promulgated by freedesktop.org add support for modern features like application docking, virtual desktop support, focus management, cooperative window management and additional window types. Many of these new features are very useful when managing limited screen space and input bandwidth. An important principle obeyed by all of these standards is that they do not specify precise behaviour. Rather, the standards describe required semantics of operations. As a consequence, applications and window managers are expected to accept any behaviour permitted by the speci£cation. This is particularly important when developing novel window management techniques that dramatically change how windows are manipulated on-screen.

2.1 Basic Window Management

At the most primitive level, window management in X starts when an application creates a top-level window (known in X parlance as a "child of the root window"). The request from the application to make the window *mapped* (visible) is not acted on directly by the window system server. Instead, the window manager captures the window mapping request and performs whatever preparations may be necessary. These preparations usually include moving the application window to the interior of another window, then creating controls to manipulate the exterior window once it becomes visible.

Once the application window is prepared, the window manager makes it visible. The user can then manipulate the window through the controls drawn by the window manager. Future requests to change the window's size, stacking or visibility from the application are again redirected by the X server to the window manager. The window manager then performs the requested operations on the user's behalf. X11 applications are expected to accept whatever position and size they receive from the window manager. For example, an early X window manager, the RTL Tiled Window Manager from CMU, would adjust the size and position of application windows to keep all windows fully visible on the screen.

2.2 Inter-client Communications Conventions

The basic window manager communication primitives of the X11 protocol suf£ce for simple applications. However, applications normally want the window manager to handle more sophisticated semantics. The core protocol provides no mechanism for applications to describe intended associations among windows and window modes. The ICCCM standard sets conventions for communicating this information.

ICCCM communication occurs through properties set on various windows, synthetic events generated by applications, and standard server-generated events. Clients set properties on their top-level windows to inform the window manager about various attributes. Common ICCCM properties include:

WM_NORMAL_HINTS Describes the set of desired sizes as a base size, a size increment and minimum and maximum sizes.

WM_HINTS Describes the desired "state" of the window which is one of Normal, Iconic or Withdrawn, along with an icon to be associated with the window and any window group association.

WM_TRANSIENT_FOR Set for pop-ups to connect them with the associated main application window. Used for dialogs, not menus.

The CDE environment and Motif window manager (mwm) included additional properties on application windows to control mwm-speci£c decorations around application windows, such as the minimize/maximize control and window menu contents. These were not codi£ed by the ICCCM, but have been widely implemented in existing window managers.

There are other properties holding window and icon name information but the basic ICCCM doesn't provide any Unicode representation leaving these names effectively limited to ASCII or perhaps ISO Latin-1.

As far as a window manager is concerned, ICCCM compliance largely revolves around correctly interpreting application requests and sending the right messages back to applications. It has wide latitude in how to position windows on the screen and few hints on how windows are expected to be used.

2.3 Extended Window Management Hints

The KDE and Gnome project teams are working together on several standards to improve interoperability of applications and desktop environments. They adopted a set of ICCCM extensions proposed by Carsten Haitzler and Marko Macek and have published them as the Extended Window Manager Hints (EWMH) on the freedesktop.org web site.

The EWMH standard takes up where the ICCCM left off. EWMH sets policies and conventions to provide applications more control over how windows are managed on the screen. A key piece of additional information speci£ed by EWMH is the classi£cation of windows into broad semantic types:

- DESKTOP A desktop window the size of the screen, placed beneath other windows. The window often contains icons that can be manipulated directly.
- DOCK A dock or panel window. The window is often the width or height of the screen and placed along the edge to hold menus or other controls. The window is typically stacked above all other windows.
- TOOLBAR,MENU A toolbar or pinnable menu window respectively (i.e. toolbars and menus "torn off" from the main application). The application may also set WM_TRANSIENT_FOR property on the window to mark the related application window.
- UTILITY A persistent utility window like a palette or toolbox. These windows are different from toolbars: they are not "torn off" from the main application. These windows are also different from dialogs, because they are not transient and will probably stay open during work in the main application window. As with toolbars and menus, WM_TRANSIENT_FOR may be set.
- SPLASH A window marking application startup.
- DIALOG A transient dialog window. Windows without an EMWH type that have WM_TRANSIENT_FOR set are assumed to be of this type.
- NORMAL A normal application window. Windows without either an EWMH type or WM_TRANSIENT_FOR are assumed to be of this type.

These types allow the application and window manager to cooperate in con£guring the desktop environment and build different elements with separate applications.

EMWH also add many new window states beyond the Withdrawn, Iconic and Normal states speci£ed by the ICCCM:

- MAXIMIZED_VERT,MAXIMIZED_HORZ The window is maximized in the vertical or horizontal dimension.
- FULLSCREEN The window should £ll the screen without any visible decorations. A presentation program would use this hint.
- ABOVE,BELOW The window should be stacked above or below most regular windows.

EWMH borrows an idea from the Motif window manager hints: applications can select which kinds of controls should be included in any window management decorations or menus. EWMH also allows applications to reserve space along the edge of the screen for panels and other controls. This instructs the window manager to avoid covering such windows when maximizing other windows. Finally, EWMH includes UTF-8 encoded window and icon titles to allow localized strings to be used. Window manager labels can thus be speci£ed in a variety of languages.

3 Related Work

There are numerous window managers available for the X Window System, each with its own merits and unique features. Several window managers were investigated in terms of their suitability for use on a handheld device with limited application storage space and memory. The following window managers were considered: saw£sh [9], blackbox [1], icewm [10], ion [17], and aewm [6]. The results of the investigation are described below.

- Saw£sh is a powerful, full-featured, programmable window manager. The power of saw£sh comes at the expense of storage space requirements. While the core saw£sh binary is only 128KB it has a large number of library dependencies, one of which contains a lisp interpreter. It is true that when compared with desktop software, saw£sh would likely not be considered a large application. But its size becomes quite signi£cant in a system with as little as 16MB available for application storage.
- Blackbox is fast and visually appealing and depends only the core X libraries and libstdc++. Unfortunately it lacks support for modern EWMH standards. Its user interface relies on multiple mouse buttons—the right mouse button is used to access its root menu. It allows applications to initially size themselves larger than the actual display.
- Icewm has good standards support. It does constrain application windows to the size of the display and is usable with a touchscreen, not relying on multiple mouse buttons. It also has a useful built-in panel. However navigating between windows on a small display is uncomfortable and its binary size is a rather large 480k.

- Ion is small and has few dependencies. It is novel in that it tiles windows on the display. This makes good use of available display area as there are never overlapping windows. Unfortunately it is very dependent on keyboard control, does not always handle dialog windows well and has limited standards support.
- Aewm is very small, has basic ICCCM compliance and only depends on core X libraries. However it is very basic both in functionality and visual appearance. Aewm relies heavily on multiple mouse buttons for its user interface making it nearly impossible to use with a touchscreen. It also allows for windows to resize themselves greater than that of the display. It is worth noting that Matchbox was initially based on aewm, though it now bears little, if any, resemblance.

Some of these window managers are of acceptable size and of nearly acceptable usability. As all are open source, they can be freely patched and adapted as needed. However none of them are ideal: the fact remains that no pre-existing window manager has been speci£cally designed for a PDA style device. The author's frustration with this situation has been the impetus for the creation of Matchbox.

4 Theory Of Operation

Consider the usage of a conventional desktop window manager on a constrained device with a 320×240 display, no keyboard, and a touchscreen capable of only generating left mouse button events.

At best, applications too big for the display will be limited and resized to the full display size. However at worst, and more commonly, applications will get their requested window size and thus be obscured off screen. This is not a useful situation for the end user.

Assuming the application does £t on the display, or has been resized by the window manager to do so. It is very unlikely the window manager will provide any easily accessible mechanism to allow the user to then select between clients. Aids for this type of operation will be missing from the window title bar, a root window menu will be inaccessible and with no keyboard, key shortcuts are not possible. The user is left to awkwardly drag the top-level window off screen to reach lower level windows. With a large number of applications open this situation soon becomes unworkable.

There are other problems too, appropriate actions will not be taken to properly facilitate an input device window such as a software keyboard. For example, a software keyboard, to work well needs to be treated as a special case by the window manager and not the same as other application windows. It ideally needs not to overlap the application its being used to enter text into. If this does happen it means unnecessary extra move resize actions for the the user. The keyboard also needs to never be given keyboard focus, otherwise it will end up sending key events to itself. While there is an ICCCM convention for specifying this, it is fairly obscure and not always implemented - for example blackbox does not.

Matchbox attempts to solve these problems primarily by managing window in a restrictive way. Restrictive management on such a device is good for the user. Actions, such as moving and resizing a window, are easy with a mouse but problematic with a stylus. If the display area is small these actions are needed repeatedly as the user struggles to move between overlapping windows. Therefore removing the need for the user to ever resize or move a main application window and handling it in a rudimentary way in the window manager provides an easier to use en system.

Matchbox organises applications in a manner similar to a deck of cards. Top-level application windows are requested and ultimately forced to take all available space, remaining locked to a static position. At any given time, only one such window is visible. The user is able to navigate through the deck of applications by various means. The window title-bar decoration may have buttons for navigating to the previous or next application in the window deck. There may be a button in the title-bar for a drop down menu listing an option for each application on the deck. Clicking an option will make that application visible. The presence, appearance and position of these buttons is con£gurable by the Matchbox theme.

Other navigation aids include keyboard shortcuts and external application task manager style programs. Any task manager that is EWMH compliant (such as those included in GNOME and KDE) will work.

This strategy is simple and restrictive in nature, but helps enormously on target devices trying to balance the applications needs with practical user control.

The strategy also £ts in with the standards discussed above. While the application provides geometry hints to the window manager, the window manager may ultimately choose to set a different geometry.

An application window may be treated differently to the above if it provides standard hints or its properties match a certain criteria. Each of these mechanisms are discussed below.

4.1 Dialogs

Matchbox attempts to accommodate application dialog windows. Rather than being resized to £ll the display like their parent application windows, their requested size is usually honored, (if within the display), and they are not statically positioned, allowing the user to drag them about the display.

Dialog windows are "paged" in the deck along with

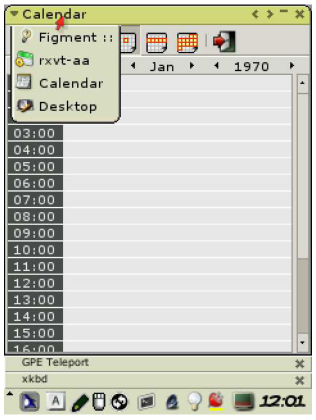

Figure 1: Ipaq screenshot showing Matchbox in action with various navigation aids.

their parent applications. Dialogs which do not have a parent, are permanently visible ands are placed on the top of the window deck.

A dialog may be resized to £t the display, not cover panels or input devices if too large.

Matchbox uses the following criteria to decide if a window's properties nominate it to be treated as a dialog.

- The window sets its WM_TRANSIENT_HINT property.
- The freedesktop.org _NET_WM_WINDOW_TYPE property is set to _NET_WM_WINDOW_TYPE_DIALOG, or _NET_WM_WINDOW_TYPE_SPLASH.
- The window speci£es that it is part of a group but not the group leader.
- Motif WM hints specify the window wants no decorations.

This last criteria is less reliable than previous ones. The rationale behind treating undecorated windows as dialogs is a heuristic based on the behaviour of many applications that use this kind of hinting. Many of these are shaped shaped windows from applications such as gkrellm or xmms which simply break terribly when forced to resize.

4.2 Panels

A panel, or dock as it is sometimes referred to, is an area of the display used to hold small applications. These "applets" may be used to launch applications or provide information and noti£cation to the user. Matchbox supports freedesktop compliant panels and places them along any edge of the display. They are always visible, and windows of other types are forced to £t around them.

A panel is identi£ed when its _NET_WM_WINDOW_TYPE property is set to _NET_WM_WINDOW_TYPE_DOCK.

4.3 Desktops

A desktop window is a full screen window with not title bar. It is positioned at the bottom of the window deck.

A panel is identi£ed when its _NET_WM_WINDOW_TYPE property is set to _NET_WM_WINDOW_TYPE_DESKTOP.

4.4 Toolbar windows

Toolbar windows behave similar to the way user interface toolbars behave in applications such as web browsers. Toolbars are placed at the bottom of the display, below main application windows and above any panels. Their presence causes main application windows to resize accommodating them and they remain visible during application paging. Their width is set to the full display width and multiple tool bars are stacked vertically on top of one another.

Toolbar windows are collapsible with the window going into an iconic state and allowing the user to quickly free up screen real estate. This state is also easily reversible via buttons on the windows frame.

The purpose of tool bar windows is to provide accommodation for small utility type windows, speci£cally input devices such as software keyboards.

A tool bar is identi£ed when its _NET_WM_WINDOW_TYPE property is set to _NET_WM_WINDOW_TYPE_TOOLBAR.

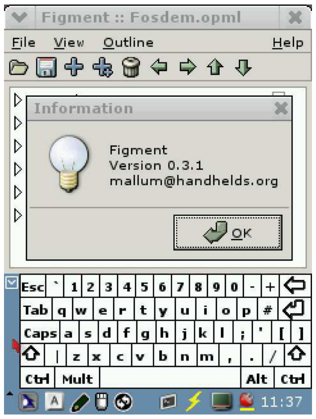

Figure 2: Ipaq screenshot showing a dialog and toolbar window.

5 Included Utilities

It is desirable for a window environment to include utilities for launching applications and managing instances of existing ones. The Matchbox window manager supports this to a degree via keyboard shortcuts and its built-in title bar task switcher.

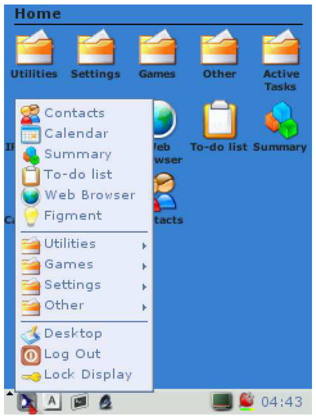

Figure 3: Ipaq screenshot showing mbdesktop, mbdock and various panel applications

Existing applications that £ll this role are either too big and unsuitable for embedded environments, or small but limited, lacking support for modern standards. For example the GNOME Panel provides a rich application management tools and much more. It is also ¤exible in usage allowing it, at least from a usability point of view, to be practical on a constrained device. However it has a huge range of large library dependencies instantly making it unsuitable when application storage space is at a premium. There are numerous simple toolkit based menu launchers available but many implement a unique non standard way of adding entries, are written in an unwanted toolkit on the target device, or lack support for X server extensions such as RandR. RandR is a modern X Server extension which allows for on the ¤y display resizes and rotations. This can confuse the positioning of older toolkit widgets that are not RandR aware.

To solve these problems, the Matchbox distribution contains a number of lightweight tools which are toolkit independent and adhere to freedesktop.org standards, allowing them to interoperate with environments other than just Matchbox. These include a panel and numerous panel applications such as system monitors and a menu based launcher, a PDA style "desktop" and a shared utility library.

5.1 The Panel and Panel applications - mbdock

The panel provides a small permanently visible area of the display to house small separate application windows. These applications are typically application managers, such as a launcher, system monitor or noti£cation tools.

The docking mechanism works independently of the window manager and implements the SYSTEM_TRAY[14] and XEMBED[5] speci£cations found at freedesktop.org. A panel application locates the panel by means of querying an X selection for a window ID then communicating with a series of X messages and then £nally being reparented and mapped by the panel.

This lightweight mechanism is also used to a degree in GNOME and KDE, affording interoperability of both the panels and panel applications

The Matchbox distribution includes numerous small panel applications. Mbmenu is a menu based application supporting entries in both the Debian "/usr/lib/menu" format and .desktop £les[2], as used by GNOME and KDE. The .desktop format as the advantage of supporting internationalization and startup noti£cation. There are monitor tools included with the distribution for memory and CPU usage, wireless signal strength, available battery power and a simple sound mixer tool . Numerous other third-party panel application are also available.

The panel also includes support for both multiple instances and multiple orientations.

5.2 The Desktop - mbdesktop

The included desktop is another application manager in a PDA style. Application icons and titles are placed in sectioned grid-like views. Mbdesktop is still in early stages of development with future plans being to allow extension to what is displayed by means of loadable modules. For example this could extend mbdesktop to do more than just application management - it could display browsable URL bookmarks, or specify an area of the desktop to provided PDA "Today" style summary textual information.

5.3 Shared utility library - libMB

LibMB contains useful shared code used by all included Matchbox utilities and optionally the Matchbox window manager itself.

Included code includes;

A small fast pixel buffer library for client side images. This library performs loading of PNG and XPM images, basic manipulation and composition. This part of the library is used by all Matchbox utilities and optionally the window manager.

The Abstraction of the XEMBED and SYSTEM_TRAY protocols for easy creation of panel applications. It also safely extends the speci£cation to allow alpha composition of panel applications on the panel.

A simple menu widget speci£cally designed for usage with touchscreens. This is used by both the panel and some panel applications.

A small .desktop £le parser. Used by application launchers.

Various utility calls, for operations such as grabbing the root window image.

6 Flexibility / Tailoring to platform

Already there is quite a range in the capabilities of platforms supported by Matchbox. There are older and new cheaper devices which may be limited to a 4-bit greyscale display and 16MB of ram. There are also newer devices emerging such as the Sharp Zaurus SL-C700 which features a 640x480 16-bit display, a fast CPU, and ample memory.

It is desirable as a developer to take advantage of increased resources on newer devices, but less desirable for the end user to £nd his older device is no longer supported.

Matchbox attempts to keep all parties happy by the use of numerous compile-time options. These options produce numerous enhancements to operation but keep the core working essentially the same. The compile-time options allow for £ne-grained control over library dependencies, features, binary size and memory usage.

A good example of this in action is the theming engine. People with higher-end devices will appreciate their window decorations being visually exciting and ¤exible—supporting multiple image formats, alpha blending and anti aliased fonts. Others however, may value their CPU cycles and storage space more preciously and happily do with out such features.

The Matchbox build system uses GNU autotools [18]. Using the con£gure script, the speci£cs of the created binaries can be controlled.

All build permutations provide the same core window management operations. Differences are primarily enriched visual look, and support for external libraries that provide features external to core window management.

With no options passed to the con£gure script, a conservative Matchbox will be built. Support for con£gurable theming is included though it lacks support for features like anti-aliased XFT text and PNG based images.

For a very tiny Matchbox the '–enable-standalone' con£gure option is used. This effectively replaces a large chunk of code implementing theming and XML parsing with a non-con£gurable theming system that uses only core X rendering functions to create a simple but fast look for window decorations. A standalone Matchbox also is dependent only on core Xlib libraries and no external con£guration or image £les.

A high-end Matchbox with most con£gure options enabled will give much richer theme support with support for PNG images, Xft anti-aliased fonts and freedesktop.org extensions such as XSETTINGS and startup noti£cation.

7 Implementation

Matchbox is implemented in about 26400 lines of C code. Table 1 shows breakdown of the lines of code among the various modules of the window manager, the utilities, and the library shared between them (libMB).

Module	Lines of code
Window Manager	10744
libMB	5152
dock	4041
desktop	2872
7 applets	3574

Table 1: Lines of Code per Module

Matchbox is written in a pseudo object-oriented style. The core of the window manager is an event loop which dispatches events to client objects as shown in Figure 4. Each client object represents an application window, and there is a separate client class for each of the supported window styles. Figure 5 shows the client class hierarchy.

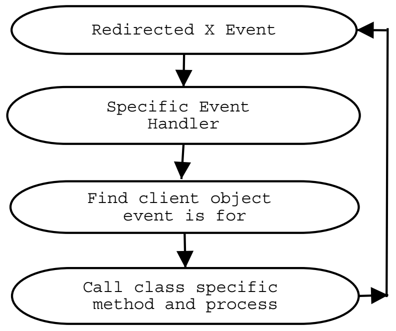
Redirected X Event
Specific Event Handler
Find client object event is for
Call class specific method and process

Figure 4: Matchbox event loop

All clients are stored in a circular doubly-linked list. The client structure is quite simple. In addition to basic client state information, (name, position, size, etc.), it contains pointers for the following class-speci£c functions:

- reparent
- redraw
- button_press
- move_resize
- con£gure
- get_coverage
- hide
- show
- iconize
- destroy

The assumption is these particular methods should provide enough abstraction for the operations on various client types - though a client type may physically perform differently to the same method called on different client type. For example the iconize method will cause a main decked application to drop to the bottom of the deck, while if called on a toolbar application it will cause its toolbar to collapse.

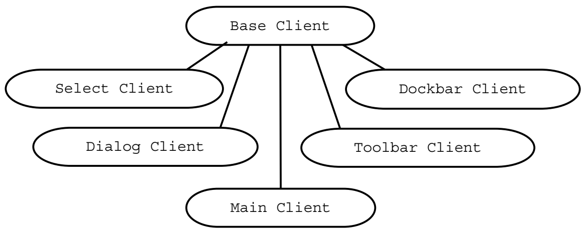

Figure 5: Matchbox client class hierarchy

A particular method or group of methods invocation can be mapped to the receiving of an event. For example on reception of an expose event, the client object for the event will be located and its redraw method called. Different window types need painting in different ways and structuring the code in this object-oriented way removes the need for lengthy if-then-else or switch-case code constructs.

Unfortunately the abstraction is not perfect and there does exist a few special cases where a workaround is required. The £xing of these would require a major reworking and restructuring of the code and with the discovered special cases being few and minor this has not yet been required.

Separate to this class structure there are various function calls which will act on the entire list of managed clients. The most important of these are window resizing and layout functions. If a client such as a toolbar resizes it will no doubt affect all other clients, so these functions manage keep the layout uniform.

There is a separate “theme engine” which performs the task of painting frame decorations. As discussed earlier, there are two interchangeable implementations of the theme engine, a ¤exible engine con£gured with XML con£guration £les and external graphics, and a standalone version designed for small £xed-use builds of Matchbox. Using the standalone engine reduces the size of Matchbox by about 25%.

8 Evaluation

Table 2 compares Matchbox with with other window managers previously mentioned.

Matchbox is able to achieve dramatic savings in size when compared to most window managers. Part of the savings is due to the restrictive window management style in Matchbox. Fewer supported styles of window management operations means less code bulk.

At the same time, Matchbox demonstrates that a window manager can be very small without going to the extreme minimalist style of a window manager such as aewm. Matchbox supports modern standards and provides high-level functionality with its theme support. The ¤exible build operations allow Matchbox to span a wide range of usage scenarios including static embedded systems. This is a a good model that can be followed by developers of many different classes of applications.

9 Future Work

The Author is currently happy with the core functionality and features provided by the Matchbox window manager. Improvements in the future will most like consist of minor improvements and bug £xes.

The project could bene£t from some formal usability testing. Most user interface improvements and decisions have been personal decisions of the author or based on ad hoc feedback from users. Most user interface problems can be quickly solved with the ¤exibility of the theme con£guration £les requires no code changes.

Also investigation and possibly improvements to usage on newer larger constrained devices including Tablet PC’s and set-top boxes. Initial experimentations on such devices have proved quite positive.

There is also the possibility or modi£cations for usage with future devices such as ’Head Up Display’ based machines and so called ’wrist-tops’.

Most future improvements lie in the included utilities. Applications such as mbdesktop are still in early stages which much experimentation and newer features waiting to be done.

LibMB needs stabilizing with a solid API as well as documenting so it can be safely used by other developers.

There is scope for new included utilities, such as a small session manager or a panel based task switcher.

Also to avoid bloat, careful consideration has to be made before the addition of any new features that cannot be made as a compile time option.

10 Conclusion

Matchbox has proven very popular within the community of users of the Familiar[8] handheld environment. Familiar is a Linux distribution is for HP Ipaqs.

GPE[11], a GNOME like PDA environment based on GTK+ has chosen Matchbox as its core window manager. It too strives for standards support so the two £t together well. GPE developers have also written numerous panel applications.

I have also had reports of Matchbox being used with GNOME and KDE on desktop machines intended to be used by small children. Matchbox is also of value with these desktop environments on platforms such as Tablet PC’s and personal video recorder style media boxes.

This success seems to suggest that design decisions made were correct.

11 Availability

Matchbox is free software released under the terms of the GNU general public license (GPL). It is

Name	Version	Size	Dependencies	Standards	Theme Support
saw£sh	1.3	192K	X libraries, GTK+ libraries, librep and rep gtk bindings	ICCCM, EWMH	yes
blackbox	0.65.0	328K	X libraries, libstdc++	ICCCM	yes
ion	20020207	176K	X libraries	Limited ICCCM	no
icewm	1.2.6	480K	X libraries, Imlib	ICCCM	yes
aewm	1.2.2	24K	X libraries	ICCCM	no
matchbox	bells and whistles 0.5r2	88K	X libraries, libpng, libxsettings, libstartup-noti£cation, libexpat	ICCCM, EWMH	yes
matchbox	default build 0.5rc2	59K	X libraries, libpng	ICCCM, EWMH	yes
matchbox	standalone 0.5rc2	44K	X libraries	ICCCM, EWMH	no

Table 2: Window Manager Comparison

known to compile for Linux, various BSDs and Solaris.Releases, documentation and more are available at http://handhelds.org/ mallum/matchbox .

12 Acknowledgements

My greatest thanks go to my £ance, whose help, encouragement and patience have helped my ideas become a reality. Thanks also to Carl Worth, Keith Packard and Jim Gettys, whose advice and motivation throughout the project have been a great help. Finally, thanks to Keith, Carl, and Bart Massey for their help in producing the £nal draft of this paper.

References

[1] Adam A. Bellinson. The blackbox window manager. `http://blackboxwm.sourceforge.net`.

[2] Preston Brown, Jonathan Blandford, and Owen Taylor. Desktop entry standard. `http://www.freedesktop.org/standards/desktop-entry-spec.html`.

[3] Kalle Dalheimer. KDE: The highway ahead. In *Linux Journal*, number 58. Feb 1999.

[4] Miguel de Icaza. The GNOME project. In *Linux Journal*, number 58. Feb 1999.

[5] Mathias Ettrich and Owen Taylor. Xembed protocol speci£cation. `http://www.freedesktop.org/standards/xembed.html`.

[6] Decklin Foster. aewm. `http://www.red-bean.com/~decklin/aewm/`.

[7] James Gettys and Keith Packard. The X Resize and Rotate Extension - RandR. In *FREENIX Track, 2001 Usenix Annual Technical Conference*, Boston, MA, June 2001. USENIX.

[8] Alexander Guy and Russell Nelson. The familiar project. `http://familiar.handhelds.org`.

[9] John Harper. saw£sh: an extensible window manager. `http://sawmill.sourceforge.net`.

[10] Marko Macek. icewm. `http://icewm.sourceforge.net`.

[11] Colin Marquardt. Gpe: The gpe palmtop environment. `http://gpe.handhelds.org`.

[12] Keith Packard. The Xft Font Library: Architecture and Users Guide. In *2001 XFree86 Technical Conference*, Oakland, CA, October 2001. USENIX.

[13] Havoc Pennington. Startup noti£cation protocol. `http://www.freedesktop.org/software/startup-notification/`.

[14] Havoc Pennington. System tray protocol speci£cation. `http://www.freedesktop.org/standards/systemtray.html`.

[15] Robert W. Schei¤er and James Gettys. *X Window System.* Digital Press, third edition, 1992.

[16] Owen Taylor. Xsettings - cross toolkit con£guration proposal. `http://www.freedesktop.org/standards/xsettings/xsettings.html`.

[17] Tuomo Valkonen. ion. `http://modeemi.cs.tut.fi/~tuomov/ion/`.

[18] Gary V. Vaughan, Ben Elliston, Tom Tromey, and Ian Lance Taylor. *GNU Autoconf, Automake and Libtool.* New Riders, 2000. ISBN 1-57870-190-2.