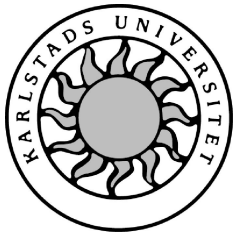


Datavetenskap

Magnus Persson, Per Nordgren

Fönsterhanterare för X-Windows

Examensarbete

2004:27

Fönsterhanterare för X-Windows

Magnus Persson, Per Nordgren

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

_____
Magnus Persson

_____
Per Nordgren

Godkänd, 2004-06-04

_____
Handledare: Stefan Alfredsson

_____
Examinator: Martin Blom

Sammanfattning

Varje dag använder de flesta något som kallas fönsterhantare vid datoranvändning. Grafiska applikationer till operativsystem har blivit vanligare och därmed också fönsterhanterare. En fönsterhanterare är det som hanterar alla fönsterhändelser, som till exempel att flytta eller förstora ett fönster. Utan fönsterhanterare skulle det till exempel vara svårt att använda grafiska system som baseras på fönster. Detta eftersom det då inte skulle gå att till exempel flytta, förstora eller minimera fönster. Innebörden skulle bli att system baserade på fönster skulle bli oanvändbara. Linux är ett av de operativsystem som växer snabbast, både med avseende på vidareutveckling av dess funktionalitet och på antalet användare. Det är ett av de operativsystem som är mest tillgängliga, eftersom det är gratis att ta hem från Internet. Detta är ett av skälen till varför Linux valdes som utvecklingsmiljö. Även om fönsterhanterare för Linuxsystem är vanliga så är syftet med detta arbete att skapa en fönsterhanterare som endast tillhandahåller den viktigaste funktionaliteten. Exempel på viktig funktionalitet är att kunna flytta, förstora och förminska fönster. Denna rapport beskriver hur arbetet gått till från planering till färdig fönsterhanterare. Rapporten innehåller information om vad X-Windows är och hur det fungerar, även vad Xlib är och en detaljerad information om fönsterhanterarens uppbyggnad.

A Windowmanager for X-Windows

Normally during interaction with a graphical computer user interface, a windowing system is used. These windows are controlled by a window manager. The window manager handles events such as moving, resizing, minimizing windows etc. Without a window manager it would be difficult to use a graphical user interface that is based upon windows. Linux is one of the fastest growing operating systems, which uses the windowing system X11. Although there already exist a number of window manager applications for X11, this project aimed at creating a window manager with only the most important functionality. This report describes how the project was done from planning into a real window manager that could be used to work with. It also includes information about X-Windows and Xlib which was a big part in the project.

Innehåll

Figurer

1 Inledning

I detta kapitel förklaras olika begrepp som kommer att användas i rapporten. Vidare beskrivs även bakgrund, syfte och mål med både projekt och rapport samt de avgränsningar som har gjorts. Även en översikt av rapportens upplägg presenteras.

1.1 Begreppsdefinition

Begrepp som används i rapporten och som behöver definieras är *fönsterhanterare*, *klient*, *applikationsfönster*, *ramfönster*, *rotfönster* och *KauWM*.

En *fönsterhanterare* är en applikation som hanterar andra applikationers grafiska upplägg på skärmen. När en applikation startas ser fönsterhanteraren till att applikationen visas i ett eget *applikationsfönster*. Detta fönster som innehåller applikationen kan sedan manipuleras med hjälp av fönsterhanteraren. Några olika exempel på hur ett fönster kan manipuleras är förflyttning, förstoring, förminskning och nedstängning av fönstret. Ett *ramfönster* används för att göra dessa manipulationer lättillgängliga för användaren. Endast applikationer som tillhandahåller någon form av grafiska användargränssnitt ska läggas in i ett fönster och visas på skärmen. Fönsterhanteraren är ett exempel på en applikation som inte ska visas på skärmen och därmed inte heller behöver något applikationsfönster eller ramfönster. I rapporten kommer ord som *klient* och *fönster* att användas. Skillnaden mellan dessa är att en klient består av applikationsfönster och ramfönster medan ett fönster kan vara ett applikationsfönster, ett ramfönster eller något annat fönster som inte är bundet till en klient. Exempel på ett fönster som inte är bundet till en klient är ett menyfönster. Rotfönstret är det fönster som alltid finns då X-Servern är startad (för mer information om X-Servern se kapitel 2.2). Det finns alltså även om ingen fönsterhanterare är igång. Rotfönstret hanteras av X-Servern tills en fönsterhanterare startas av användaren. Då blir istället fönsterhanteraren ansvarig för rotfönstret. Rotfönstret är det fönster som täcker hela skärmen och kallas skrivbord eftersom det alltid ligger under alla andra fönster och är grundfönstret. Rotfönstret benämns även som skrivbord i rapporten.

Fönsterhanteraren som utvecklas kallas för KauWM (Karlstad university WindowManager). I rapporten kommer fönsterhanteraren benämnas som KauWM, för att undvika missförstånd med andra nämnda fönsterhanterare.

1.2 Bakgrund

Det finns många olika fönsterhanterare för Linux. De flesta innehåller mycket funktionalitet, vilket kan medföra att de blir komplicerade att använda. Som exempel finns KDE[6] som har det mesta som kan tänkas inom funktionalitet. Ett exempel på denna funktionalitet är virtuella skrivbord, som möjliggör användandet av flera skrivbord samtidigt. Detta kan vara användbart om flera applikationer används samtidigt, då det blir möjligt att visa några applikationer på varje skrivbord. Ett annat exempel på funktionalitet är (utan närmare beskrivning) menyer som aktiveras vid musklick på ramfönstret. Det vi vill göra är att utveckla en fönsterhanterare som endast tillhandahåller den viktigaste funktionaliteten. Exempel på viktig funktionalitet för en fönsterhanterare är att kunna flytta, förstora och minimera fönster. En första tanke var att skapa en fönsterhanterare för Microsoft Windows[9] eftersom det inte finns många att välja på för det operativsystemet. Vi insåg dock att det skulle bli svårt att skapa en fönsterhanterare för Microsoft Windows eftersom det finns för lite information inom området samt dåligt med tillgängliga källkoder. Det finns däremot mycket information om fönsterhanterare för Linux. Det finns även mycket information om Xlib[1] och X-Windows[7] som de flesta fönsterhanterare för Linux är uppbyggda av. Det var detta som fick oss att välja att utveckla en fönsterhanterare för X-Windows under Linux. Eftersom Linux är ett ständigt växande operativsystem (som nämndes i sammanfattningen) så är det bara bra att ha fler fönsterhanterare att välja bland.

1.3 Syfte och mål

Det primära syftet är att skapa en enkel, snabb och användarvänlig fönsterhanterare för Linuxsystem. Den ska tillhandahålla den viktigaste funktionaliteten hos en fönsterhanterare, det vill säga

att skapa fönster till de applikationer som startas och att sedan kunna manipulera dessa fönster på ett enkelt och bra sätt. Fönsterhanteraren som utvecklas kommer att ha viss funktionalitet som finns hos de flesta fönsterhanterare, till exempel förstora, förminska och ikonifiera fönster. Arbetet har även influerats av WindowMaker, till exempel som att visa ett fönster med endast ramen synlig (borderfierat). Syftet är även att lära oss hur en fönsterhanterare är konstruerad.

Målet med KauWM är att skapa en fungerande fönsterhanterare. KauWM ska ha grundläggande fönsterfunktionalitet samt gå att arbeta med. Rapportens mål är att beskriva på ett enkelt sätt vad en fönsterhanterare är och hur den är uppbyggd. Rapporten ska även beskriva hur utvecklingen av fönsterhanteraren sker.

1.4 Avgränsning

En avgränsning som kommer att införas är att bakgrundsbild eller bakgrundsfärg inte kommer att kunna ändras på skrivbordet. Virtuella skrivbord kommer heller inte bli implementerade och det kommer inte finnas något stöd för andra fönsterhanterare. En avancerad meny som exempelvis startmenyn i Microsoft Windows anses inte nödvändigt hos en fönsterhanterare i den storleksordning som är tänkt i detta projekt. Endast en grundläggande meny med ett begränsat antal applikationer är tänkt att implementeras. Detta för att KauWM inte kommer att tillhandahålla lika stor funktionalitet som exempelvis fönsterhanteraren i Microsoft Windows.

1.5 Upplägg av rapporten

Rapporten som följer är uppbyggd enligt följande. Kapitel 2 förklarar grunderna till det som behövdes för utveckling av KauWM. Kapitel 3 tar upp de delar som har med användandet av KauWM att göra. I kapitel 4 beskrivs hela KauWM i detalj inklusive dess funktionalitet. I kapitel 5, som är det sista, förklaras de problem som uppkom under projektets gång. Dessutom ges förslag och diskuteras funderingar kring vidareutveckling av KauWM.

2 X-Windows

I detta kapitel förklaras begrepp som X-Windows, X11, Xlib och klient-server modellen. Dessa delar är de som KauWM grundar sig på.

2.1 X-Windows och X11

X-Windows tillhandahåller basen för ett grafiskt användargränssnitt. Detta avser exempelvis uppritning på skärmen, att kunna flytta fönster och att hantera mushändelser. Tanken med X-Windows är att ha en gemensam bas så att applikationer kan användas på många olika datorer och inte bara på en viss tillverkares datorer. Genom att ha X-Windows som grund behöver inga stora förändringar ske för att en applikation ska fungera på olika datorer. De applikationer som är utvecklade för X-Windows kommer att kunna användas med alla de olika fönsterhanterare som finns för X-Windows.

X-Windows är ett system som utvecklades under 1980-talet på MIT (Massachusetts Institute of Technology). Syftet var att utveckla ett grafiskt användargränssnitt som kunde användas över nätverk. Det var först och främst tänkt att användas i UNIX-miljöer. MIT hade vid denna tiden flera olika system, vilket gjorde att de inte kunde koppla samman dessa. De olika systemen använde olika tekniker för att rita upp grafik på skärmen. De behövde ett system som var plattformsoberoende och som kunde koppla ihop deras datorer och startade därför 1984 utvecklingen av X-Windows. 1986 släpptes version 10 även kallad X10, som var den första versionen av X-Windows som fick stor utbredning. 1987 släpptes version 11 även kallad X11, som idag är den aktuella versionen av X-Windows. Efter detta har flera olika grupper ansvarat för vidareutveckling av X-Windows och det har släppts flera nya releaser av version 11.

2.2 Server och klient

X-Windows är baserat på en klient-server modell som går ut på att dela upp ansvaret för beräkningar och uppritning på olika datorer. På detta sätt kan beräkningar ske på en dator (kallad klient) och

uppritning på skärmen kan ske på en annan dator (kallad X-Server), som antingen är sammankopplad med klienten i ett lokalt nätverk eller över Internet. Det går även att köra X-Server och klient på samma dator. En klient i detta sammanhang bör inte blandas ihop med klient i fönstersammanhang, då detta är två skilda saker. Vad som kallas klient och X-Server kan kännas förvirrande i början, eftersom det är tvärtom mot det normala scenariot där en klient används för att koppla upp sig mot en server. I X11's klient-server modell kontrollerar X-Servern skärmen och sköter all inmatning och uppritning m.h.a. tangentbordet, musen eller skärmen. Klienten tar sedan hand om alla sorters beräkningar, som till exempel att köra applikationer. När något sedan ska visas på skärmen säger klienten till X-Servern vad som ska ritas upp. En fördel med denna uppdelning är att en snabb dator kan användas som klient och sköta alla beräkningar. Sedan kan uppkoppling mot klienten ske med flera olika X-Servrar som inte behöver vara så kraftfulla eftersom de bara ritar upp enkel grafik. En användare kan även ha flera klienter igång på olika datorer för att få en utspridd arbetsbelastning.

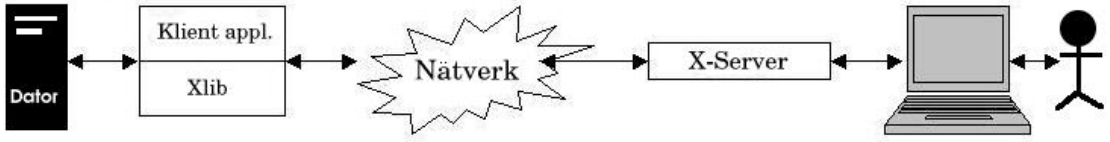
2.3 Xlib

Xlib är ett programmeringsgränssnitt för X-Windows som bland annat tillhandahåller funktionalitet för:

- Färger
- Muspekare
- Uppkoppling mot X-Server
- Uppritning av enkla objekt (t.ex. rektanglar, text, linjer)
- Felhantering
- Händelsehantering
- Typsnitt

- Tangentbordsfunktioner
- Skapande av fönster samt borttagande av fönster

Eftersom X-Windows konstruerades för att användas i nätverk så skickar Xlib alla förfrågningar över nätverket till X-Servern (Figur 2.1 visar hur detta sker). Bilden visar hur en applikation använder sig av Xlib som i sin tur skickar data över nätverket till X-Servern. Detta visualiseras på skärmen hos den dator som X-Servern körs på. Även när Xlib och X-Server körs på samma maskin skickas data med X protokollet. Detta sker genom en intern kanal istället för över externt nätverk.



Figur 2.1: X-Windows med klient-server modell.

X protokollet tillhandahåller förfrågningar, svar, händelser och felhantering. Alla protokollförfrågningar genereras av Xlib till X-Servern. Dessa förfrågningar kan till exempel vara att ändra färg på ett fönster, ändra storlek på fönstret eller rita upp en linje på skärmen. De flesta funktionsanrop i Xlib skickar protokollförfrågningar. De som inte gör detta hanterar datastrukturer internt som inte X-Servern behöver känna till. Ett protokollsvar skickas från servern till Xlib då något ska rapporteras som svar på förfrågningar. Alla förfrågningar får inte svar, till exempel de som skickas för att uppritning på skärmen ska ske. De förfrågningar som kräver svar kallas för *round-trip requests* och bör inte användas för ofta i klientapplikationer eftersom det sänker prestandan hos klienten. En händelse skickas från X-Servern till Xlib då exempelvis ett tangentnedtryck eller en förflyttning av musen har inträffat. Dessa händelser hamnar i en händelsekö, som sedan Xlib läser ifrån.

Felmeddelanden som kommer från X-Servern berättar för Xlib att en tidigare förfrågan resulterade i ett fel. Felmeddelanden kan hanteras på två sätt av Xlib. Endera hanterar Xlib dem genom att skriva ut felet och avsluta, eller så kan fönsterhanteraren implementera en funktion

som hanterar felmeddelanden på ett mer effektivt sätt. Xlib buffrar alla förfrågningar som sker av klienten och skickar dem sedan som paket. Detta görs för att slippa begära tillgång till nätverket vid varje förfrågan. Klienten kan dock begära att Xlib ska skicka bufferten med förfrågningar från klientapplikationen direkt istället för att vänta på att Xlib ska göra detta. Xlib buffrar både inkommande och utgående förfrågningar respektive svar. Detta gör att uppritning inte sker förrän bufferten har blivit skickad. Detsamma gäller för felmeddelanden, dessa uppkommer inte förrän X-Servern har läst av förfrågningarna. Eftersom det kan ta tid för X-Servern att läsa förfrågningarna och generera felmeddelanden så kan klienten ha hunnit utföra andra operationer under denna tid. Detta kan göra det svårare att felsöka program eftersom klienten inte vet vilken operation felmeddelandet kommer ifrån.

Xlib tillhandahåller något som kallas för *property*, rakt översatt till egenskap. Varje fönster har ett antal egenskaper som alla klienter som körs på X-Servern har tillgång till. Egenskaperna kallas för *atomer* i Xlib. En atom är ett unikt ID nummer för en viss egenskap. Till exempel så är atomen *XA_WM_NAME* namnet på en klient. I Xlib är det möjligt att skapa egna atomer eller använda de som redan finns definierade. De som redan finns börjar med *XA_*.

3 Översiktliga beskrivningar av KauWM

I detta kapitel beskrivs hur KauWM kan användas. Exempelvis beskrivs hur exekverbar fil skapas från källkod för att starta KauWM. Även vissa konfigurationsmöjligheter beskrivs.

3.1 Övergripande om KauWM

KauWM är en klient till X-Servern. Skillnaden mellan KauWM och andra klienter till X-Servern är att KauWM hanterar de andra klienterna som körs mot X-Servern samt att den ej har applikationsfönster eller ramfönster. Detta sker genom att bestämma var de ska placeras på skärmen samt storleken på varje klientfönster. Detta är inte det enda som KauWM gör utan den har många andra funktioner som skiljer sig från en vanlig klient. Med en vanlig klient menas att en klient blir visualiserad av X-Servern. KauWM hanterar rotfönstret och sätter attribut som det ska ha, exempelvis vilka händelser som rotfönstret ska ta emot. Exempel på händelser är musklick och tangentbordsnedtryck. Vad som bör uppmärksammas är att KauWM inte hanterar sig själv som en klient och därför inte finns med i listan över anslutna klienter(se kapitel 4.4). Detta beror på att KauWM inte visualiseras på X-Servern, vilket övriga klienter gör. KauWM består av olika delar. Den första delen är initieringen av KauWM. Här skapas olika grafikrutiner samt allokering av den färg som ska användas för varje grafikrutin. Även typsnitt samt musmarkörer som ska användas initieras här. Den andra delen är en upprepning som läser av alla händelser från händelsekön. Det är denna upprepning som är huvuddelen i KauWM.

Den tredje delen i KauWM är sökning efter klienter i en länkad lista (se kapitel 4.4). För varje händelse som sker, söks en klient som händelsen är kopplad till. Sökningen sker i den länkade listan där pekare till alla klienter som är synliga och aktiva på skärmen finns. Då rätt klient har hittats anropas en motsvarande funktion till händelsen hos klienten. Klienten i sin tur utför det som förväntas. När detta skett fortsätter upprepningen att läsa av nästa händelse i kön och allt upprepas på nytt. Denna upprepning slutar aldrig under KauWMs exekvering. I huvudklassen finns en initieringsfunktion (se kapitel 4.1) som skapar vissa variabla objekt som diskuteras i det

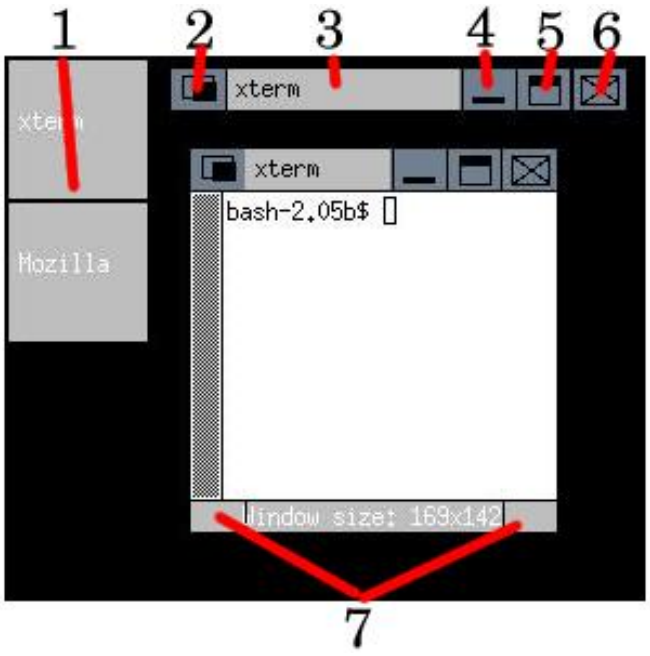
hänvisade kapitlet. Vid denna initiering sätts en felhanteringsfunktion med en fördefinierad Xlib funktion. Denna felhanteringsfunktion är implementerad att endast skriva ut vilket fel X-Servern rapporterade.

KauWM är uppbyggd av olika klasser i C++[8]. En klass kallas *WManager* och är huvudklassen. Denna klass hanterar huvudloopen, initieringen, grafikrutiner, typsnitt och musmarkörer samt andra grundläggande funktioner. En annan klass är klientklassen vid namn *Client*. Klientklassen hanterar all den funktionalitet som varje klient har. Dess funktioner hanterar olika händelser som uppritning av ramen runt varje klient, förflyttning, förminskning, nedstängning, förstoring samt andra användarstyrda händelser. Tills sist finns klassen vid namn *Icon*, som är en klass för ikonsystemet. Denna klass används för att skapa ikoner och hantera dessa. En ikon skapas då användaren klickar på ikonifieringsknappen längst upp till vänster på ramen till ett fönster. Ikoner som skapas placeras längst upp till vänster på rotfönstret. Om flera ikoner skapas placeras de under den första ikonen som skapades (se figur 4.10).

3.2 Funktionalitet

Den funktionalitet som finns för en användare är att stänga ner varje klient med en nedstängningsknapp tillhörande varje klient. Användaren kan maximera en klient så att den täcker hela den tillgängliga skärmytan. Användaren kan även minimera en klient så att den inte blir synlig annat än i en lista eller att kunna minimera en klient till ikonläge. Ett annat sätt att förminska en klient är att borderfiera klienten så att endast ramen syns men har kvar den funktionalitet som en klient har. Det enda som en klient i borderfierat läge saknar är förstoring och förminskning i de nedre hörnen på klienten (de hörnen syns inte i borderfierat läge). Användare kan även förflytta klienter på skärmen, genom att klicka på och hålla nere vänster musknapp. Som användare går det även att klicka fram den klient som användaren vill ha fokus på och arbeta i, vilket betyder att hela klienten blir synlig. Längst ner i varje hörn på en klient kan användaren dra för att förstora eller förminska en klient (se figur 3.1). För varje beskriven funktionalitet ovan finns en knapp på ramen till klienten (se figur 3.1). I ramen visas även namnet på klienten. Ikonläge är ett läge

där klienten minimeras och visas som en ikon på rotfönstret (se figur 3.1). Denna ikon går att klicka på för att klienten ska hamna på skärmen igen. Vid vanlig minimering skapas ingen ikon utan klienten hamnar endast i en lista som kan visas genom att klicka med vänster musknapp på rotfönstret (se figur 4.2). Det går att starta applikationer i KauWM genom att högerklicka på rotfönstret. Då kommer en applikationsmeny fram med flera applikationer som går att starta (se figur 4.1). Med muspekaren görs valet av applikation som ska exekveras. Vid klick utanför menyn försvinner endast menyn utan någon applikationsexekvering.



Figur 3.1: Bild på funktionaliteterna hos KauWM.

1. Ikonerna hos KauWM
2. Knappen för att ikonifiera en klient
3. Förflyttning och borderfiering av en klient
4. Knappen för att minimera en klient
5. Knappen för att maximera en klient
6. Knappen för att stänga ner en klient
7. Fält för att förminska och förstora en klient

3.3 Konfigurationsmöjligheter

De konfigurationsmöjligheter av KauWM som finns i nuläget är begränsade. Ingen konfigurationsfil finns implementerad så den konfiguration som kan göras, görs i huvudheaderfilen (main.hpp) för KauWM. Det som kan ställas in där är bland annat ikonstorlek och hur fönster ska ritas upp vid förflyttning, förstoring och minimering. De två alternativen är att rita upp hela fönstret då förflyttning sker (realtidsuppdatering) eller att endast rita upp en ram för fönstret (rammarkör). Ikonstorleken är satt till 64x64 pixlar, vilket brukar passa de flesta upplösningar.

3.4 Kompilering

För att kompilera en version av KauWM så används applikationen *make*. En makefile finns inkluderad i källkodskatalogen. I terminalläge skrivs *make* och en exekverbar fil kallad *KauWM* skapas. Denna startas genom att skriva `./KauWM` i samma katalog. KauWM kan inte startas då en annan fönsterhanterare är igång. För att starta KauWM så kan uppstart av X-Windows ske i *failsafe* läge. När X-Windows körs i *failsafe* läge så körs nämligen ingen fönsterhanterare. I *failsafe* läge utförs sedan instruktionerna som ovan. Om KauWM ska användas permanent kan filen `.xinitrc` editeras som ligger i hemkatalogen för aktuell användare. Finns inte denna fil så kan den skapas. Innehållet ska vara `exec /path/to/wm/KauWM`. De bibliotek som används vid kompileringen är *X11*. För att kompilera utan felsökning, så ska `-DDEBUG` avmarkeras efter `DEFINES` i makefilen.

4 Detaljerade beskrivningar

En detaljerad beskrivning av KauWM ges i detta kapitel. Grunden till KauWM beskrivs först, därefter klienthantering, ikonsystemet och sist hur lagring av klienter sker med hjälp av en lista.

4.1 Fönsterhanterarsystemet

Fönsterhanterarsystemet består av klassen *WMManager*. Det första som anropas vid start är initWm(). Denna funktion har hand om initieringen av KauWM. De delar som initieras här är:

- Grafikkontext
- Typsnitt
- Muspekare
- Skärmen
- Rotfönster
- Atomer

De grafikkontext (uppsättning av färger samt fonter) som initieras är de som används av klienterna för att rita upp grafik på fönsterramarna. Därefter anropas av initieringsfunktionen addActiveWindows(). Denna funktion har som uppgift att skapa klienter av applikationerna som redan körs, för att sedan lägga in pekare till dem i den länkade listan (se kapitel 4.4). Det funktionen gör är att leta igenom det träd som Xlib skapar av alla applikationer som är kopplade till X-Windows. Xlib bygger nämligen upp en trädstruktur av de klienter som körs mot X-Servern, detta för att till exempel veta vilka fönster som överlappar andra fönster. Om en applikation har rätt attribut (till exempel att det är angett att det ska vara ett fönster och inte bara köras i bakgrunden), så skapas en ny klient och en pekare till denna sparas undan i den länkade listan. Om allt detta går som det ska så returneras INIT_WM_OK tillbaka till huvudfunktionen (main). Därefter anropas funktionen mainLoop(). Det är denna funktion som är kärnan i KauWM. Den består av en oändlig

upprepning som tar en händelse från X-Windows händelsekö. Vilken typ av händelse det är läses av och motsvarande funktion till händelsen anropas. De händelser som stödjs är:

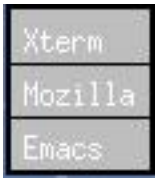
- ButtonPress
- ButtonRelease
- MapRequest
- ConfigureRequest
- PropertyNotify
- KeyPress
- DestroyNotify
- EnterNotify
- Expose

ButtonPress händelse sker då en musknapp (vilken som helst) trycks ned. Efter det så släpps förhoppningsvis musknappen och en *ButtonRelease* händelse sker. I detta läge används endast *ButtonRelease* händelsen i klienterna vid förflyttning av fönster. En *MapRequest* händelse sker då en ny applikation startas som ska visualiseras som ett fönster. Detta sker även då ett fönster skapas om på nytt (till exempel från minimerat läge tillbaka till normalt läge då hela fönstret är uppritad på skärmen). *ConfigureRequest* inträffar då en ny applikation vill skapa ett fönster med en viss storlek och position på skärmen. Klienten får den storlek samt position som applikationen efterfrågar. *PropertyNotify* sker till exempel vid namnbyte hos en klient. Exempel på detta är ett grafiskt terminalfönster (även kallat Xterm) som byter namn till den sökväg som användaren befinner sig i hos terminalfönstret. Vid en sådan händelse uppdateras namnet på ramen runt fönstret. Som namnet antyder så är en *KeyPress* händelse något som sker vid tangenttryckning. *DestroyNotify* sker då ett fönster ska stängas ner, detta stänger dock inte ner applikationen direkt

(förklaras i kapitel 4.2). *EnterNotify* sker varje gång muspekaren kommer över ett fönster. *Expose* sker exempelvis vid musklick på ett fönster för att få fokus på det aktuella fönstret. För mer information om vad som sker vid varje händelse, se kapitel 4.2 *Klientsystemet*. Efter varje inläsning och hantering av en händelse i huvudloopen fortsätter en ny inläsning från händelsekön. En av de viktigaste funktionerna i basklassen (WManager) som har beskrivits i kapitel 3.1 är findClient(...). Denna funktion söker genom den länkade listan efter den klient som tillhandahåller det fönster som en händelse uppträder i. Pekare till denna klient returneras sedan. Ett fönster som en händelse uppträder i kan vara ett ramfönster, applikationsfönster eller ikonfönster. Alla funktioner som hanterar de händelser som beskrivs ovan använder sig av findClient(...). De funktionerna gör inte mycket mer än att söka upp rätt klient samt anropa motsvarande funktion i klienten. Några av de funktioner som hanterar händelserna kontrollerar dock ifall en klient är ikonifierad. Den funktion som hanterar musklick för fönster kontrollerar först ifall musklickningen inträffade på rotfönstret innan den söker upp ett annat fönster. Detta görs eftersom rotfönstret inte är en klient utan hanteras för sig. Vad som händer då klick i rotfönstret sker beskrivs i nästa stycke.

Resterande funktioner som finns i huvudklassen är redrawAll() som uppdaterar grafiken på alla klienter som KauWM hanterar. removeClient(...) tar bort pekare till klient i länkade listan. getIconifiedClients() returnerar antal klienter som är ikonifierade, vilket i sin tur används av ikonsystemet (se kapitel 4.3). Execute(...) är en funktion som exekverar ett terminalkommando utifrån inkommande parameter. För att starta applikationer med KauWM så används en applikationsmeny (Figur 4.1 visar hur denna meny ser ut). Denna meny kommer fram då musklick på rotfönstret sker. Denna meny innehåller inte fler än tre applikationer, xterm, mozilla samt emacs. Vissa delar av menyn är statiskt implementerade men största delen av menyn är dynamisk för att vidareutveckling ska kunna ske. Det som sker vid menyskapandet är att ett fönster skapas med storlek utifrån antalet rader som menyn ska bestå av (antal applikationer) samt storleken på det längsta namnet i menyn. Då menyn är skapad så körs en loop som väntar på inkommande musklick. Om ett musklick sker utanför menyn så försvinner den. Om musklicket istället sker

i menyn kontrolleras vart i menyn det sker och den applikation som har den platsen exekveras. Förutom applikationsmenyn finns en annan meny som har hand om de klienter som blivit minimerade. Minimering och ikonifiering av klienter fungerar nästan på samma sätt. Skilladen är att vid ikonifiering skapas en ikon på rotfönstret som användaren klickar på för att återställa klienten medan vid minimering får användaren endast fram klienten på nytt genom minimeringsmenyn. Denna minimeringsmeny fungerar ungefär på samma sätt som applikationsmenyn, fast dynamiskt (Figur 4.2 visar hur denna meny ser ut). Minimeringsmenyn är dynamisk eftersom den hanterar olika antal rader i menyn. Eftersom antalet minimerade klienter varierar så måste även minimeringsmenyn variera i storlek. Detta görs genom att beräkna antalet minimerade klienter utifrån en variabel i klienterna som är satt till sant eller falskt. Namnen på de klienter som är minimerade visas i minimeringsmenyn. Denna meny visas genom att klicka med vänster musknapp på rotfönstret. Vid val av klient som finns i menyn så kommer den klient som valdes att hamna överst, det vill säga över alla andra klienter på skärmen. Klienten som då är tillbaka till normalt läge är markerat som icke minimerad och kommer där med inte längre vara med i minimeringsmenyn. Förutom detta finns det i fönsterhanterarsystemet lite debuggningsfunktioner som endast används i utvecklingssyfte.



Figur 4.1: Applikationsmenyn för KauWM, med tre applikationsval.

4.2 Klientsystemet

Klientsystemet tillhandahåller allt det som sker med en klient och dess fönster. Alla de händelser som skickas från huvudloopen i fönsterhanterarklassen till klienten finns implementerade här. En händelse kan hanteras på olika sätt av klienten. Ett musklick kan exempelvis betyda förflyttning, förstoring, förminskning eller nedstängning av en klient.

Figur 4.2: Minimeringsmenyn med flera applikationer minimerade.

För att skapa en ny klient anropas funktionen createClient(...). Denna funktion initierar följande delar:

- Klientfönstrets unika ID lagras.
- Klienten sätts till initierad.
- Ikonifiering sätts till falskt.
- Klient sätts till att vara synlig (ej minimerad).
- Delar som har med förflyttning av fönster sätts.
- Namn på klienten lagras.
- Ikonnamnet på klienten lagras.
- Storlek på fönster lagras.
- Klient fönstret identifieras mot X-Servern.
- Border för klientfönster skapas och identifieras mot X-Servern.

Funktionen som hanterar musklickhändelser kontrollerar först om klienten är ikonifierad. Om en *ButtonPress* händelse kommer till denna funktion när klienten är ikonifierad betyder det att musklick på ikonen har skett. Deikonifiering sker i detta läge. Om klienten inte är ikonifierad

kontrolleras vilken musknapp som blev nedtryckt. Om det var vänster musknapp sker ytterligare kontroller. Eftersom ramen innehåller nedstängning, ikonifiering, maximering, minimering och förstoring/förminskning måste positionen på musklicket undersökas.Denna jämförs med området som motsvarar knapparna. Om mittenknappen på musen blev nedtryckt *borderfieras* klienten, vilket betyder att bara ramen för klienten är synlig. Figur 4.3 visar ett fönster i borderfierat läge. Med höger musknapp väljs endast vilket fönster som ska vara aktivt.

Figur 4.3: Borderfierat fönster.

Vid skapande av en ny klient sätts applikationsfönstret som barn till ramfönstret (ramfönstret blir förälder till applikationsfönstret). Detta görs för att applikationsfönstret ska följa de händelser som ramfönstret svarar på, exempelvis flyttas runt på skrivbordet. Den funktion som skapar ramfönstret identifierar även ramfönstret hos X-Servern. Därefter anropas en funktion som ritar upp grafik på ramfönstret. Nästa steg i denna funktion är att byta förälder till applikationsfönstret. Från att rotfönstret har varit förälder så blir nu ramfönstret förälder till applikationsfönstret. Varför detta sker beskrivs tidigare i detta kapitel.

Vid DestroyNotify händelsen ska klienten tas bort. Detta sker genom ett anrop till funktionen close(). Denna funktion låter applikationsfönstret få rotfönstret som förälder istället för ramfönstret. Efter byte av förälder för applikationsfönstret tas ramfönstret bort. Vissa operationer för att frigöra använt minne sker för att undvika minnesläckage. Ett anrop till huvudklassen *WManager* sker så att pekaren till klienten tas bort ur den länkade listan. Detta är dock inte allt som sker vid nedstängning av en klient, vilket kommer att förklaras längre fram i detta kapitel.

Då namnet på en klient ändras under körning så skickas en *PropertyNotify* händelse ut. Denna händelse hanteras av klienten och kontrolleras. Ifall atomen *XA_WM_NAME* är den som skickas ut så uppdateras namnet på klienten genom att namnet hämtas på nytt. Till sist så uppdateras borderfönstret för att det nya namnet på fönstret ska bli synligt. Exempel på detta beskrivs i kapitel 3.1 *Övergripande om fönsterhanteraren*. Där tas ett exempel upp om ett terminalfön-

ster som byter namn vid sökvägsbyte. Funktionen som ritar upp alla detaljer på fönsterramen (borderfönstret) hämtar först bredden och höjden på fönstret. Sedan ritas klientnamnet upp på ramen. Därefter ritas knapparna för ikonifiering, minimering, maximering samt nedstängning upp på ramen. Här ritas även boxarna (avgränsningarna) upp runt knapparna. Om klienten inte är ikonifierad så ritas storleken på ramfönstret upp längst ned på ramfönstret.

Förflyttning av fönster sker med funktionen moveClient(). Först så byts muspekaren ut mot en annan för att indikera fönsterflytt. En upprepning sker där händelser avläses på samma sätt som i huvudloopen (den upprepning som finns i *WManager*). De händelser som söks är *ButtonRelease* och *MotionNotify*. Den sistnämda händelsen inträffar då användaren flyttar på musmarkören. Vid *ButtonRelease* kontrolleras vilken typ av förflyttningsuppritning som ska användas. De metoder för förflyttningsuppritning som finns är realtidsuppdatering eller en rammarkör som visar till vilken position ett fönster ska flyttas. Med realtidsuppdatering menas att användaren ser hela fönstret flyttas på skärmen utan ändring av dess utseende. Med rammarkör menas att fönstrets utseende ändras (det tas egentligen tillfälligt bort) och endast en ram bestående av en rektangel visas. Fönstret placeras på den position som är aktuell vid *ButtonRelease* händelsen. Om den händelse som inträffar är *MotionNotify* så kontrolleras vart musmarkören befinner sig. Som nämndes ovan så kontrolleras vilken typ av förflyttningsuppritning som ska ske (realtidsuppdatering eller rammarkör). Till sist ritas fönstret upp på den aktuella positionen. Vid realtidsuppdatering så ritas även all grafik på borderfönstret upp. När endast rammarkör ska ritas upp vid fönsterförflyttningen, ritas endast en rektangel upp. Rektangeln har samma storlek som den klient som ska förflyttas. Denna ram försvinner vid *ButtonRelease* händelsen och fönstret ritas upp som beskrivs ovan. Färgen sätts på rektangeln och den ritas upp efter storleken på fönstret samt positionen på rotfönstret. Det fönster som musmarkören befinner sig över får tangentbordsfokus. Med tangentbordsfokus menas att alla tangentbordsnedslag skickas till det fönster som har fokus. Detta sker vid *EnterNotify* händelsen. För att maximera en klient (ramfönster samt applikationsfönster) till fullskärm (då även över ikoner) finns en knapp på borderfönstret som ligger i mitten av de tre knapparna till höger längst upp på ramen (se figur 4.4). Vid maximering kontrolleras om klienten

redan är maximerad eller ej. Om klienten är maximerad så ändras dess fönster tillbaka till den storlek som de var i innan maximeringen. Om klienten inte är maximerad så lagras de gamla positionerna för klientens fönster samt höjden och bredden. Sedan förstoras klientens fönster till helskärm. En maximerad klient kan inte bli förstorad eller förminskad med annat än maximeringsknappen. Förflyttning kan heller inte ske då en klient är maximerad.

Figur 4.4: Den inringade delen visar maximeringsknappen på ett fönster.

Motsatsen till maximera är minimera. Detta sker med knappen längst till vänster av de tre knapparna uppe i högra hörnet på ramen (se figur 4.5). Funktionaliteten bakom minimering är att aktuell klients fönster tas bort från uppritningen hos X-Servern. Detta gör att X-Servern inte ritar upp de fönster som klienten tillhandahåller, även om de finns kvar. För att få fram klienten igen så anropas en funktion som återställer uppritandet av klientens fönster hos X-Servern.

Figur 4.5: Den inringade delen visar minimeringsknappen på ett fönster.

För att förminska eller förstora ett fönster så finns det två markerade fält längst ned i båda hörnen av ramfönstret (se figur 4.6). Funktionen för att förminska och förstora en klient är väldigt lik förflyttningsfunktionen. På samma sätt som vid förflyttning finns två lägen att rita upp fönstret på. Dessa lägen är som nämndes tidigare, realtidsuppdatering av fönstret eller en rammarkör. Förminskning och förstoring kan ske åt x-led och y-led samtidigt. Vid förflyttning räknas endast en x och y koordinat ut. Här måste nya storleken räknas ut samt förskjutning i förhållande till gamla positionen.

Ikonifiering sker med knappen längst upp till vänster på ramen (se figur 4.7). Det som sker i denna funktionen är att en instans av *Icon* klassen skapas och initieras. Funktionen anropar

Figur 4.6: De ljust markerade delarna visar var förstoring och förminskning sker.

init(...) i ikonklassen *Icon* med följande parametrar, applikationsfönstrets ID, ramfönstrets ID, ikonnamnet och klientnamnet. Sist sätts klienten till att vara ikonifierad (se figur 4.10).

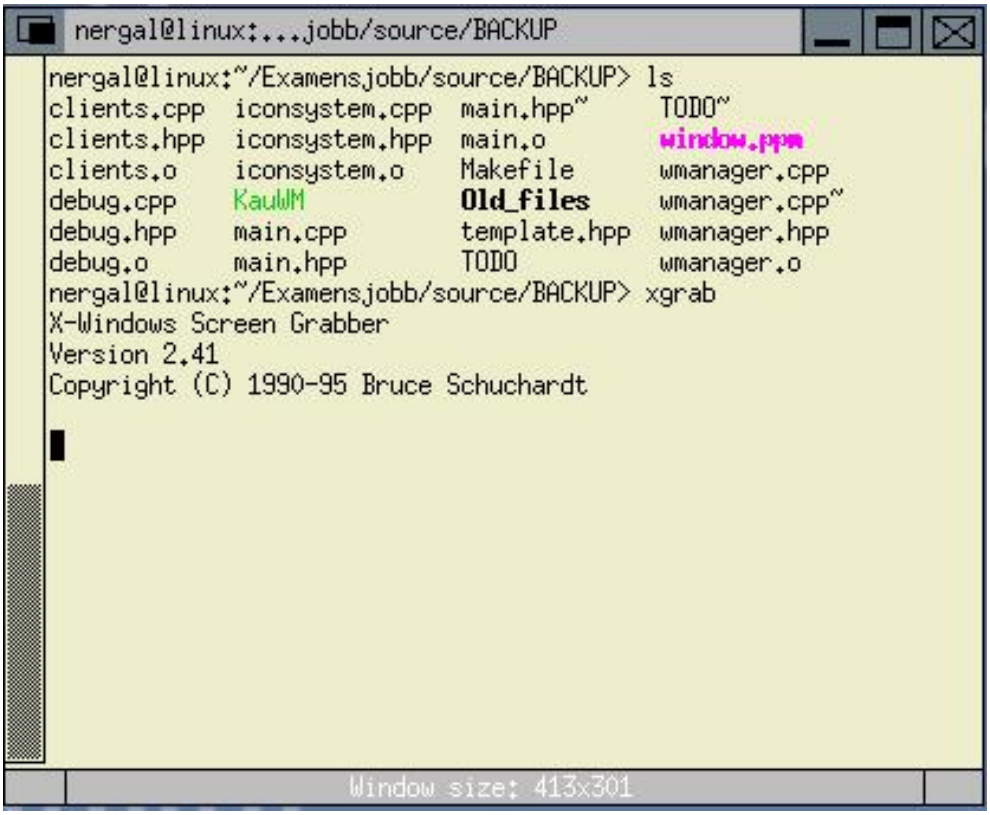
Figur 4.7: Den inringade delen visar ikonifieringsknappen på ett fönster.

En funktionalitet som inte är så vanlig bland fönsterhanterare är att *borderfiera* ett fönster (se figur 4.3). Det som menas med detta är att endast visa överdelen av ramen för ett fönster. Det som då är synligt är knapparna och namnet på klienten. Den delen av ramen som visar fönsterstorlek och har förstoring samt förminskningshantering visas ej.

Nedstängning av en klient sker med krysset längst upp i högra hörnet på borderfönstret (se figur 4.8). Vid nedstängning anropas i sin tur closeClient() som söker efter atomen WM_DELETE bland de atomer som klienten i fråga tillhandahåller. Om denna atom hittas betyder det att klientapplikationen själv kan sköta nedstängning. I detta fall behövs bara WM_DELETE skickas till applikationen, som då själv sköter nedstängning. Om inte applikationen stödjer denna atom sker en direkt nedstängning av applikationen, vilket dock är en sista utväg för att stänga ner klienten. Ofta är det äldre applikationer som inte stödjer WM_DELETE atomen.

Figur 4.8: Den inringade delen visar nedstängningsknappen på ett fönster.

Utöver de beskrivna funktionerna finns ett antal interna funktioner som hanterar felsökningsfunktioner och lokala variabler för klassen. Figur 4.9 visar en klient med den funktionalitet som har beskrivits i dett kapitel.



Figur 4.9: En applikation med KauWM's ramfönster.

4.3 Ikonsystemet

Ikonsystemet är ett enkelt konstruerat system som är långt ifrån färdigutvecklat. Ikonifiering sker på sådant sätt att då användaren valt att ikonifiera ett fönster så skapas en ny instans av klassen *Icon*. En funktion init(...) anropas i den nya instansen som i sin tur sparar undan information om klienten. Denna information består av namn, ikonamn, klientfönster ID samt borderfönster ID. Funktionen anropar sedan unmapper(...) som avmappar den klient som användaren valt att ikonifiera. Det som sker vid avmappning av en klient är att den inte längre ritas upp på skärmen av X-Servern. När klienten sedan blivit avmappad så skapas en ikon i form av ett nytt fönster, med funktionen createIcon(). Ikoner består endast av ett fönster. Till skillnad från applikationsfönstren, så har ikonerna inget ramfönster. Storleken på en ikon är som standard 64x64 pixlar. Ikonnamnet som en klient har skrivs ut på ikonen (se figur 4.10). För att göra ikonerna snyggare

kan grafiska bilder skapas, detta är dock en framtida vidareutveckling. Ikonsystemet hanterar två typer av händelser, vilka är *Expose* och *ButtonPress*.

Figur 4.10: Applikationer minimerade som ikoner.

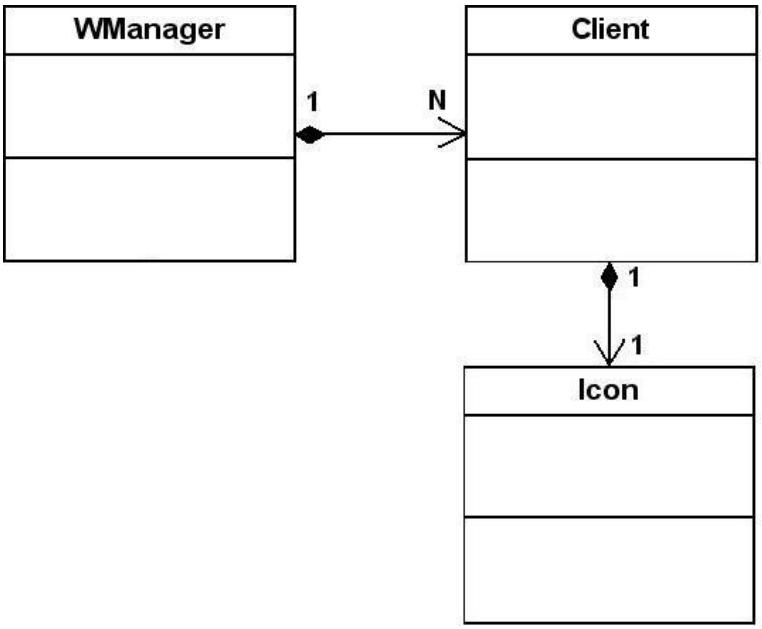
Det finns stöd för tre musknappar i ikonsystemet. Genom att klicka med vänster musknapp på ikonen så återskapas klientfönstret som det var innan det blev ikonifierat. Det som sker är att ikonfönstret tas bort och klientfönstret återmappas. Återmappning innebär att X-Servern kommer att börja rita upp klientfönstret på skärmen igen. Detta sköter funktionen remapper(). De två övriga musknapparna har ännu inte någon funktionalitet inom ikonsystemet. Förslag på framtida funktionalitet kan vara att stänga ner eller maximera en klient från ikonifierat läge.

4.4 Listhantering

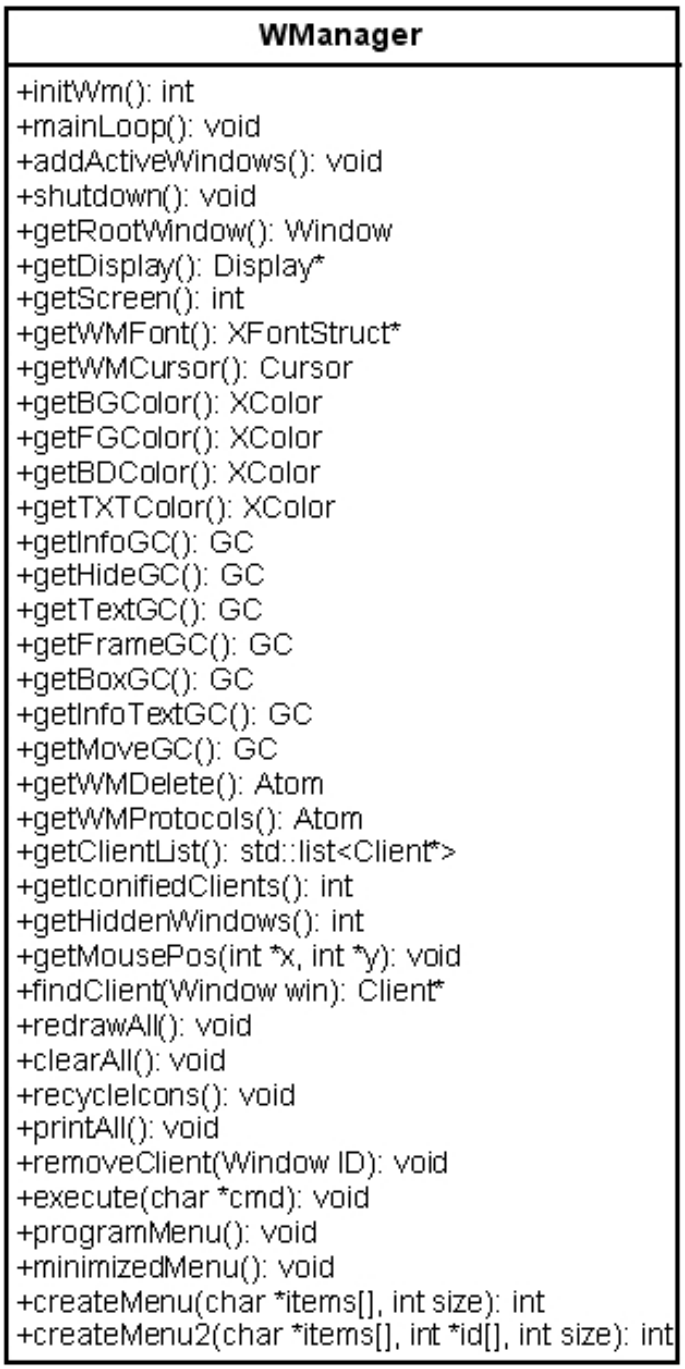
En lista används till att spara undan pekare till alla klienter som KauWM har hand om. Denna lista använder sig av STL (Standard Template Library) och är en länkad lista. Eftersom den länkade listan hela tiden används till att hålla ordning på klienterna så kan listan sägas vara en av de viktigare delarna i KauWM. Den funktionalitet som används för listan är att lägga till och ta bort pekare till klienter. Sökning i listan finns implementerat externt i fönsterhanteraren. Detta för att få korrekt sökning efter aktuella klienter (pekare till klienterna). En egen sökfunktion implementeras eftersom STL inte klarar av den typ av sökning som KauWM kräver.

4.5 Klassdiagram

I detta delkapitel visas klassdiagram över de klasser som KauWM tillhandahåller. Klassdiagrammen visar funktioner samt variabler som klasserna har. Figur 4.11 visar kopplingen mellan de olika klasserna. Figurerna 4.12 och 4.13 visar klassen WManager. Client klassen visas i figurer 4.14 och 4.15. Sista diagrammet är klassen Icon som visas i figur 4.16. I klassdiagrammen indikeras publika metoder och variabler med + och privata metoder samt variabler med -.



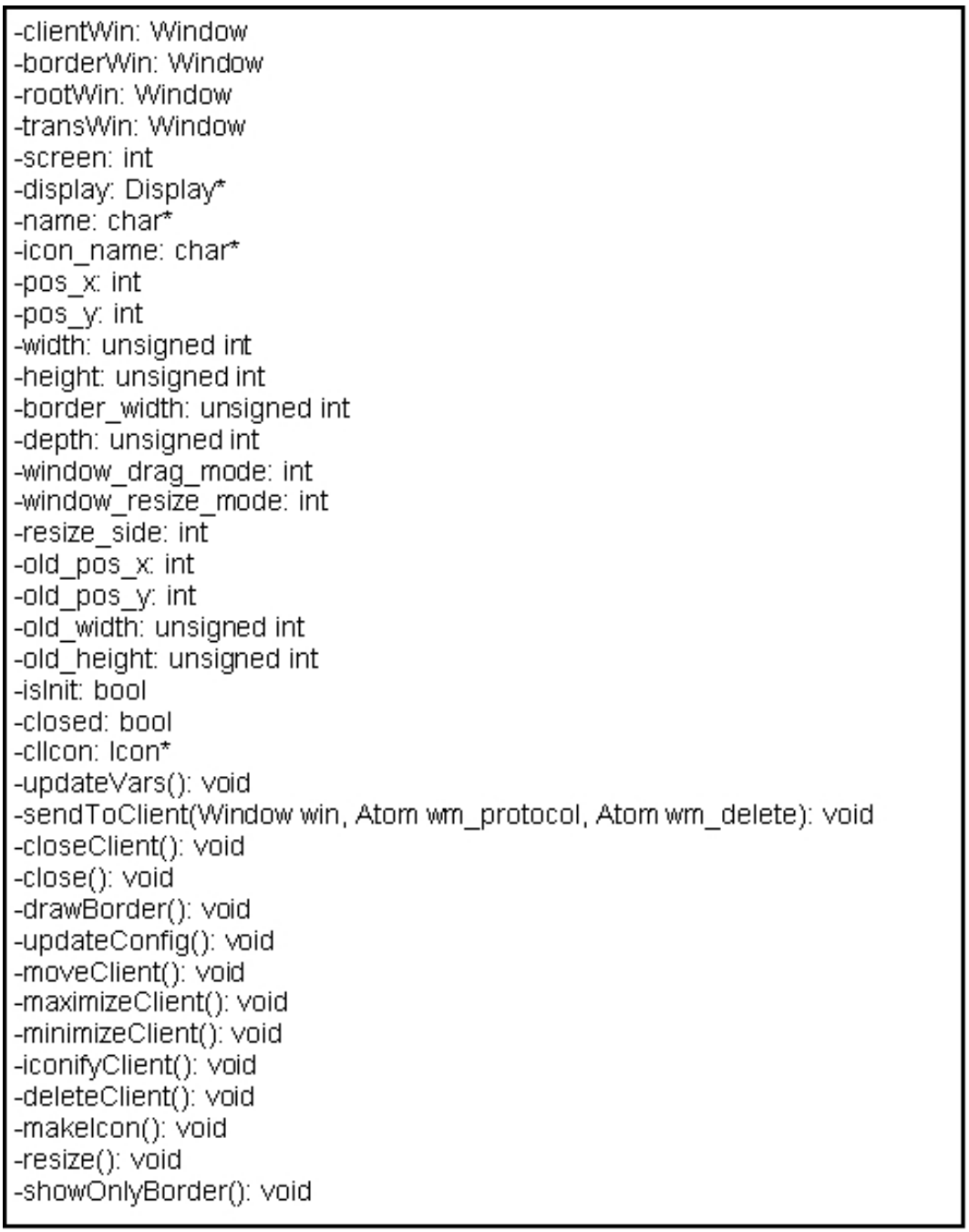
Figur 4.11: Bilden visar hur klasserna hör ihop.



Figur 4.12: Bilden visar första delen av klassen WManager.

Figur 4.13: Bilden visar andra delen av klassen WManager.

Figur 4.14: Bilden visar första delen av klassen Client.



Figur 4.15: Bilden visar andra delen av klassen Client.

Figur 4.16: Bilden visar klassen Icon.

5 Sammanfattning

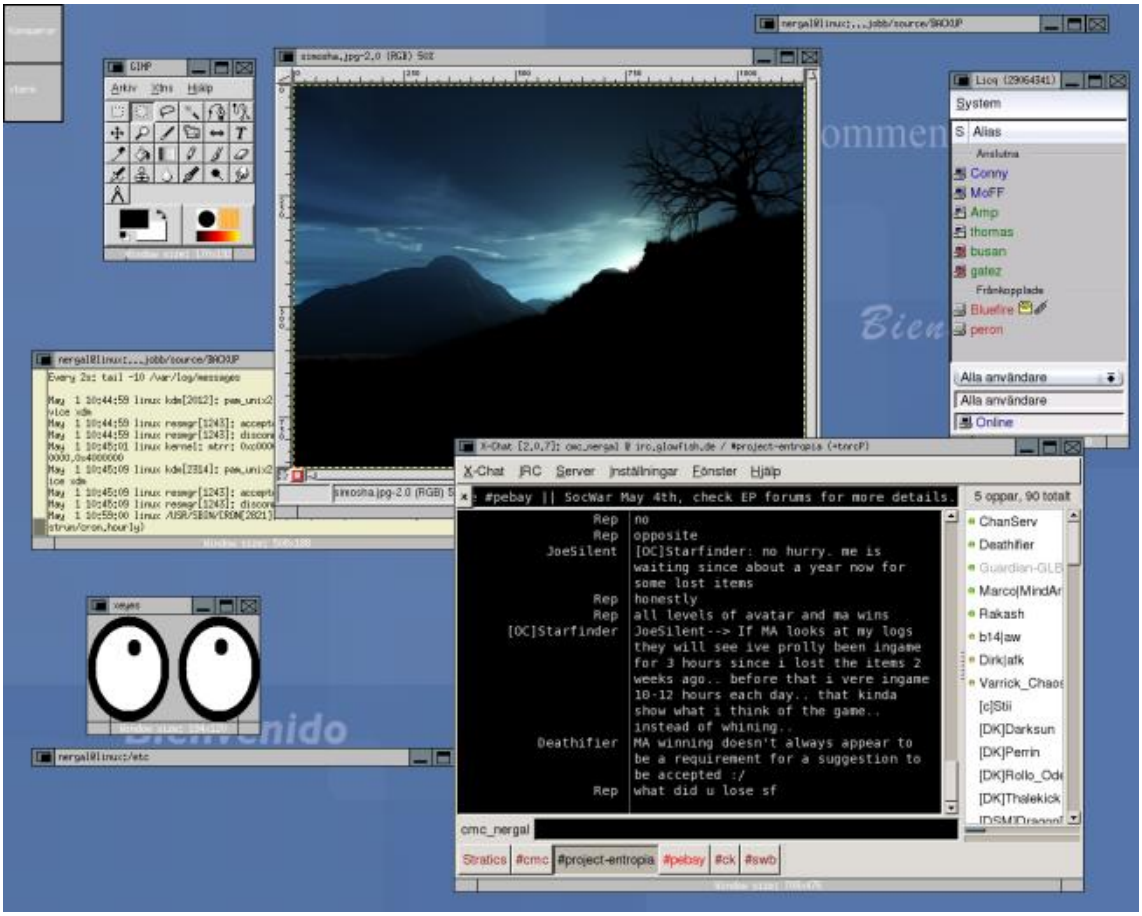
I detta kapitel beskrivs hur den praktiska delen genomfördes, det vill säga utvecklandet av en fönsterhanterare. Även den teoretiska delen behandlas, alltså hur denna rapport tillkommit. Det ges några förslag på vidareutveckling av KauWM och till sist avslutas kapitlet med några slutkommentarer.

5.1 Praktisk del

Planeringen av projektets praktiska del var relativt bra i jämförelse med den förbestämda tidsplanen. Den största delen gick åt till att skriva kod samt inläsning på områden som täcktes av projektet. Från början var tanken att använda QT[2] som grafiskt programmeringsgränssnitt. Detta var tänkt att användas till fönsterdekoration och ikoner. Efter inläsning på området och tester, insågs att det skulle bli onödigt stort att involvera QT för den tänkta grafiken. Även om det skulle se bättre ut att använda QT för fönsterdekorationer så är det inte det viktigaste hur det ser ut, utan hur det fungerar. Eftersom Xlib redan användes för initieringar av KauWM så valde vi att enbart arbeta med Xlib för grafiken till KauWM. Vissa problem uppkom under projektets gång (se kapitel 5.3), de flesta löstes dock. Ett sätt att få förslag på hur olika problem kan lösas är att titta på andra fönsterhanterare. En fönsterhanterare som var ett bra exempel att titta på var aewm[3]. Denna fönsterhanterare är skriven i programspråket C[4], vilket underlättade för oss då detta projekt utvecklades i C++. De flesta fönsterhanterare är uppbyggda på liknande sätt, dessutom är programspråken C och C++ ganska lika. Fönsterhanteraren aewm gav oss därför vissa tips på lösningar till problem som uppstod. KauWM har utvecklas under GPL[5] vilket medför att källkoden är öppen för alla. Detta för att källkoden ska kunna publiceras tillsammans med rapporten.

Målet med projektet var att få en fungerande fönsterhanterare för X-Windows under operativsystemet Linux. De mål som var uppsatta i projektets början blev uppfyllda. Dessutom lades lite ytterligare funktionalitet till, exempelvis enkla menyer för att visa minimerade klienter samt

att starta vissa applikationer. En bild på där KauWM används med många applikationer igång kan ses i figur 5.1. I nuläget ser KauWM ut som på denna bild.



Figur 5.1: Bild på KauWM med flera applikationer igång.

5.2 Teoretisk del

Rapporten var planerad att påbörjas tidigt. Så blev dock inte fallet på grund av vissa problem som uppstod med den praktiska delen (se kapitel 5.3). Från början planerades en viss struktur på rapporten som efter hand ändrades för att skapa bättre förståelse. Rapporten beskriver hur projektet gick och de problem som uppstod. Den beskriver även hur den skapade fönsterhanteraren fungerar och är uppbyggd. Rapporten publiceras tillsammans med källkoden för att skapa bättre förståelse.

5.3 Problem

Eftersom det från början var tänkt att alla klienter skulle placeras i en länkad lista så var en länkad lista ett måste att implementera. Vi skapade en länkad lista utifrån information på Internet och egna färdigheter. Vi baserade den på C++ templates så att vi kunde spara vilken datatyp som helst i den. Problemet var att vi inte kunde få den att spara undan pekare till klienterna så vi fick spara undan dem som referenser. Det här gick bra större delen av projektet tills vi insåg att det blev minnesläckor på väldigt många ställen. Vi insåg att vi skulle vara tvungna att spara pekare till klienterna i den länkade listan istället. Detta var dock inte så enkelt att lägga till. Efter att ha fått lite tips om den länkade listan i STL (standard template library) som finns inbyggt i C++ så användes denna lista. Med denna kunde pekare till klienterna sparas undan, vilket medförde att många minnesläckor försvann. Dessutom medförde införandet av den nya listan att det blev mycket mindre kod eftersom listan redan finns implementerad i STL.

Ett problem som fortfarande finns kvar är uppdateringen av ramen vid förflyttning, förminskning och förstoring. Ramen ska vid dessa händelser ritas upp över alla fönster som finns på skrivbordet, men i nuläget fungerar inte uppdateringen av grafiken som den ska vid dessa händelser. Eftersom detta problem fortfarande är olöst i KauWM så ritas ramen endast upp på rotfönstret. De problem som uppstod på den teoretiska delen var bland annat att vi påbörjade rapporten för sent. En anledning till detta var att mycket tid i början gick åt till inläsning inom området. Det

var även svårt att börja på rapporten innan projektet hade kommit igång. Att skapa en röd tråd i rapporten var problematiskt eftersom det är en beskrivande rapport av ett arbete som utförts, men efter flera revisioner är vi nöjda även med rapporten.

5.4 Vidareutveckling

Det finns mycket som kan vidareutvecklas på KauWM. Stödet för andra fönsterhantare (i form av skrivbordsmiljöer, till exempel KDE) bör läggas till. Den stödjer heller inte ICCCM (InterClient Communication Convention Manual). Detta gör att vissa applikationer inte fungerar perfekt med KauWM. Detta medför att vissa fönster kan mappas på fel sätt. Ikonsystemet är inte så långt utvecklat ännu. Till exempel så är ikonplaceringen för tillfället inte helt färdigutvecklad. För att få en bra placering av ikoner och omplacering vid borttagning av en ikon så bör en länkad lista används för detta. Användandet av en länkad lista skulle medföra att numrering av ikonerna kan ske, vilket skulle underlätta vid ikonplacering. Uppdateringsproblemet vid förflyttning, förminskning och förstoring som togs upp i kapitel 5.1 skulle kunna lösas genom vidareutveckling. Ett annat uppdateringsproblem är exempelvis då ett fönster förflyttas över ett annat applikationsfönster. Grafiken på applikationsfönstret uppdateras ej förrän användaren placerat fönstret. Det vore bra om uppdateringsproblemen klarades upp.

Exempel på vidareutveckling som inte är lika viktig eller kritisk är att skapa bättre grafik på ikoner och fönsterramar. Att kunna byta bakgrundsbild eller bakgrundsfärg på skrivbordsytan är ytterligare en funktionalitet som skulle kunna läggas till. Virtuella skrivbord är en bra egenskap, som KauWM i nuläget saknar. Detta skulle dock ta för lång tid att lägga till vilket vi inte hann med till deadline. Många kodoptimeringar skulle kunna ske genom att skriva om funktioner så att det går att använda samma funktion till olika ändamål. Exempel på detta är menyerna. Nu används två olika funktioner för att skapa de två olika menyerna för minimerade fönster och applikationsstart. Koden för dessa menyer är dock nästan lika. Det skulle kunna ändras så att endast en funktion skapar båda dessa menyer. Snabbkommandon med tangentkombinationer finns implementerade men inga är aktiva, men på grund av tidsbrist hanns detta ej med. Exempelvis

skulle det kunna läggas till att alt-tab skulle byta aktivt fönster.

5.5 Slutkommentarer

Att göra ett examensarbete var nytt för oss, vilket visade sig vara riktigt roligt och kunskapsprövande. Valet att skapa en enkel fönsterhanterare var ett bra projekt som examensarbete eftersom tiden räckte till för att skapa det viktigaste. En annan bra egenskap var projektets skalbarhet eftersom vissa delar kunde skapas och andra mindre viktiga delar kunde utelämnas. Projektet har givit oss stor kunskap om hur en fönsterhanterare fungerar och är uppbyggd, men samtidigt har vi insett att det finns väldigt mycket mer att lära sig inom området. Då vi började med projektet så trodde vi att det skulle bli svårt att sätta sig in i Xlib och hur det fungerar. Vi bläddrade genom många manualer på Internet och sökte i litteraturen i ämnet som visade sig vara väldigt stort. Men efter att vi börjat programmera lite och skapat några testprogram verkade det inte allt för svårt. Vi blev mycket nöjda med resultatet av projektet, en fönsterhanterare som vi själva använder. Projektet har gett oss ökad kunskap och erfarenhet. Exempel på vad vi lärt oss är hur en fönsterhanterare fungerar, hur X-Windows fungerar och hur programmering med Xlib går till.

Referenser

[1] O'Reilly Associates. Edited by Adrian Nye. *Volume Two: Xlib Reference Manual*. O'Reilly Associates, Inc, Sebastapol, 1990. ISBN 0-937175-12-9.

[2] Matthias Kalle Dalheimer. *Programming with QT*. O'Reilly Associates, 2002. ISBN 0596000642.

[3] Foster Decklin. *For more information look at www.red-bean.com/ decklin/aewm/*. 2003.

[4] Brian Kernigan. *The C Programming Language*. Prentice-Hall, 1988. ISBN 0131103628.

[5] GNU Public License. *For more information look at www.gnu.org/copyleft/gpl.html*.

[6] David Nash. *KDE Bible*. Wiley Publishing, 2000. ISBN 0764546929.

[7] O'Reilly Associates. By Adrian Nye. *Volume One: Xlib Programming Manual*. O'Reilly Associates, Inc., United States of America, 1995. ISBN 1-56592-002-3.

[8] Bjarne Stroustrup. *C++ Programming Language*. Addisson-Wesley, 2000. ISBN 0201700735.

[9] Annika Ullberg, Hegardt. *Microsoft Windows XP: Handboken*. Pagina Förlags AB, Sverige, 2002. ISBN 9163607387.

A Funktionsbeskrivningar för klasser

Endast de viktigaste och grundläggande funktionerna är beskrivna, för övrig information om funktioner se bifogad källkod. De funktioner som ej är beskrivna är bland annat funktioner som endast returnerar en privat variabels värde i klassen. Dessa funktioner ses inte som relevanta att beskriva.

A.1 Klassen Client

Funktionsnamn: createClient(Window)
Beskrivning: Ställer in variabler hos klienten.
Pre: True
Post: Klienten är initierad och klar att användas.

Funktionsnamn: clMouseButtonPress(XEvent)
Beskrivning: Hanterar musklickhändelser som genereras till klienten. Tre musknappar stöds, vänster-, mitt-, högerknapp.
Pre: True
Post:
(vänsterknapp) Beroende på position av klicket så är följande funktioner exekverade stäng, ikonifiera, minimera, förstora, förminska, flytta etc.
(mittenknapp) Ifall klick skedde på ramen så är fönstret borderfierat.
(högerknapp) Fönster markerat som aktivt.

Funktionsnamn: clDestroyNotify(XEvent)
Beskrivning: Tar emot ett DestroyNotify händelse och hanterar detta. Tar bort fönster för klienten genom att avmappa borderfönster samt ta bort klienten från länkade listan.
Pre: Klienten existerar och finns i länkade listan.

Post: Klienten borttagen från listan och fönster avmappat.

Funktionsnamn: clExpose(XEvent)

Beskrivning: Hanterar Expose händelse som ritar om borderfönstret.

Pre: True

Post: ramen uppdaterad.

Funktionsnamn: clMapRequest(XEvent)

Beskrivning: Hanterar MapRequest händelse som lägger till borderfönster samt klientfönster hos X-Server (mappar).

Pre: True

Post: Borderfönstret samt klientfönstret återmappat.

Funktionsnamn: clPropertyNoitfy(XEvent)

Beskrivning: Hanterar PropertyNotify händelse som här uppdaterar namnet på klienten.

Pre: Klienten existerar

Post: Namnet på klienten uppdaterat och ramen uppdaterad med nytt namn.

Funktionsnamn: clEnterNotify(XEvent)

Beskrivning: Hanterar EnterNotify händelse som sätter tangentbordsfokus på klienten.

Pre: Klienten är inte ikonifierad.

Post: tangentbordsfokus satt på klienten.

Funktionsnamn: clConfigureRequest(XEvent)

Beskrivning: Hanterar ConfigureRequest händelse för att ändra på någon atom hos klienten.

Pre: Klienten existerar.

Post: Begärd ändring utförd på klienten.

Funktionsnamn: showWindow()

Beskrivning: Återmappar borderfönster samt klientfönster och ritar om dekorationen på ramen.

Pre: Klienten existerar.

Post: borderfönster samt klientfönster återmappat och uppdatering av dekoration på ramen (borderfönstret) har skett.

Funktionsnamn: drawDecoration()

Beskrivning: Ritar upp dekorationen på borderfönstret, dvs knappar och text samt linjer.

Pre: Klienten existerar.

Post: dekorationen på ramen uppritad.

Funktionsnamn: updateVars()

Beskrivning: Uppdaterar privata variabler (vidd, bredd, x, y etc.)

Pre: True

Post: De privata variablerna är uppdaterade.

Funktionsnamn: sendToClient(Window, Atom, Atom)

Beskrivning: Skickar en händelsebegäran till en klient.

Pre: Klienten existerar.

Post: Händelsebegäran skickad till klient.

Funktionsnamn: closeClient()

Beskrivning: Skickar en nedstängningsbegäran till en klient om denna stödjer sådan begäran, annars dödas klienten (processen).

Pre: Klienten existerar.

Post: Klienten nedstängd.

Funktionsnamn: close()
Beskrivning: Avmappar fönster samt tar bort klient ur länkade listan och ändrar förälder på klient till rotfönstret. Anropas vid DestroyNotify.
Pre: True
Post: Klient borttagen.

Funktionsnamn: drawMoveFrame(int, int, unsigned int, unsigned int)
Beskrivning: Tar emot x,y,vidd,bredd parametrar som sedan en ram ritas upp utifrån. Denna ramen är endast en rektangel på rotfönstret.
Pre: True
Post: Ramen uppritad på rotfönstret.

Funktionsnamn: drawBorder()
Beskrivning: Skapar borderfönster till klientfönstret, mappar detta och sätter det till förälder för klientfönstret (istället för rotfönstret). (borde ha ett annat mer passande namn).
Pre: True
Post: Borderfönster skapat och mappat.

Funktionsnamn: updateConfig()
Beskrivning: Uppdaterar fönsterkonfigurationen (width, height etc.)
Pre: Klienten existerar.
Post: Fönstrets konfiguration uppdaterad.

Funktionsnamn: moveClient()
Beskrivning: Flyttar klienten på skrivbordet. Två lägen finns, realtidsuppdatering eller endast rita upp en ram.

Pre: Klienten existerar och klient är ej ikonifierad, maximerad eller minimerad.

Post: Fönstret flyttat till vald position.

Funktionsnamn: maximizeClient()

Beskrivning: Maximerar klienten till hela skärmstorleken.

Pre: Klienten existerar och är inte ikonifierad.

Post: Fönstret maximerat.

Funktionsnamn: minimizeClient()

Beskrivning: Minimerar en klient så den endast är synlig i minimeringsmenyn.

Pre: Klienten existerar och är inte ikonifierad.

Post: Klienten är minimerad.

Funktionsnamn: makeIcon()

Beskrivning: Skapar en ikon och initierar denna.

Pre: klienten är inte ikoniferad redan och klienten existerar.

Post: Ikon för klient skapad och initierad.

Funktionsnamn: resize()

Beskrivning: Ändrar storlek på fönster till mindre eller större.

Pre: Klienten existerar och är inte maximerad, minimerad eller ikonifierad.

Post: Storleken på fönstret är ändrat.

Funktionsnamn: showOnlyBorder()

Beskrivning: Borderfierar klienten till att endast vara synlig som border med knappar och text.

Pre: Klienten existerar.

Post: Klienten är borderfierad.

A.2 Klassen Icon

Funktionsnamn: init(char*, char*, Window, Window)

Beskrivning: Tar emot namn, ikonnamn, ID för border- och klientfönster. Sparar undan dessa och anropar vidare initiering av ikon.

Pre: True

Post: Ikonen är initierad.

Funktionsnamn: iconButtonClick(XEvent)

Beskrivning: Hanterar musklick händelser på ikonfönstret. Vid vänsterklick så deikonifieras ikonen.

Pre: Vänster musknapp nedtryckt.

Post: Ikonen borttagen och klienten återmappad.

Funktionsnamn: iconExpose(XEvent)

Beskrivning: Ritar om grafiken för ikonen (texten).

Pre: Ikonen existerar.

Post: Grafiken uppdaterad på ikonen.

Funktionsnamn: remapper()

Beskrivning: Återmappar klienten till ursprungligt läge och tar bort ikonfönstret samt frigör minne.

Pre: Ikonen existerar.

Post: ikonen borttagen och klienten återskapad.

Funktionsnamn: drawGfx()

Beskrivning: Ritar upp grafik på ikonfönstret.

Pre: Ikonen existerar.

Post: Grafik på ikonfönstret uppritad.

Funktionsnamn: createIcon()

Beskrivning: skapar ikonfönster samt mappar detta.

Pre: True

Post: Ikonfönster skapat.

Funktionsnamn: unmapper(Window)

Beskrivning: Avmappar borderfönster och klientfönster hos klienten.

Pre: Borderfönster samt klientfönster mappade.

Post: Border- och klientfönster avmappat.

Funktionsnamn: calc_y()

Beskrivning: Räknar ut antalet ikonifierade fönster för att få ut placering i y-led.

Pre: True

Post: antalet ikonifierade klienter returnerade.

A.3 Klassen WManager

Funktionsnamn: initWm()

Beskrivning: Initierar alla delar i fönsterhanteraren så som skärm, rotfönster, grafik contexts etc.

Pre: True

Post: Fönsterhanteraren är initierad.

Funktionsnamn: mainLoop()

Beskrivning: Loopar igenom händelsekön med de händelser som sker och anropar motsvarande funktion för varje händelse.

Pre: True

Post: True

Funktionsnamn: addActiveWindows()

Beskrivning: Lägger till de program som redan finns igång i länkade listan samt skapar klienter av dem.

Pre: True

Post: Aktiva fönster inlagda i länkade listan.

Funktionsnamn: getIconifiedClients()

Beskrivning: Returnerar antalet klienter som är ikonifierade, används av ikonsystemet.

Pre: True

Post: Antalet ikonifierade klienter returnerade.

Funktionsnamn: getMousePos(int*, int*)

Beskrivning: Hämtar aktuell muspekareposition på skrivbordet som referens.

Pre: True

Post: Aktuell musposition sparad i inkommande parametrar.

Funktionsnamn: redrawAll()

Beskrivning: Anropar drawDecoration() i alla klienter i länkade listan, vilket ritar om all grafik på klienterna.

Pre: Länkade listan ej tom.

Post: Grafiken hos alla klienter i länkade listan uppdaterad.

Funktionsnamn: clearAll()

Beskrivning: Tömmer alla fönster i länkade listan

Pre: Länkade listan ej tom.

Post: All grafik hos klienterna utsuddad.

Funktionsnamn: removeClient(Window)

Beskrivning: Tar bort en klient ur länkade listan utifrån klientens fönster ID.

Pre: Klienten finns i listan

Post: Klienten borttagen ur länkade listan.

Funktionsnamn: execute(char*)

Beskrivning: Exekverar ett konsolkommando i ett sh skal utifrån inkommande parameter.

Pre: Inkommande kommando existerar och får köras.

Post: Kommando exekverat.

Funktionsnamn: programMenu()

Beskrivning: Skapar en meny med program som går att exekvera med muspekarval.

Pre: Meny är ej uppritad.

Post: Meny är uppritad och val har skett. (musklick)

Funktionsnamn: createMenu(char**, int)

Beskrivning: Skapar ett menyfönster med val utifrån inkommande parameterlista. Denna meny skapad för att starta program.

Pre: True

Post: Meny är skapad och val har skett.

Funktionsnamn: minimizedMenu()

Beskrivning: Skapar en meny för att visa de minimerade klienterna.

Pre: True

Post: Meny är uppritad och val har skett (musklick)

Funktionsnamn: createMenu2(char**, int)

Beskrivning: Skapar en meny med val utifrån inkommande parameterlista, optimerad för att användas till minimerade klienter.

Pre: True

Post: Meny är skapad och val har skett.

Funktionsnamn: getHiddenWindows()

Beskrivning: Letar igenom länkade listan efter klienter som är minimerade (gömda)

Pre: True

Post: Antalet minimerade klienter returnerat.

Funktionsnamn: findClient(Window)

Beskrivning: Letar upp ifall en klient finns i länkade listan utifrån fönster ID.

Pre: True

Post: Ifall klient finns så returneras pekare till denna, annars returneras en icke initierad klient.

Funktionsnamn: mouseButtonPressHandler(XEvent)
Beskrivning: Letar upp i vilken klient händelsen skedde och kontrollerar ifall vissa krav uppnås och skickar vidare till motsvarande funktion hos klienten.
Pre: True
Post: Åtgärd till händelse genomförd.

Funktionsnamn: mapRequestHandler(XEvent)
Beskrivning: Letar upp i vilken klient händelsen skedde i och kontroller ifall vissa krav uppnås samt skickar vidare till motsvarande funktion hos klienten. Om klienten inte redan finns så skapas en ny och läggs in i länkade listan.
Pre: True
Post: Klient skapad och inlagt i länkade listan eller åtgärd för händelsen genomförd.

Funktionsnamn: keyPressHandler(XEvent)
Beskrivning: Tar hand om tangentbordsnedtryckningar och utför åtgärd för utvalda tangenter.
Pre: True
Post: Åtgärd för händelse utförd.

Funktionsnamn: destroyNotifyHandler(XEvent)
Beskrivning: Letar upp vilken klient som händelsen skedde i och kontrollerar vissa krav ifall dessa uppnås samt skickar vidare händelsen till motsvarande funktion hos klienten.
Pre: True
Post: Åtgärd för händelse genomförd.

Funktionsnamn: exposeHandler(XEvent)
Beskrivning: Letar upp vilken klient som händelsen skedde i och kontrollerar vissa krav ifall dessa uppnås samt skickar vidare händelsen till motsvarade funktion i klienten.

Pre: True
Post: Åtgärd till händelsen utförd.

Funktionsnamn: propertyNotifyHandler(XEvent)
Beskrivning: Letar upp vilken klient som händelsen skedde i och kontrollerar vissa krav ifall dessa uppnås samt skickar vidare händelsen till motsvarade händelsehanterare i klienten.
Pre: True
Post: Åtgärd för händelse utförd.

Funktionsnamn: enterNotifyHandler(XEvent)
Beskrivning: Letar upp i vilken klient som händelsen skedde i och kontrollerar vissa krav ifall dessa uppnås, därefter skickas händelsen vidare till klientens händelsehanterare.
Pre: True
Post: Åtgärd för händelse utförd.

Funktionsnamn: configureRequestHandler(XEvent)
Beskrivning: Om en ny klient skickar denna händelse så får den de krav den vill på startposition och storlek etc. Annars ifall klient redan finns så kontrolleras ifall den inte är en ikon, om inte så skickas händelsen vidare till klientens händelsehanterare.
Pre:
Post:

B Ordlista

Listan nedan presenteras för att förtydliga vissa ord som förekommer i rapporten.

- Applikation - En applikation är ett mjukvaruprogram på en dator.
- Terminalfönster - Ett fönster i Linux/Unix som tar emot kommandon från användare och exekverar dessa.
- terminalkommando - Ett kommando för exekvera program eller utföra vissa händelser.
- make - Ett program för att kompilera ett program med vissa direktiv förklarade i tillhörande makefile.
- Makefile - En fil som bestämmer vad som ska göras av programmet make.
- Xterm - Ett grafiskt terminalfönster i Linux.
- programmeringsgränssnitt - Ett verktyg för att skriva kod på ett mer effektivt och abstrakt sätt.
- Skrivbord - Även kallat rotfönster i X-Windows. Underliggande fönster som hanterar exempelvis bakgrundsbild.

C Källkoden för fönsterhanteraren

Källkoden följer efter detta blad.