An Overview of twm (Tom's Window Manager)

Tom LaStrange

Solbourne Computer Inc.
Longmont, CO
toml@Solbourne.COM

ABSTRACT

twm is a publicly available window manager for the X Window System Version 11. It started out as a simple reparenting window manager with much of the same capabilities found in existing window managers. Since its initial release in April 1988, it has continued to evolve from suggestions and contributions taken from the X community. Its strengths are ease of use, small code size, and feature-rich functionality. This paper focuses on the basic architecture of twm and some of the unique features which set it apart from other window managers.

1 Introduction

twm is an easy-to-use window manager for the X Window System that complies with the conventions proposed in the Inter-Client Communications Conventions Manual (ICCCM). It is a reparenting window manager; thus, client programs that create windows, which have the X root window as their parent, are given a new parent. The new parent is a *frame* window created by twm. The frame window contains not only the window of the client but can optionally contain a title bar that contains several *buttons*. The title bar and title bar buttons allow the user to perform basic window manipulations such as moving and resizing without the need to simultaneously press keys on the keyboard. The title bar also displays the name of the client window.

While some of the concepts described in this paper are familiar to the current twm user, many features are described that will not be available until the X11, Release 4 version of twm.

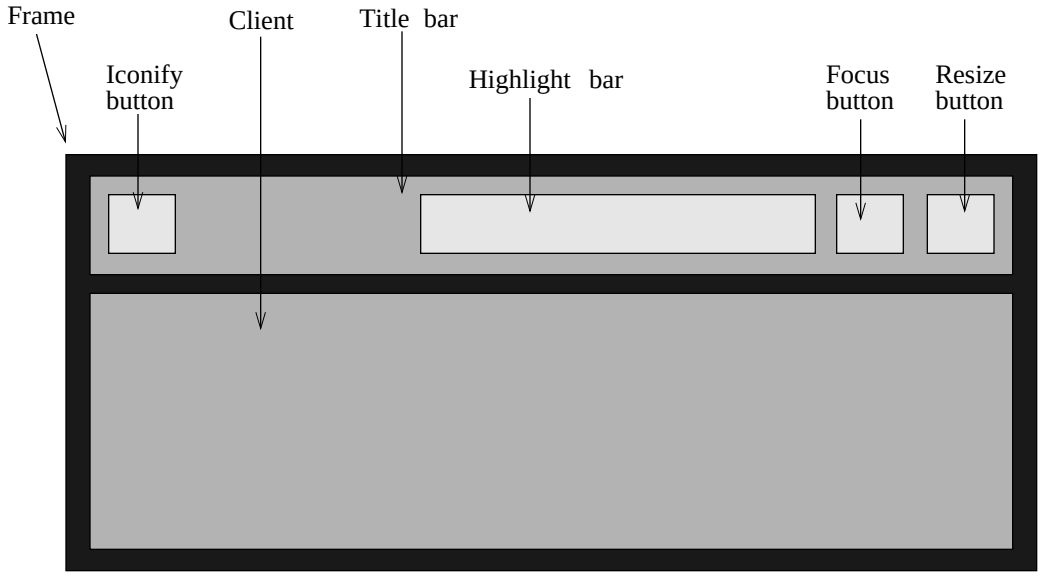
2 Basic Architecture

Unlike most current X clients, twm is not based on a toolkit or widget set. It was written directly on top of the Xlib C interface. There are pros and cons to this approach. About the only disadvantage to not using a toolkit or widget set is that resource specifications for twm are specified in a separate twm initialization file rather than in the standard .Xdefaults file. Although this could be remedied, it would cause twm resources to be split across two files: the .Xdefaults file for color and variable declarations, and a twm initialization file for menus. Advantages to not using the toolkit are many, including smaller code size, better control over resource usage such as windows and pixmaps, and more flexibility. The following sections describe portions of twm in more detail.

2.1 Reparenting

As previously stated, twm is a reparenting window manager. This basically means that when a client program creates a window that is a child of the X root window and maps it (attempts to make it visible), twm gets an event from the X server indicating that a window is requesting to be mapped. twm does a number of functions with this event. The most significant function is that the client window is *reparented* by twm. This means the client window is no longer a child of the root window; it becomes a child of a twm *frame* window. In addition to the frame window, twm may or may not create a title bar which is also a child of the frame window. The following diagram illustrates the hierarchy of windows in the frame. Each shade shows a different level in the

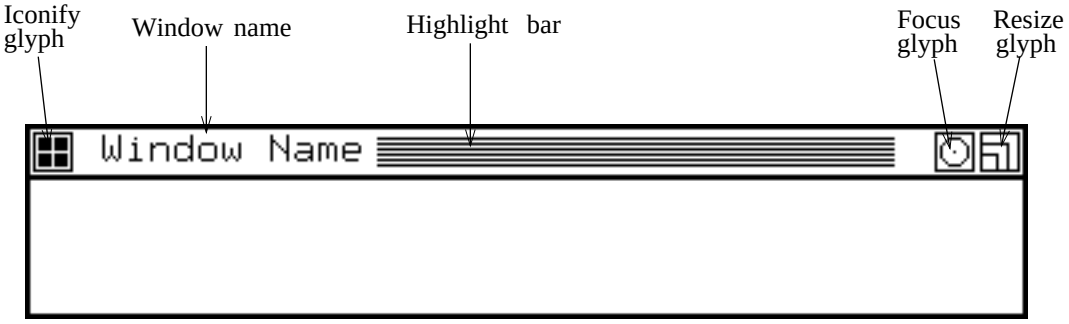
hierarchy, the lowest level is the darkest.



The following sections cover the frame and its function in greater detail.

2.2 The Title Bar

When X Version 11 was introduced, the concept of a title bar shifted from being the responsibility of the client to being the job of the window manager. A title bar is a horizontal rectangle placed above a client window. The twm title bar has five areas of interest as show below:



By default, the title bar recognizes pointer button presses and performs certain functions depending on what button is pressed. The sensitive area is any area of the title bar not covered by one of the three glyphs. By default, pointer button one causes the window to be raised to the active position (top); that is unobscured. Pointer button two initiates a move operation, which allows the user to move the window on the screen. Pointer button three causes the window to be lowered to the bottom of the stacking order. The title bar also has the feature of resizing itself and the buttons to correspond with the font currently displaying the window name. Each area of the title bar now be described in detail.

2.2.1 The Iconify Glyph

When a pointer button press is detected in this glyph, the frame window becomes unmapped (made not visible), and depending on how twm is setup, one of several actions occur. The default action is that a small icon window is mapped. The icon window typically contains two areas: an image area, and a text area which contains the icon name of the client. The icon name may or may not be the same as that displayed in the title bar. More about icons is discussed in section 6.

2.2.2 The Window Name Area

The window name area contains the current name of the client window. This name may be changed by the client at any time and such a change is reflected in the title bar. A useful example is one of a terminal emulator that displays the current directory in the title bar.

2.2.3 The Highlight Bar

The highlight bar is represented as a series of horizontal lines. The highlight bar is displayed within the title bar of the active window. The active window has the keyboard focus.

2.2.4 The Focus Glyph

When a pointer button press is detected in this glyph, the keyboard remains focused to the client window until the user presses the focus glyph a second time or executes the twm function to focus the keyboard on the root window. By default the keyboard focus simply follows the pointer. The focus glyph is intended to look like a bull's-eye because you need to *focus* on a bull's-eye in order to hit it.

2.2.5 The Resize Glyph

The resize glyph allows the user to resize the client window. When a pointer button press is detected here, the cursor changes and an outline appears which represents the outline of the frame. The user may then move the pointer to any outside edge of the frame and begin resizing from that position. The resize operation is completed when the pointer button is released. The resize operation may be aborted by pressing another pointer button while in the middle of the resize operation.

3 Initialization

When twm begins execution it attempts to manage each unmanaged screen being run by the X server. By unmanaged screen we mean, each screen that exists and does not currently have a window manager running. This allows you to have one window manager process running for all screens, rather than having a separate window manager process for each screen.

After having found each screen being used by the X server, twm sets up default values for all variables such as color, cursors, appearence, and title bar pointer functions. Following this, twm attempts to read an initialization file. This allows the user to customize the behavior of twm. There are three different default files that twm attempts to read to get initialization information:

1. $HOME/.twmrc.<screen number>
2. $HOME/.twmrc
3. /usr/lib/X11/twm/system.twmrc

In addition to these default files, the user may also supply an initialization file by starting twm with the appropriate option. Note that the initialization file is read and parsed once for each screen being managed. This al-

lows appropriate colors to be allocated and pixmaps of the proper depth to be created for each screen. Option number one allows the user to have a separate initialization file for each screen. This is useful for the user who needs a different environment on each screen. If no initialization file is found with a screen number suffix, the $HOME/.twmrc file is attempted. If multiple screens are being managed but the same environment is desired on all screens, this is the only file that needs to be present. If no local initialization files are found, a system initialization file is used. This system initialization file can be setup such that new users do not have to get a copy of an existing initialization file, a system wide default can be setup here. This system initialization file is not read if a local initialization file is found.

4 Window Manager Functions

All window managers provide some basic functions. These functions include such common operations as raising, lowering, iconifying, and resizing of windows. We have already described the default functions that are available from the title bar, but this is only a small set of the available functions. There are currently more than 50 twm functions that allow the user to control the appearance of the screen, the keyboard focus, and the general operation of the window manager.

twm functions can be executed in three different ways:

1 From a pointer button press
2 From a keyboard key press
3 From a menu

A context in which to execute the function and any modifier keys needed may also be specified. The context simply specifies, with one exception, where the pointer needs to be positioned before the function executes. The following contexts may be specified with pointer button and keyboard key specifications.

root	Execute the function if the pointer is in the root window.
frame	Execute the function if the pointer is in the twm frame.
title	Execute the function if the pointer is in the title bar.
window	Execute the function if the pointer is in the client window.
icon	Execute the function if the pointer is in the associated icon window.
icon**m**gr	Execute the function if the pointer is in the icon manager entry.
all	Execute the function no matter where the pointer is.

The boldfaced letter of each context shows the one letter abbreviation for each context. In addition to these contexts, a double quoted string may also be specified for functions tied to a key on the keyboard. Modifier keys may also be specified that need to be pressed before the function executes. twm recognizes the following modifier keys:

shift	The Shift key
control	The Control key
meta	The meta key. The keyface will vary depending on what keyboard you are using.

Once again, the boldface letter indicates the one letter abbreviation that may be used. The format of a pointer button or keyboard key specification is as follows:

[*button* | *key*] = [*modifiers*] : *context* : *function*

Now for some examples:

```
Button1  =        : root   : f.menu "system menu"
Button2  =        : tfwi   : f.function "raise-lower-move"
"F1"     = shift  : t|f|w|i: f.raise
```

The first example pops up the menu "`system menu`" when pointer button one is pressed in the root window. The second executes the function stream "raise-lower-move" if the pointer is in a title bar, a frame, a client window, or an icon. The third example executes the **f.raise** function if the pointer is in a title bar, a frame, a client window, or an icon, and the F1 key is pressed while the Shift key is held down.

4.1 Function Streams

Function streams provide the user the ability to execute multiple functions with one action. A function stream is nothing more than a list of functions to execute. An invocation of a function stream can be seen in the previous section with the command **f.function**. We'll go through two examples on function streams to give an idea of how they might be used. The first example defines a function that raises the window and focuses the keyboard to it when pointer button one is pressed within the client window. This would be useful in setting up a *click-to-type* environment, where the user is required to click a pointer button in a window in order to enter text.

```
Button1=    : window : f.function "raise-n-focus"
Function "raise-n-focus" {
      f.raise
      f.focus
}
```

The second example shows an extremely useful one for those with a one button pointer. It simply implements a function that allows the user to raise, lower, or move the window with one pointer button pressed in the title bar.

```
MoveDelta 5
Button1=    : title: f.function "raise-lower-move"
Function "raise-lower-move" {
      f.move
      f.raiselower
}
```

When pointer button one is pressed in the title bar of any window the "`raise-lower-move`" function is executed. The **f.move** function is executed and if the pointer is moved more than five pixels, the window move is performed. If the pointer is not moved more than five pixels, the **f.raiselower** function is executed.

5 Variables

twm has a number of variables that control the appearence and operation of the window manager. The variables can be categorized into a few different groups:

- Color variables
- Cursor variables
- General appearence variables
- Icon variables

5.1 Color Variables

There are a number of variables that configure the color of twm created windows. Configureable areas include:

- Title bars
- Icons
- Icon manager entries
- Menus

Color variables are specified within a *Color List*. A color list begins with either **Color** or **Monochrome** and simply contains the individual color variables. This scheme allows you to use the same initialization file when running on either a color or monochrome system. In addition, most colors allow client or class specific colors to be specified so that groups of windows can share the same colors. For example:

```
Color {
      TitleForeground "black" {
            "XTerm"     "blue"
            "fred"      "green"
      }

      TitleBackground "white" {
            "XTerm"     "grey"
            "fred"      "yellow"
      }
}
```

This example shows several things, it first sets the default title bar foreground color to `black` and the default title bar background color to `white`. Following the defaults, we see some client specific settings. If a window were created that had a class or name of `XTerm`, the foreground and background colors of the title bar would be set to `blue` and `grey`, respectively. If a window is created that has a class or name of `fred`, the title bar foreground and background colors would be set to `green` and `yellow`. The name matching algorithm that twm uses matches a client name before a class name, so if a window were created that had a class of `XTerm` and a name of `fred`, the title bar foreground and background would be set to `green` and `yellow`.

5.1.1 Menu Colors

While the default menu colors are specified in a color list, menu colors can also be specified in the menu definition itself. This allows the user to have different colors for each menu item. The following example shows setting up of the default menu colors and also defining colors on a per entry basis in a menu.

```
Color {
      MenuForeground "black"
      MenuBackground "white"
      MenuTitleForeground "white"
      MenuTitleBackground "black"
}

Menu "An Example" ("white" : "red")       # this is the highlight color
{
      "Window Ops"      ("white" : "green")     f.title
      "Raise Window"                            f.raise
      "Lower Window"                            f.lower
      "(De)Iconify"                             f.iconify
      "Move"            ("white" : "red")       f.move
}
```

Setting up the default colors is fairly straightforward, let's look at the menu. The color specification for menu items has the form:

("*foreground*" : "*background*")

The color specified on the line with the `Menu` keyword specifies the colors to be displayed when the pointer is in a menu entry. We also see that two of the entries in the menu contain color specifications. These entries are displayed with the colors they have requested.

It gets better, twm also has the capability to interpolate colors between menu entries. If we were to put the keyword `InterpolateMenuColors` in our initialization file, our example menu would have entries with the background of each entry interpolated from `green` to `red`.

5.2 Cursor Variables

Variables exist to customize the appearence of the cursor (pointer image) when positioned within twm windows and when executing certain commands. The cursor may be taken from one of the predefined cursors in the cursor font or may be a custom cursor created from bitmap files. The following list specifies the cursors that can be customized and where/when the cursor is displayed.

- Frame the window frame
- Title in the title bar
- Icon in icons
- IconMgr in an icon manager
- Move during window movement
- Resize during a window resize operation
- Menu in a pop up menu
- Button in a title bar or icon manager button
- Wait when twm is busy
- Select when twm is waiting for the user to select a window for an operation
- Destroy when a window needs to be chosen to be destroyed

6 Icons

Icons are a concept supported by most window managers. An icon is simply a postage stamp sized window that is displayed instead of a larger client window. When a window is temporarily not needed, the user may choose to turn it into an icon thereby saving screen real estate. twm supports icons in two ways, traditional icons and through the use of an *Icon Manager*.

6.1 Traditional Icons

A traditional icon is typically a small window containing two distinct areas: an image area and a text area. The image area usually contains a small graphical representation of the client while the text area contains textual data that describes the client. A traditional icon can also have just a text area with no image area. Below are examples of traditional icons.

The two icons on the left illustrate icons with both an image area and a text area. The icon on the right is textual only.

Variables are available that allow customization of the appearence of icons. These variables allow the user to specify foreground, background, and border colors for the icons. You can also provide a list of bitmap files to use as the icon images.

Icons in twm are *active*. An active icon is one that enables you to position the pointer on top of the icon, type keys on the keyboard, and have those keystrokes directed to the client. This assumes of course that the keyboard focus has not been explicitly set to a certain client.

6.2 Icon Managers

In addition to traditional icons, twm introduces a new concept called an *Icon Manager*. An icon manager is simply a window where placeholders for clients are created and always displayed. An icon manager can be used in conjunction with traditional icons or in place of traditional icons. Icon managers have clear advantages over traditional icons. These advantages include the following:

- Group icon icons together
- Typically take up less screen real estate than traditional icons
- Allow an almost mouse-free environment to be created

About the only disadvantage to using an icon manager is that there is not a user definable image area in an icon manager entry. The following illustration is how a typical icon manager may appear:

This particular icon manager has three columns and currently contains eight entries, three of which are currently iconified. An icon manager is pretty much like any normal client window on the screen. If desired, it can have a title bar and can be moved, resized, raised, and lowered, like any other window. The icon manager can even be iconified! In the above illustration, the clients **myhost**, **xmh**, and one of the **xterm** windows are currently iconic. A window in an iconic state has a small glyph displayed to the left of the name, it is a replica of the iconify glyph used in the title bar. By default, a pointer button press in either the icon manager entry or the small iconify glyph either iconifies or deiconifies the client depending on its current state. The client named **george** shows a darkened area around its entry. This darkened outline indicates that it is the client that currently has the keyboard focus. As you move the pointer between clients, you will typically see the active icon manager entry also changing. You will also notice that the entries in this particular icon manager are sorted. The default behavior of twm is simply to place new entries at the end of the icon manager, but there does exist a variable that turns on a sorting function for the icon manager. There also exists a function to force an icon manager to be sorted. If the name of an entry were to change and the icon manager entries were being sorted, the icon manager would be resorted.

Icon manager entries are also active in the same way the traditional icons are active. If the pointer is positioned over an icon manager entry and keys on the keyboard are pressed, the characters are sent to the clients top level window.

twm supports multiple icon managers and each may have one or more columns. Multiple icon managers are useful for grouping windows together. For example you could have an icon manager that would hold all **xterm** windows, have another that would hold windows associated with a CAD package, and so on. The following example from an initialization file shows how you might set up multiple icon managers:

```
IconManagers {
      "Frame"     "=100x5+10+10"    1
      "XTerm"     "=300x5+120+10"   3
      "CAD"       "=200x5+500+10"   2
}
```

This example would cause three icon managers to be created, one that would have entries for windows with a class or name of “Frame,” one for windows that have a class or name of “XTerm,” and one for windows with a name or class of “CAD.” There is also a default icon manager created that can contain any entries not picked up by a specific icon manager. The number following the geometry specification is the number of columns that the icon manager should display. These icon managers will not become visible until they have at least one entry, and will be unmapped when they have no entries.

6.2.1 An Almost Mouse-free Environment Using Icon Managers

As we stated in the previous section, it is possible to setup an environment where the pointer would only be needed to move and resize windows, and to pop up menus. Just about every other twm function can be executed from the keyboard. One of the contexts available to define keyboard and pointer functions is **iconmgr**. This context is used when the pointer is positioned in an icon manager entry. When a function is executed with this context, the operation is performed on the corresponding client window. Let’s take for example the following keyboard function definition:

```
"F1" = : iconmgr  : f.raise
```

This key definition would cause the client window to be raised when the pointer is in an icon manager entry and the F1 key is pressed.

Now let's setup our mouse-free environment using some of the other available functions:

```
NoTitleFocus
"R3" =            : all       : f.warpto "XTerm Icon Manager"
"Up" =            : iconmgr   : f.upiconmgr
"Down" =          : iconmgr   : f.downiconmgr
"Left" =          : iconmgr   : f.lefticonmgr
"Right" =         : iconmgr   : f.righticonmgr
"Left" = shift    : iconmgr   : f.previconmgr
"Right" = shift   : iconmgr   : f.nexticonmgr
"F1" =            : all       : f.raise
"F2" =            : all       : f.lower
"L1" =            : all       : f.iconify
```

6.2.2 Summary

Now let's see what we've got. Regardless of the pointer's current position when the R3 is pressed, the pointer is warped to the XTerm icon manager window. Now that we're in the XTerm icon manager, the ↑, ↓, →, and ← keys have been defined to allow you to move around the icon manager. Because the icon manager entries are active, you can simply type on the keyboard while the pointer is in an entry and the keystrokes are directed to the client.

We have also set up the F1 key to raise the window, the F2 key to lower the window, and the L1 key to iconify the window. These functions are executed when the pointer is in an icon manager entry. The user can also use those keys when the pointer is positioned in the title bar or main window of a client. Additionally, the ← and → keys can be used in conjunction with the Shift key, to warp the pointer between icon managers. These functions will not only warp between icon managers on a single screen but will also warp the pointer to visible icon managers on other screens being managed by twm. The symbols used here for keyboard keys, may or may not be present on your keyboard.

7 Conclusion

twm fills a void in the standard X Version 11 release because it provides the novice X user with a friendly easy to use interface to the X Window System. twm also provides advanced features for the more experienced user.

Because it is not based on a toolkit or widget set, server resources have been kept to a minimum as well as obtaining the best possible performance.