X Window Manager: Technical Specification

Third Year Project

By Ben Sudzius and David Craig

0. Table of contents

1. Introduction

1.1 Overview

The product that was developed is an X window manager for the Linux operating system. Conveniently it is called X Window Manager. As the name suggests, this program controls and manages the windows that are displayed on the monitor display of a computer running Linux.

As X Window Managers main intention is to control the windows that are displayed by client applications, it handles the resizing and movement of such windows. It does this through user input and a selection of keys, mouse buttons, and mouse movements. The window manager manages how keyboard and mouse inputs are interpreted through the use of event masks.

X Window Manager is a reparenting window manager. This essentially allows it to place decorations on top of client windows. X Window Manager creates a frame for all top level windows(clients) which are windows in of themselves. Within this frame it places a title bar for each client window that contains a close button and a maximise button. Non reparenting window managers are incapable of doing this.

The window manager uses the X Window System (X11) to intercept calls made by the client program to the X server. It decides how these calls will be handled and what requests it will honor. The window manager dictates how the client programs are displayed and manipulated on the screen. Because it is essentially a GUI, it uses an event loop where all functions are handled with X library calls until the user quits the program.

1.2 Glossary

X Window System (X11) - X11 provides the basic framework for interacting with bitmap display of a monitor on a high level of abstraction. It provides mechanisms for drawing and moving windows on a screen and how the input devices(mainly a keyboard and mouse) interact with the display.

Client - In this report anything referred to as a client will be an application that was opened in the window manager.

X Library (Xlib) - Library that contains functions for interacting with the X server which allows the programmer to create X programs without knowing the low level details of the protocol. Developed by the X.Org Foundation in 1985.

Reparenting window manager - A reparenting window manager creates a parent window for all top level windows in the X Window System. Therefore every top level window becomes a child and the reparented window becomes it's parent. Is used to create frames and title bars/buttons.

Events - Events that are masked are reported to the window manager which then dictates how to deal with them. Events include keyboard presses, mouse clicks, window destruction and a whole range of other things necessary for a functional window manager to be notified of.

Event masks - A "mask" that filters the type of events that a window manager wants to deal with.

Mapping - The process of displaying a window on the screen, that has been created and configured. Similarly unmapping is the process of hiding a window from the display.

Call - A function call from the X Library to the X Server that allows X11 programs be created and managed.

2. System Architecture

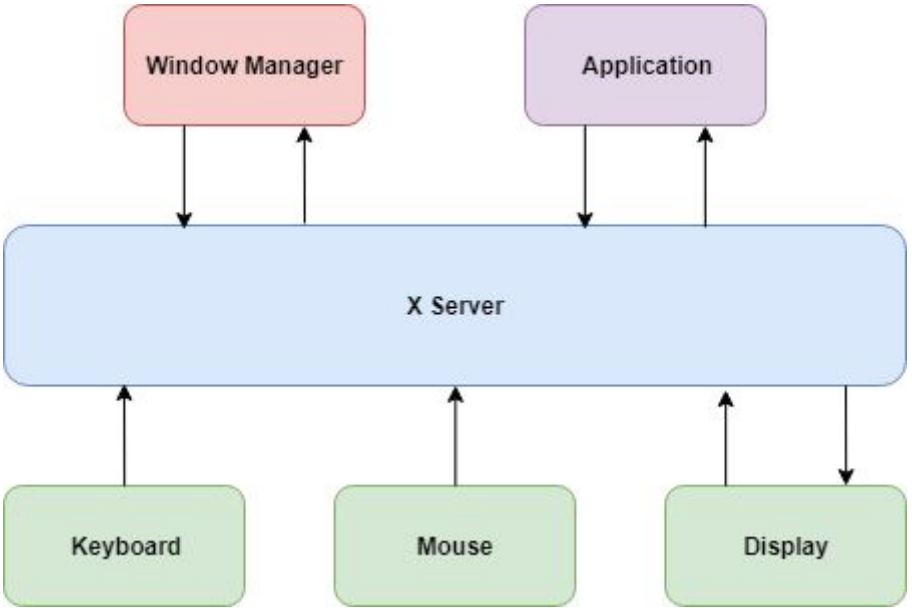

Fig 2.1

Fig 2.1 seen above illustrates the overarching architecture of the product. It shows how all inputs are all handled by the X Server. Specified inputs (events) from the application, keyboard, mouse, or display are sent to the window manager which then chooses how to handle and react to them.

2.1 Window Manager

The window manager specifies certain masks, based on the type of events it want to receive from the X Server. The two main types of events are requests and notifies. Intercepted requests have not yet been completed and allow the window manager to act upon them. Intercepted notifies have already happened and tell the window manager what action has been performed. The window manager is able to manipulate the windows with the use of these events.

2.2 Keyboard and Mouse

Masked keys and buttons events sent to the X Server are reported to the window manager. This allows the window manager to run functions such as closing or resizing a window.

2.3 Application

Application sends requests and notifies based on what X Library call it makes. This lets the window manager know what to do.

2.4 Display

Displays all of the information reported by the X Server such as displaying windows and resizing them. Returns the results displayed.

3. High-Level Design

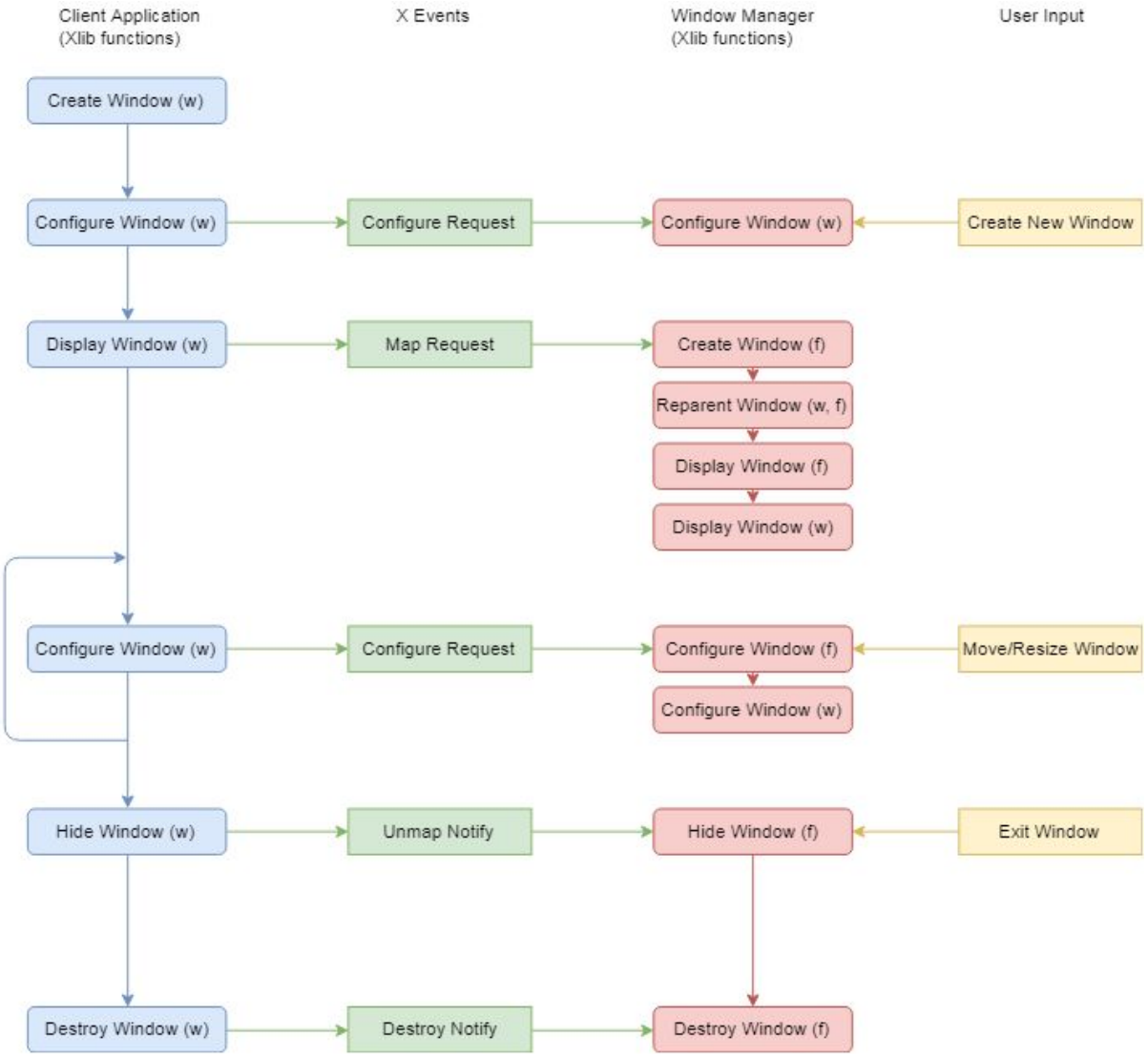

Fig 3.1 Event Loop

Fig 3.1 shows the basic event loop and how the window manager and client applications interact.

3.1 Basic Event Loop Diagram

The above diagram shows the basic event loop of our program. Similarly to other GUI's thewindow manager runs an event loop and deals with events as they appear. It shows how a window gets created and reparented and what happens when a user tries to close or move a window. There are many more functions that deal with motion and key presses but this is the basic life cycle of a client window.

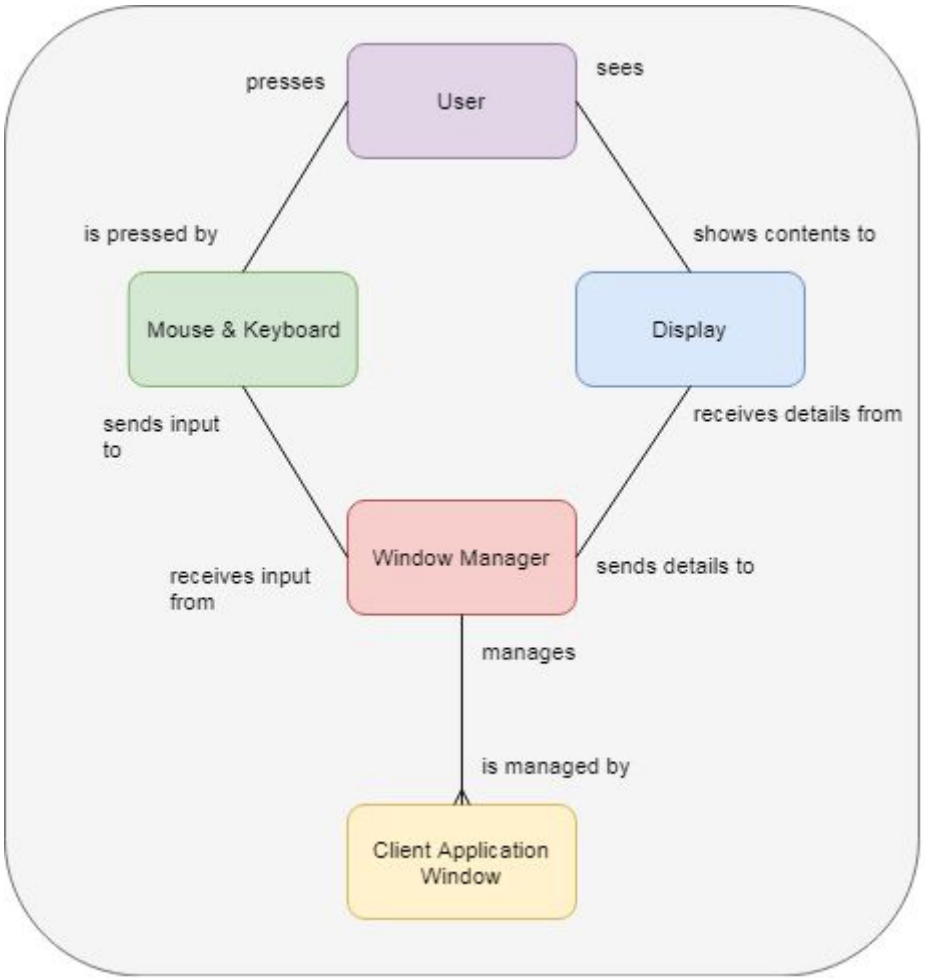

Fig 3.2 Logical Data Structure Diagram

Fig 3.2 is Logical Data Structure Diagram of the window manager.

4. Problems and Resolution

4.1 Using Python

The biggest hurdle we faced in this project was trying to use the Python programming language. The code for an X11 window manager is less about the programming language and more about how you handle the X Library. Python's X Library is severely lacking in well written documentation, and the lack of Python window managers made it extremely difficult to start understanding and writing code. Although we knew the theory behind how the X Server worked, translating it to code proved difficult and severely hampered progress.

Most X11 window managers are written in C, which led to the decision of switching the programming language. Thanks to Volume One: Xlib Programming Manual, good documentation, and the huge range of examples we were able to start progressing steadily. We later moved to C++ for their use of unordered maps, which was an easy port as the code stayed very similar. Although project began later than anticipated, the theory learned before hand proved to be invaluable.

4.2 Reparenting Windows

Reparenting windows was one of the biggest challenges of this project. It involves getting all top level client windows and adding a parent window on top. This caused several problems as figuring out which window was already reparented and knowing when to unframe clients when they unmapped and closed. It also introduced many new challenges as the window manager no longer had to just worry about the clients, but now had to deal with the frame and all associated windows.

Fixes included the addition of and unordered map, the use of cout to check code issues, using XQueryTree(), and numerous revisions to the code. The issues and solutions are explained more in depth below.

4.3 Window Resizing

A problem that persisted build of the program was the issue of resizing. Bugs such as the program crashing, the frame and title not resizing, and the client windows not resizing. There were several fixes throughout the lifetime of the project. A huge issue was that windows with the reparented frame would not resize. The discovery of the XQueryTree() call helped solve many issues as it allowed us to select the parent and children of specified windows. This meant we were able to select all the children of the frame window, which included the title bar, client application, and title buttons.

There was another issue of the client window not resizing itself. It would resize but the contents within the client window itself would not. After debugging the code and reading up on it it turned out that after the client got reframed, the event mask for it's configuration was not set. Enabling this allowed the clients to resize itself correctly.

4.4 Destroying Windows and Unframing

In the early stages of closing windows, the client itself would close but it would leave the it's frame behind. There were several ideas that were proposed but the one we went with was using an unordered map to map each frame to a client window. This allowed us to destroy any frame that is associated with a client window.