

Multimaus / Remotemaus

Mehrere PCs, mehrere Mäuse, mehrere Cursor

Bachelorthesis

Studiengang: Bsc Informatik - Mobile Computing
Autor/in: Bachmann Rico
Betreuer/in: Danuser Andreas
Auftraggeber/in: -
Experten: Jäggi Thomas
Datum: 05.01.19

Management Summary

Arbeitsteam: Bachmann Rico
Betreuer: Danuser Andreas
Experte: Jäggi Thomas

Externe Ressourcen:

- keine

Material:

- 2x MacBook, zur Verfügung gestellt von der BFH und von Christoph Kiser.
- Laptop HP, Linux Mint, zur Verfügung gestellt von Rico Bachmann
- Laptop HP, Windows 10, zur Verfügung gestellt von Rico Bachmann

Tools:

- Clion IDE, mir g++-Toolchain (mingw oder cygwin für Windows)
- github, Sourcecode-Repository und Taskmanagement-Tool (<https://github.com/i3luefirech/mmBTbachr2>)

Abstrakt:

- Die Multi-/Remotemaus ist ein Tool, welches die Barrieren einer normalen Maus und eines normalen Betriebssystem-Cursors durchbricht. Die Multimaus ermöglicht es, mehrere Cursor auf einem Computer zu nutzen. Die Remotemaus erlaubt es, den Cursor mit einer Maus über mehrere Rechner zu benutzen. Dies kann Personen unterstützen, welche mehrere Computer am selben Arbeitsplatz in Betrieb haben oder es kann die Teamarbeit über mehrere Rechner erleichtern.

Ziel ist es, unter möglichst vielen Betriebssystemen, die Multi-/Remotemaus und ihre Funktionalität zu ermöglichen. Dafür muss das nötige Wissen über das Maus- und Hardwarehandling erarbeitet werden, es muss herausgefunden werden wie der Betriebssystem-Cursor gesteuert werden kann und es werden die nötigen Informationen, zum Zeichnen eigener Mauscursors, erschaffen. Dazu wird noch geprüft, welche aktuellen Produkte schon auf dem Markt sind und was die Unterschiede zur Multi-/Remotemaus ausmachen. Das Projekt umfasst das Erschaffen und Erarbeiten der nötigen Informationen, eine saubere Implementation der Funktionalität, auf einem oder mehreren Betriebssystemen, sowie die ausführliche Dokumentation des erhaltenen Wissens und dessen Umsetzung. Im ersten Timeslice der Arbeit wird das neue Wissen geschaffen, durch lesen von Literatur und fremdem Sourcecode. Im mittleren Timeslice wird daraus ein sauberer Lösungsansatz erarbeitet und umgesetzt. Im letzten Timeslice, wird die Umsetzung verfeinert, sowie die nötige Dokumentation erstellt.

Zur Zeit sind mehrere, der Multi-/Remotemaus ähnliche, Produkte im Umlauf. Dabei wurden drei Konkurrenzprodukte genauer angeschaut, Synergy, Logitech Flow und Microsoft Multipoint. Im allgemeinen lässt sich feststellen, dass diese Produkte zwar eine Remotemaus-Funktionalität anbieten, jedoch keine Multimaus-Funktionalität.

Als Lösungsansatz wird das Prinzip „man in the middle“ gewählt. Dabei wird unsere Software die Mausdaten vor dem Betriebssystem-Cursor abgreifen. Die abgegriffenen Daten werden nun entweder einem lokalen gezeichneten Cursor oder einem remote Cursor, via Netzwerk, zur Verfügung gestellt. Der Zugriff auf den Betriebssystem-Cursor soll mit einem „first come, first serve“ Ansatz gehandelt werden.

Die praktische Umsetzung dieses Projekts wurde mit Augenmerk auf *nix-Betriebssysteme erledigt. Dabei wurde auf folgende Technologien gesetzt, OpenGL zum Zeichnen der eigenen Cursor, Xlib zur Kontrolle des Betriebssystem-Cursor und Betriebssystem-Fenster, nlohmann_json zum Handling von JSON-Konfigurationsdateien, sowie Sockets für die Netzwerkkommunikation und klassische iostreams zum Lesen der Mausevents von einem Devicefile.

Die Umsetzung des Projektes ist gut gelungen und das Resultat gibt uns eine Software, welche die Grundfunktionalität der Multi-/Remotemaus unter *nix-Systemen zur Verfügung stellt. Die meisten der Herausforderungen die sich stellten, konnten, dank der gut erarbeiteten Informationsgrundlagen, mit einer einfachen, aber stabilen Umsetzung gemeistert werden.

Das Endprodukt bietet eine gute Grundlage, um auch in Zukunft weitere Hardware, auf einfache Weise, remote zu nutzen. Schon jetzt kann die Multi-/Remotemaus den Arbeitsprozess bei Team-Arbeiten oder das Arbeiten mit mehreren Rechnern an einem Arbeitsplatz, stark vereinfachen. Dadurch bildet sich auch schon ein Markt wo das Produkt platziert werden kann.

Inhaltsverzeichnis

1	Einleitung	5
2	Ausgangslage	7
2.1	Aufgabenstellung	7
2.2	Herausforderungen	7
2.2.1	Maus-Handling	7
2.2.2	Window-Server	8
2.2.2.1	Betriebssystem-Cursor	9
2.2.3	Zeichnen von Mauscursors	9
2.3	Motivation	10
2.3.1	Mehrere Computer an einem Arbeitsplatz	10
2.3.2	Teamarbeit	10
2.3.3	Bezug zur P2-Arbeit	10
2.3.4	Remote-devices	10
3	Arbeitstechnik	11
3.1	Scope	11
3.2	Out of Scope	11
3.3	Vorgehen	11
3.4	Zeitplanung	12
3.4.1	Zeitplan	12
3.4.2	Fazit Zeitplanung	12
4	Stand der Technik	13
4.1	Synergy	13
4.2	Logitech Flow	13
4.3	Microsoft Multipoint	14
4.3.1	Multipoint Maus	14
4.3.2	Multipoint Server	15
5	Lösungsansatz	17
5.1	Maus-Handling	18
5.2	Netzwerk / Kommunikation	19
5.2.1	Sockets	19
5.2.1.1	UDP Client	19
5.2.1.2	UDP Server	19
5.2.2	Datenformat	19
5.2.2.1	Konfigurationsinformationen	19
5.2.2.2	Übertragungsdaten	20
5.3	Cursor Zeichnen	20
5.3.1	OpenGL	20
5.3.1.1	Cursor	21
5.3.2	Window-Server	21
6	Architektur	22
6.1	Scope	22
6.2	Klassen Diagramme	22
6.3	Use-Cases	23
6.4	HW	24
7	Implementation	25
7.1	Maus-Handling	25
7.2	Zeichnen eines Cursors	27
7.3	Datenverarbeitung	30
7.4	Kommunikation	31
7.4.1	UDP Client	31
7.4.2	UDP Server	32

8	Resultat	34
8.1	Möglicher Markt	34
9	Schlussfolgerungen/Fazit	35
10	Abbildungsverzeichnis	36
11	Glossar	37
12	Literaturverzeichnis	38
13	Anhang	39
13.1	Sourcecode & Dokumentation	39
13.2	Journal	39
13.3	Planungsdiagramme	42
13.3.1	Zeitplanung	42
13.3.2	Softwareentwicklung	43
13.3.2.1	Scope	43
13.3.2.2	Klassendiagramme	43
13.3.2.3	Use-Cases	44
13.3.2.4	Sequenz-Diagramme	45
13.4	Anleitungen	45
13.4.1	Compilieren vom Sourcecode	45
13.4.1.1	Konfigurieren der Multi-/Remotemaus	45
14	Selbständigkeitserklärung	47

1 Einleitung

Die Multi-/Remotemaus ist ein Tool, welches die Barrieren einer normalen Maus und eines normalen Betriebssystem-Cursors durchbricht. Die Multimaus ermöglicht es, mehrere Cursor auf einem Computer zu nutzen. Die Remotemaus erlaubt es, den Cursor mit einer Maus über mehrere Rechner zu benutzen. Zusammen entsteht die Multi-/Remotemaus.

Das unterschiedliche Verhalten der verschiedenen Betriebssysteme stellt die grösste Herausforderung dieser Arbeit dar. Es gibt mehrere Punkte welche die Umsetzung der Multi-/Remotemaus beeinflussen. Wie zum Beispiel das Maus-Handling. Trotz einem grundsätzlich ähnlichen Verhalten unterscheiden sich die Betriebssysteme stark voreinander.

Wie werden die Mausdaten vom Treiber dem Window-Server weiter gegeben und wie stellt das Betriebssystem die Mausdaten einer Userspace-Applikation zur Verfügung? Eine weitere Hürde, die es zu meistern gilt, ist der Betriebssystem-Cursor. Wie kann der Betriebssystem-Cursor gesteuert werden und wie steuert eine normale Maus den Betriebssystem-Cursor? Wie halte ich mehrere Mauscursor? Antworten zu diesen Fragen finden sie im Kapitel 2 Ausgangslage.

Warum sollte überhaupt eine Multi-/Remotemaus entwickelt werden? Genau diese Frage habe ich mir öfter gestellt. Die Multi-/Remotemaus kann Personen unterstützen, welche mehrere Rechner an einem Arbeitsplatz in Betrieb haben. Es macht das Handling einfacher, wenn man alle Rechner mit der selben Maus bedienen kann. Ein weiteres Szenario ist die Teamarbeit mit mehreren Rechnern. Um einem Kollegen etwas zu zeigen, kann dank der Multi-/Remotemaus, mit der eigenen Maus einen Cursor auf seinem Rechner übernommen werden. Man kann ihm nun die nötigen Schritte mit der Maus zeigen oder sogar ausführen. Dabei bleibt sein Cursor, der lokalen Maus, an der bisherigen Stelle.

In dieser Arbeit werden im Zusammenhang mit der Aufgabenstellung folgende Themen bearbeitet: Es wird vertieftes Wissen zum Maus- und Hardwarehandling von Betriebssystemen erarbeitet. Dafür wird sich in Fachliteratur eingelesen. Weiter wird Wissen über Window-Server sowie Betriebssystem-Cursor erarbeitet. Dazu wird auch noch einiges an Grundlagen über OpenGL gelernt. Mit den erschaffenen Informationen wird ein Lösungsansatz für die Multimaus generiert. Diese Lösungsansatz soll, für ein oder mehrere Betriebssysteme, umgesetzt werden. Für das ganze Projekt wird eine Zeitplanung gemacht. Mehr zu diesen Themen im Kapitel 3 Arbeitstechnik.

Während meiner Recherchen sind mir einige der Multi-/Remotemaus ähnliche Produkte über den Weg gelaufen. Dabei sind mir vor allem folgende Produkte ins Auge gestochen: Synergy, Logitech Flow und Microsoft Multipoint. Synergy ist eine Opensource basierende Lösung einer Remotemaus, ohne Multimausfunktion. Logitech Flow ist eine auf Logitech Mäuse bezogene Lösung. Wenn man die richtige Logitech Maus besitzt, so wird einem mit Flow eine Softwarelösung für eine Remotemaus angeboten. Microsoft Multipoint, bietet ein SDK zum erstellen einer Multimaus Applikation zur Verfügung und es bietet einen Windows-Server Service für mehrere Windows-Sessions auf einem Rechner zur Verfügung. Diese Themen sind genauer behandelt im Kapitel 4 Stand der Technik.

Mit einem Prinzip, ähnlich des „man in the middle“-Verfahrens aus der IT-Security, versucht unsere Lösung sich zwischen das IO-Handling des Betriebssystems und den Window-Server zu setzen. An dieser Schnittstelle sollen die nötigen Mausevents abgegriffen werden. Diese Daten sollen nun verarbeitet werden. Entweder werden die Daten an einen der lokalen Cursor weiter gegeben, oder sie können über den Client des Programms an einen Remote-Rechner gesendet werden. Unsere Lösung muss auch einen Server implementieren, welche versendete Mausdaten entgegen nehmen kann. Genauere Informationen zum Lösungsansatz finden sich unter Kapitel 5 Lösungsansatz.

Zur Umsetzung des Lösungsansatzes und zur Modellierung wurden UML und UML ähnliche Diagramme verwendet. Unsere Multi-/Remotemaus wird aus mehreren Klassen bestehen. Die Hauptklasse wird der Multi-/Remotemaus-Controller darstellen. Dieser benötigt einen Client und einen Server zum senden und empfangen der verschiedenen Events. Um das Tool konfigurierbar zu machen, wird auch eine Klasse zum Parsen der Konfiguration benötigt. Um die verschiedenen Lokalen Mäuse zu handeln, wird eine Mäusecontrollerklasse so wie eine Mausklassse benötigt. Das gleiche gilt für die lokalen Cursors. Weitere Informationen zur Software-Architektur können im Kapitel 6 Architektur nachgelesen werden.

Bei der Implementation wurde die Priorität auf eine Software für *nix-Systeme (Unix und unixähnliche Systeme) gesetzt. Dabei mussten einige Punkte speziell beachtet werden. Um das Maushandling umzusetzen wird direkt auf das Mausfile zugegriffen. Damit dieses Mausfile nicht auch vom Window-Server gelesen wird, muss der Window-Server umkonfiguriert werden. In einem eigenen Thread wird jetzt das Devicefile überwacht. Wird ein Event vom Mausfile erhalten, wird das in unser Datenformat gepackt und der Multi-/Remotemaus übergeben. Um den Betriebssystem-Cursor zu steuern, wurde das Test-Modul des Window-Servers genutzt. Mit dieser Bibliothek kann im Code eine Maus simuliert werden. Dies ziehen wir uns zu Nutzen, wenn eine Klickaktion an den richtigen Mauscursor weitergegeben werden muss. Um die Kommunikation für die Remotemaus sicher zu stellen, wurde der Client und auch der Server mit UDP Sockets implementiert. Um die Cursor zu zeichnen, habe ich mit Hilfe der Xlib Bibliothek und dem Window-Server transparente Fenster erstellt. In diese transparenten Fenster wird mit OpenGL ein einfacher Cursor als Dreieck dargestellt. Genauere Informationen zur Implementation finden sich im Kapitel 7 Implementation. Wer Interesse am Sourcecode hat, kann das github Repository anschauen, oder die Codedokumentation im Anhang studieren, zu finden unter 14.1 Sourcecode & Doku.

Aus dem praktischen Teil der Arbeit entstand eine stabile Software, welche die Grundfunktionalität der Multi-/Remotemaus umsetzt. Die Software ist unter Linux und Mac OS lauffähig. Unter Mac OS nur begrenzt. Das fertige Resultat wurde auf einem Laptop mit Linux Mint, sowie mit einem MacBook getestet. Das Tool bietet noch Verbesserungspotenzial für die Zukunft., es können noch Funktionen wie Datenverschlüsselung oder ein Teilen der Zwischenablage implementiert werden. Auch der Port für Windows ist noch offen. Für das Endprodukt bildet sich ein möglicher Markt im Bildungssektor. Durch die Unterstützung bei Teamarbeit kann es ein wichtiges Tool bei der digitalisierten Ausbildung werden. Auch in der Wirtschaft findet sich ein Markt. Bei Betrieben in denen es vorkommt, dass eine Person mehrere Rechner an einem Arbeitsplatz betreibt, kann die Multi-/Remotemaus den Vorteil bringen, alle Computer mit der selben Maus zu steuern. Weiter Angaben zum Endresultat und einem möglichen Markt finden sich im Kapitel 8 Resultat

Über die ganze Arbeit konnte ich sehr viel neues und vertieftes Wissen über die verschiedenen Betriebssysteme lernen. Im Grossen und Ganzen kann sich der erarbeitete Lösungsansatz sehen lassen. Die gewählte Problemlösung so wie das Handeln der Mausevents klappt wie geplant. Das erfassen der Mausdaten würde ich, vor allem unter Mac OS, heute anders angehen. Ein detailliertes Fazit kann in Kapitel 9 Fazit/Schlussfolgerung gelesen werden.

Im Kapitel 14 Anhang befinden sich Dokument mit weiteren detaillierten Informationen zu dieser Bachelorthesis. Darunter befindet sich der Link zum Sourcecode, sowie die Sourcecode Dokumentation. Diese könnte vor allem für Entwickler sehr interessant sein und gibt einen tiefen Einblick in die Umsetzung des Projektes. Im Arbeitsjournal findet man weitere Informationen zum Verlauf des Projektes. Wer gerne den Sourcecode compilieren möchte, um die Multi-/Remotemaus zu testen, findet die Anleitungen dafür auch im Anhang

2 Ausgangslage

2.1 Aufgabenstellung

Nach dem ich in meiner P2-Arbeit aufzeigen konnte, dass es Grundsätzlich möglich ist, die Multi-/Remotemaus umzusetzen, ergaben sich daraus folgende Aufgaben:

- Basierend auf den Resultaten der Machbarkeitsstudie soll klar strukturiert zusammengestellt und aufgezeigt werden, welches die technischen Hürden sind, um über ein Daten-Netzwerk eine „zweite“ Maus auf einem Computer darzustellen und zu bewegen. Insbesondere sollen diese Hürden für Linux, Windows und Mac OS gezeigt werden.
- Ergänzend zu dieser Machbarkeit soll (soweit möglich) untersucht werden, welche Produkte am Markt bereits eine solche oder ähnliche Funktion aufweisen und wie die Funktion umgesetzt worden ist.
- Basierend auf diesen Vorbereitungen soll eine grundsätzliche Architektur vorgeschlagen werden für eine Remotemaus Treibersoftware.
- Im nächsten Schritt soll ein solcher Treiber für mindestens ein, besser zwei oder drei Betriebssystem(e) implementiert werden. Diese Implementation(en) soll(en) grundsätzlich in hoher Qualität erfolgen, damit das resultierende Produkt auch genutzt werden kann.
- Für den oder die implementierten Treiber soll in geeigneter Form eine Dokumentation für die Installation und Nutzung des Treibers gemacht werden.
- Ebenso soll skizziert werden, wie eine solche Treibersoftware im Markt verbreitet werden kann.

2.2 Herausforderungen

Das unterschiedliche Verhalten der verschiedenen Betriebssysteme stellt die grösste Herausforderung dieser Arbeit dar. Es gibt mehrere Punkte, welche die Umsetzung der Multi-/Remotemaus beeinflussen. Einer dieser Punkte ist das Maus-Handling. Trotz einem grundsätzlich ähnlichen Verhalten unterscheiden sich die Betriebssysteme stark. Wie werden die Mausdaten vom Treiber dem Window-Server weitergegeben und wie stellt das Betriebssystem die Mausdaten einer Userspace-Applikation zur Verfügung? Eine weitere Hürde die es zu meistern gilt, ist der Betriebssystem-Cursor. Wie kann der Betriebssystem-Cursor gesteuert werden und wie steuert eine normale Maus den Betriebssystem-Cursor? Wie erhalte ich mehrere Mauscursor?

2.2.1 Maus-Handling

Grundsätzlich funktioniert das Maus-Handling bei allen Betriebssystemen nach dem selben Prinzip. Die Hardware sendet ein Interrupt an den Prozessor ,um diesem mitzuteilen, dass er Daten für den Prozessor hat. Ein Interrupt ist ein Signal für den Prozessor, seine momentane Arbeit zu unterbrechen und eine definierte Aktion durchzuführen. Der Prozessor lädt in unserem Fall die Daten von der Maus und sendet diese an den Maus-Treiber. Der Maus-Treiber verarbeitet nun diese Hardwaredaten und sendet dem Betriebssystem ein Event, welches die Mausdaten enthält. Dieser Mausevent wird nun an den Window-Server weiter gegeben. Der Window-Server verarbeitet den Mausevent und zeichnet den Betriebssystem-Cursor. Wenn es sich um einen nicht auf die Bewegung bezogenen Event handelt, muss der Window-Server schauen, dass die nötigen Userspace-Applikationen den Event propagiert erhalten.

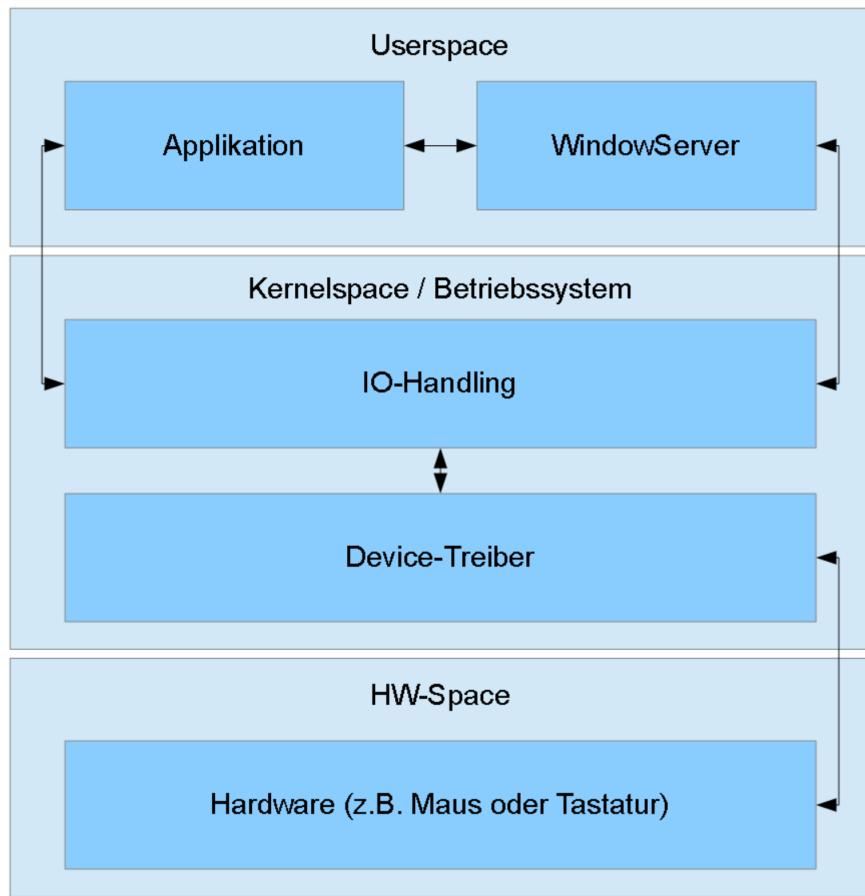


Illustration 1: Hardwarehandling eines Betriebssystems

Das genaue Handling und die Implementation der erwähnten Mousevents vom Maustreiber, sowie das Propagieren von Mousevents unterscheiden sich zwischen den Betriebssystemen. Während *nix-Systeme sich in den meisten Fällen sehr ähnlich verhalten, so stellt Windows hier den Exoten dar.

Unter *nix-Systemen wird der rohe Mouseevent vom Maustreiber über ein sogenanntes Devicefile propagiert. Nach dem Leitsatz „everything is a file“, wird also auch ein Mausdevice als File dargestellt. Unter *nix-Systemen wird alles als File behandelt, was heißen soll, dass ein File, ein Ordner oder auch ein Device vom Handling her immer ein File darstellen. Alle Diese Files lassen sich einfach mit open(), read(), write() und close() kontrollieren. Die Devicefiles für Benutzereingaben befinden sich unter /dev/input/eventN. Von diesen Devicefiles können nun die Mousevents ausgelesen werden.

Unter Windows wird der rohe Mouseevent als sogenannte Windows-Message versendet. Da aber eine Userspace-Applikation im Normalfall nur die für sie bestimmten Windows-Messages erhält, kriegt man eigentlich keine rohen Mousevents. Es gibt aber Workarounds, welche es einem erlauben, Windows-Messages zu erhalten, welche nicht zu seiner Applikation gehören. Via WindowHook kann so auf die rohen Maus-Events zugegriffen werden. Das dieser Ansatz klappt, konnte schon in der P2-Arbeit gezeigt werden.

2.2.2 Window-Server

Der Window-Server ist für die graphische Darstellung von Fenstern und den Betriebssystem-Cursor verantwortlich, sowie zum Propagieren von Betriebssystem- oder Hardwareevents, welche sich auf die graphische Benutzereingabe beziehen.

Bei *nix Systemen kommt als Window-Server im Normalfall X11 zum Zug. (Die meisten Linux-Distributionen verwenden X11 selbst, unter Mac OS wird ein Derivat von X11 verwendet, der sogenannte xQuartz) Dadurch verhalten sich alle *nix-Systeme gleich. Der Window-Server liest die vom Treiber propagierten rohen Mousevents vom Devicefile. Die Mousevents werden von ihm verarbeitet. Events zur Mausbewegung werden zur Bewegung des Betriebssystem-Cursors verwendet. Events zu einer Klick-Aktion werden verwendet um die

nötigen Klick-Events an die korrekten Fenster weiter zugeben. Die Multi-/Remotemaus wird genau diese Daten abgreifen müssen und verhindern, dass der Window-Server die rohen Daten weiter verarbeitet.

Bei Windowssystemen ist der Window-Server DWM im Einsatz. DWM steht für Desktop Window Manager. Der DWM erhält vom Betriebssystem die rohen Mausevents. Die Mausevents werden von ihm verarbeitet. Events zur Mausbewegung werden zur Bewegung des Betriebssystem-Cursors verwendet. Events zu einer Klick-Aktion werden verwendet um die nötigen Klick-Events, via Windows-Message an die korrekten Fenster weiter zugeben. Die Multi-/Remotemaus muss diese Mausevent-Daten via WindowHook abgreifen und verhindern, dass sie an den Window-Server weiter gegeben werden. Es kann nicht einfach auf den Device zugegriffen werden, wie dies mit dem Devicefile unter Linux möglich ist.

2.2.2.1 Betriebssystem-Cursor

Der Betriebssystem-Cursor wird normalerweise durch den Window-Server gesteuert. Wenn der Window-Server die rohen Mausevents erhält, so werden diese verarbeitet und gebraucht um den Cursor zu steuern. Bewegungsevents der Maus werden auf X- und Y-Koordinaten des Bildschirms gerechnet und verschieben den Cursor an die gewünschte Stelle. Tastenevents der Maus werden verwendet um Cursor-Klicks zu generieren. Der Klickevent wird nur an das nötige Fenster weitergeleitet.

Um den Cursor unter *nix-Systemen zu steuern, gibt es mehrere Ansätze. Es kann ein virtueller Mausdevice erstellt werden. Dieser wird vom Window-Server gelesen und steuert somit den Cursor. Dieser Ansatz ist aber sehr umständlich. Es besteht auch die Möglichkeit mit normalen XEvents eine Maus zu simulieren, aber auch das ist sehr umständlich. Ein XEvent ist ein roher Event, welcher von X11 verwendet wird um beliebige Events und beliebige Fenster zu propagieren. Zuletzt besteht noch ein Ansatz mit X11 eine Maus zu simulieren. Im Modul Xtest von X11 gibt es alle nötigen Funktionen, um ohne grosse Umstände das Mausverhalten zu simulieren. Das Modul ist eigentlich angedacht, um Ein Window-Serversystem zu testen, ohne manuell die Maus bedienen zu müssen. Diese Funktion kann eventuell genutzt werden, um den Cursor via unserer Software zu steuern.

Unter Windows lässt sich die Maus ganz einfach steuern. Via normalen Windows-API Aufrufen, kann der Cursor an eine bestimmte Position gesetzt werden und auch das Klick-verhalten einer Maus kann via Windows-API Aufruf erledigt werden. Das dieser Ansatz funktioniert, konnte schon in der P2-Arbeit nachgewiesen werden.

2.2.3 Zeichnen von Mauscursors

Alle bekannten Window-Server bieten dem Benutzer nur einen Betriebssystem-Cursor an. Da unser Projekt mehrere Cursor darstellen soll, müssen welche gezeichnet werden. Beim Zeichnen der Mauscursor gibt es viele kleinere Herausforderungen. Um einen Cursor zu zeichnen brauchen wir ein unsichtbares Fenster, in dem wir ein einfaches Symbol (Dreieck zur Darstellung des Cursors) zeichnen können.

Zum Zeichnen in einem Fenster gibt es beinahe Bibliotheken wie Sand am Meer. Darum sollte das Zeichnen des Dreiecks keine grösseren Probleme machen.

Die Herausforderung stellt sich hier im unsichtbaren Fenster. Normalerweise hat ein Fenster eine Hintergrundfarbe und eine vorgegebene Dekoration. Mit Dekoration ist der Rand um das Fenster, sowie der Fenstertitel und mögliche Knöpfe (Minimieren, Maximieren, Schliessen, etc.) gemeint. Beides sollte bei unserem unsichtbaren Fenster nicht vorhanden sein. Anstatt einer Hintergrundfarbe, wollen wir das aktuelle Geschehen im Hintergrund des Fensters sehen. Die Dekoration des Fensters soll auch ganz verschwinden.

Da wir mehrere Cursor zeichnen wollen, stellt sich folgende Frage: Zeichen wir mehrere Cursor in einem Fenster, oder erstellen wir mehrere Fenster mit je einem Cursor darin? Wenn wir mehrere Fenster handeln, wird etwas mehr Overhead vom Window-Server anfallen. Bei einer Lösung mit einem Fenster, würde das Zeichnen der Cursor komplexer.

2.3 Motivation

2.3.1 Mehrere Computer an einem Arbeitsplatz

Ich habe regelmässig mehr als einen Computer in Betrieb und dies am selben Arbeitsplatz. Immer und immer wieder passiert es mir dabei, dass ich die falsche Maus in der Hand habe und mich dann im ersten Moment wundere, warum mein Cursor sich nicht bewegt. Es sollte doch eigentlich möglich sein, seine Maus remote zu verwenden. Vor allem wenn die Rechner sowieso schon im selben Netzwerk angeschlossen sind. Eine normale Remote-Desktop-Verbindung aufzustellen ist hier nicht der richtige Ansatz, da ich beide Bildschirme im Blickfeld habe.

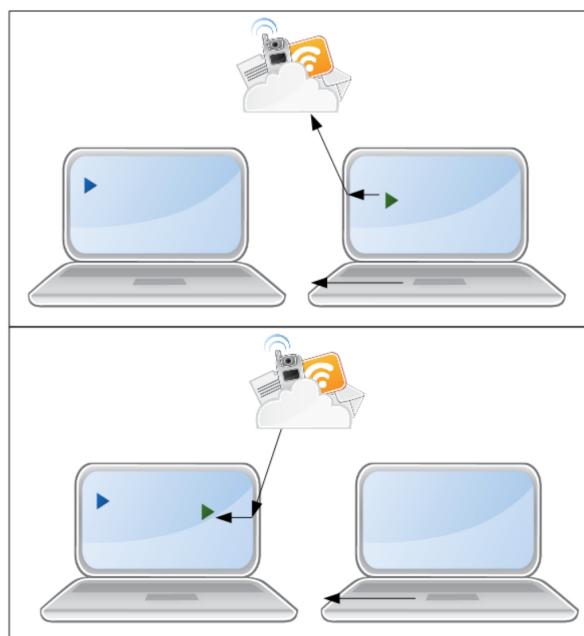


Illustration 2: Remotemaus-Funktion

2.3.2 Teamarbeit

Wir sitzen zu dritt an einer Gruppenarbeit nebeneinander. Jeder arbeitet an seinem Laptop. Das eine Gruppenmitglied findet eine wichtige Funktion eines Tools nicht. Jetzt haben wir die Möglichkeit, es ihm mit Worten zu erklären. Doch leider stellt sich das oft nicht als einfache Aufgabe heraus. Dann zeigen wir ihm am besten wie die Funktion aufzurufen ist. Jetzt beginnt der Kampf mit der Maus, wechselt man jetzt den Sitzplatz, greift man dem Andern vorne durch, um seine Maus zu bedienen? Warum kann man nicht einfach seine Maus remote verwenden, am besten gleich mit eigenem Cursor, und kann dem Kollegen, auf einfache Art weiter helfen, indem man direkt auf seinem Rechner, mit der lokalen Maus zeigt, wie die Funktion zu verwenden ist.

2.3.3 Bezug zur P2-Arbeit

In meiner P2-Arbeit, konnte ich aufzeigen, dass es eigentlich Möglich ist, eine Multi-/Remotemaus umzusetzen. Da ich dieses Tool schon länger im Hinterkopf hatte und jetzt eine simple Machbarkeitsstudie Vorlag, war das Feuer für die Multi-/Remotemaus entbrannt. Die technischen Hürden welche es für das Projekt zu meistern gilt, wirken interessant und anspruchsvoll.

2.3.4 Remote-devices

Mit einer Software wie der Multi-/Remotemaus erhält man ein Werkzeug, welches einem die Grundlage für einen beliebigen anderen Remote-Device zur Verfügung stellt. Mit dem Prinzip hinter der Multi-/Remotemaus, könnten auf die selbe Art auch andere Devices ohne grossen Aufwand remote verwendet werden. In einem ersten weiteren Schritt währen dann sicher andere Userinput-Devices interessant, sowie eine Tastatur oder ein Zeichnungspad. Doch theoretisch lässt sich die Idee auf beliebige Devices weiter ziehen.

3 Arbeitstechnik

3.1 Scope

Der Scope dieser Arbeit orientiert sich an der Aufgabenstellung, sowie am Wissen das zu erarbeiten ist, um die auftretenden Herausforderungen zu bewältigen.

Diese Arbeit wird folgende Punkte beinhalten:

- Erarbeiten von vertieftem Wissen über das Maus- und Hardwarehandling von Betriebssystemen
- Einlesen in Literatur zu Maus- und Hardwarehandling von Betriebssystemen
- Erarbeiten von vertieftem Wissen zu Window-Server und Betriebssystem-Cursor
- Erarbeiten von grundsätzlichem Wissen zu OpenGL oder einer ähnlichen Bibliothek
- Recherche nach ähnlichen oder gleichen Produkten auf dem Markt
- Aufzeigen und dokumentieren des erarbeiteten Wissens
- Erarbeiten eines sauberen Lösungsansatzes für die Multi-/Remotemaus anhand der Erarbeiteten Informationen
- Umsetzen des eigenen Lösungsansatzes für ein oder mehrere Betriebssystem(e)
- Aufzeigen und dokumentieren der praktischen Arbeit
- Aufzeigen der Erkenntnis aus dem Resultat der praktischen Arbeit
- Erarbeiten aller nötigen Dokumente zum Benutzen der Multi-/Remotemaus
- Gedanken zu möglichen Vermarktung der Multi-/Remotemaus

3.2 Out of Scope

Der Aspekt der Security wird als „out of scope“ betrachtet. Ein passendes Sicherheitskonzept für diese Applikation zu erstellen wäre zwar interessant, würde aber den Rahmen dieser Arbeit sprengen. Für einen angehenden Informatiker mit der Vertiefung Security, wäre das sicher ein anspruchsvolles Thema für eine eigene Arbeit.

Die Implementation einer Windows-Version wird als „out of scope“ betrachtet. Es wird besser eine saubere Implementation für *nix-Systeme erstellt und die Zeit dafür investiert. Besser eine qualitativ gute Software, als mehrere instabile Lösungen!

Eine geteilte Zwischenablage wird in dieser Version der Multi-/Remotemaus noch nicht zur Verfügung gestellt. Das wäre aber sicher ein wichtiges Feature für die Zukunft dieser Software und muss im Backlog gehalten werden.

3.3 Vorgehen

Während des ganzen Projekts wird ein Arbeitsjournal auf Wochenbasis geführt. Alle wichtigen und gemachten Arbeiten und alle wichtigen Erkenntnisse werden in Arbeitsjournal eingetragen. Zum Organisieren der Arbeit und zum Handling von Sourcecode, wird github verwendet. Wichtige Tasks sind als Issue zu erstellen und ihr Fortschritt aktuell zu halten.

In einem ersten Timeslice soll das nötige Wissen für die Arbeit erschaffen werden. Es geht darum Fachliteratur zu lesen, Dokumentationen zu eingesetzten Technologien zu studieren, sowie das grundsätzliche Wissen zu Betriebssystemen und dem Hardwarehandling zu festigen und zu vertiefen. Das weitere Vorgehen bei der Arbeit soll detaillierter geplant werden.

In einem zweiten Timeslice soll auf Grund der erarbeiteten Informationen ein möglicher Lösungsansatz erarbeitet werden. Alle nötigen Informationen welche zur Softwareentwicklung gebraucht werden, werden in UML-Diagrammen dargestellt. Es soll erster „proof of concept“-Code geschrieben werden, damit sichtbar wird, ob der entwickelte Lösungsansatz auf gutem Weg ist. Mit Hilfe des erstellten Codes, den erarbeiteten Informationen und dem Wissen aus der P2-Arbeit sollte die Implementation der Software möglichst weit voran getrieben werden.

Im letzten Timeslice, wird die aktuelle Software fertig geschrieben. Es gilt möglichst viele erkannte Fehler zu korrigieren und ein optimales Produkt zu erhalten. Dazu wird aus den Informationen und dem Wissen aus dem Projekt die Dokumentation erstellt.

3.4 Zeitplanung

3.4.1 Zeitplan

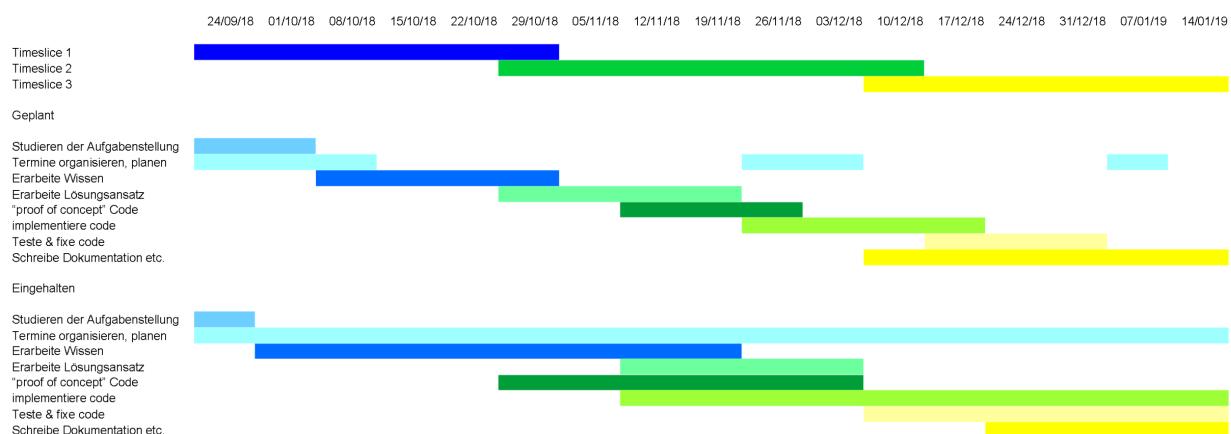


Illustration 3: Timetable - Zeitplanung

Die Zeitplanung wurde auf die Woche genau durchgeführt. Zuerst wurde eine grobe Unterteilung in drei Timeslices gemacht. Jedem dieser Timeslices wurden mehrere grobe Tasks zugeteilt. Jedem groben Task wurde dann ein vermuteter Zeitaufwand gegeben. Anhand dieser Daten wurde der Soll-Zeitplan erstellt.

3.4.2 Fazit Zeitplanung

Der Zeitplan war nur auf Wochen aufgeteilt und liess dadurch einen gewissen Spielraum zu. Trotzdem konnte nicht ganz am geplanten Ablauf festgehalten werden.

Um auf Anhieb einen richtigen Lösungsansatz zu finden, habe ich mich dafür entschieden, längere Zeit in das Erarbeiten vom nötigen Fachwissen zu stecken. Ich konnte mir vertiefte Informationen zu den Themen, Maus-Handling, X11 Window-Server und OpenGL erarbeiten. Durch das zusätzlich erarbeitete Wissen, konnte zu Beginn ein sauberer Lösungsansatz entwickelt werden, welcher bis zum Schluss nicht gross geändert werden musste. Auch bei der Implementation war das zusätzliche Wissen von grosser Hilfe.

Um auch bei der Implementation nicht auf „try and error“ angewiesen zu sein, habe ich mich entschlossen, mehr „proof of concept“-Code zu schreiben. Viel von diesem Code konnte später in der richtigen Implementation eingesetzt werden.

Obwohl bei der Implementation des Codes schnell gute Resultate zu sehen waren, so musste ich doch feststellen, dass die geplante Zeit für eine saubere Umsetzung nicht reichen wird. Dadurch kam der Entschluss die Zeit zum Umsetzen der Lösung, bis ans Projekt Ende zu verlängern, sowie einige Wochen früher zu beginnen.

4 Stand der Technik

4.1 Synergy

Synergy ist eine inzwischen kostenpflichtige Software, welche auf einem opensource Kern basiert. Synergy erlaubt die Maus und die Tastatur über mehrere Computer hinweg zu benutzen. Es können fertige Binaries gegen Bezahlung heruntergeladen werden, oder man lädt sich den Sourcecode des Kerns herunter und kompiliert es sich selbstständig. Das Projekt unterstützt, für die meisten Funktionen, alle Betriebssysteme (Linux, Mac OS und Windows)

Es werden folgende Funktionen von Synergy angeboten:

- Maus und Tastatursharing
- Zwischenablagensharing
- Drag and Drop von Files (nur Windows und Mac OS)
- Bildschirmschoner Synchronisation
- Hintergrundservice (Windows)
- Hotkeys, Keyswaping
- SSL Encryption (pro Edition)

Grob ausgedrückt unterstützt Synergy den Teil der Remotemaus meines Projektes. Dazu haben sie schon einige Features implementiert, welche ich mir nur als „nice to have“ vorgenommen habe, wie z.B. das gemeinsame Benutzen der Zwischenablage, oder die Verschlüsselung der Daten. Es bietet auch mehrere Funktionen, welche von meinem Projekt nicht abgedeckt werden, wie die Bildschirmschoner Synchronisation oder das setzen von Hotkeys. Trotz dieser zusätzlichen Funktionen, kann Synergy die Multi-/Remotemaus nicht ersetzen. Denn Synergy lässt einem die Maus und Tastatur zwar über mehrere Rechner verwenden. Es steht einem aber immer nur der Betriebssystem-Cursor zur Verfügung. Die Funktion der Multimaus ist somit nicht abgedeckt.

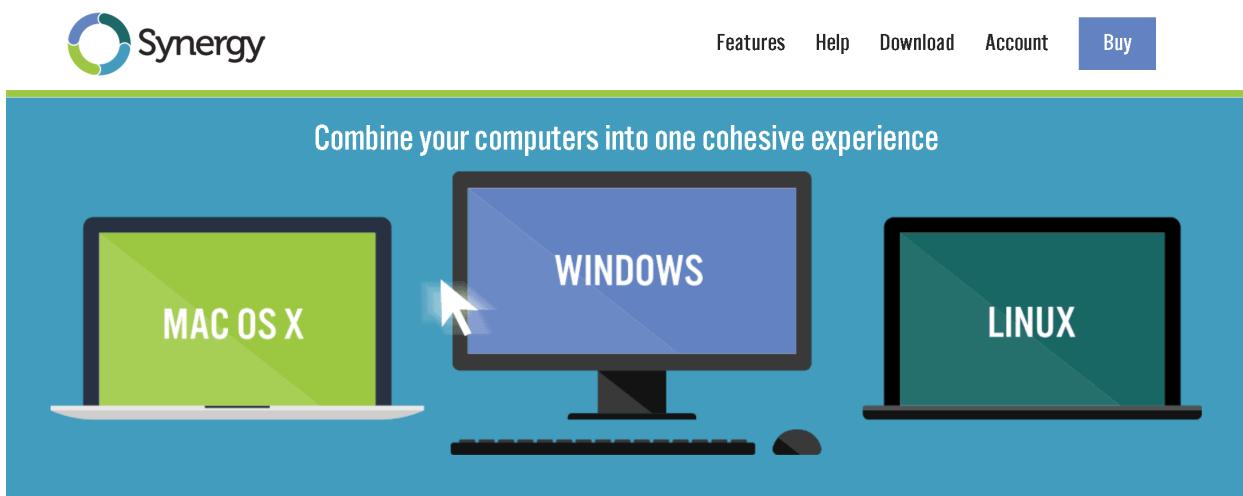


Illustration 4: Screenshot Synergy Homepage - <https://symless.com/synergy>

4.2 Logitech Flow

Logitech bietet seit einiger Zeit Mäuse an, welche dem Business Segment angehören und ein Maus- und ein Zwischenablagensharing anbieten. Das ganze Feature wird von Logitech als Multi-Computer Control bezeichnet und läuft unter dem Namen Logitech Flow. Leider konnte ich, ohne in Besitz einer der Mäuse zu gelangen, die Software von Logitech nicht genauer studieren. Anhand des Werbevideos von Logitech und deren Homepage,

scheint es sich aber wirklich nur um ein reines Maussharing mit Zwischenablagensharing zu handeln. Des weiteren wird die Software im Moment nur für Mac OS und Windows angeboten.

Das Flow Tool von Logitech wird zur Zeit von folgenden Mäusen unterstützt:

- MX Master 2S
- MX Anywhere 2S
- M720 Triathlon
- M590 Multi-Device silent
- M585 Multi-Device

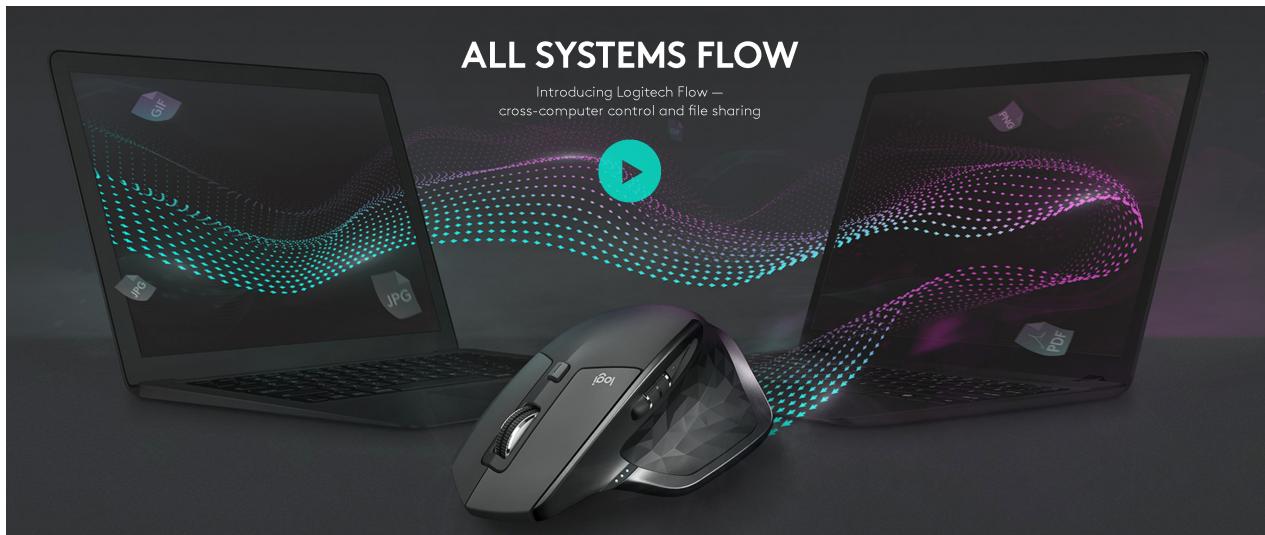


Illustration 5: Screenshot Logitech Flow Homepage - <https://www.logitech.com/en-us/product/options/page/flow-multi-device-control>

Ein grosser Nachteil dieser Lösung ist, dass man herstellerabhängig ist und auch, dass man zum Kauf von spezifischer HW gezwungen ist, um die Logitech-Lösung zu verwenden. Wie auch Synergy, bietet die Logitech-Lösung eine Remotemaus mit zusätzlicher Zwischenablage an. Aber auch hier fehlt wieder die Funktionalität der Multimaus, es steht einem pro Computer nur der Betriebssystem-Cursor zur Verfügung.

4.3 Microsoft Multipoint

Microsoft Multipoint bietet Services für Windows-Server an, sowie eine API für eine Art Multimaus.

4.3.1 Multipoint Maus

Die Multimaus von Microsoft Multipoint bietet einem ein SDK um eine lokale Multimausfunktion zu implementieren. Ein SDK ist ein Service Development Kit, das ist ein Tool das hilft, einen bestimmten Service oder ein bestimmtes Tool, in seine Applikation zu implementieren. Es ist ein ähnliches Tool wie ein Framework.

Mit Hilfe dieses SDKs kann jeder HW Maus ein Cursor in der Applikation zur Verfügung gestellt werden. Die Applikation kann jetzt für einen beliebigen Nutzen programmiert werden.

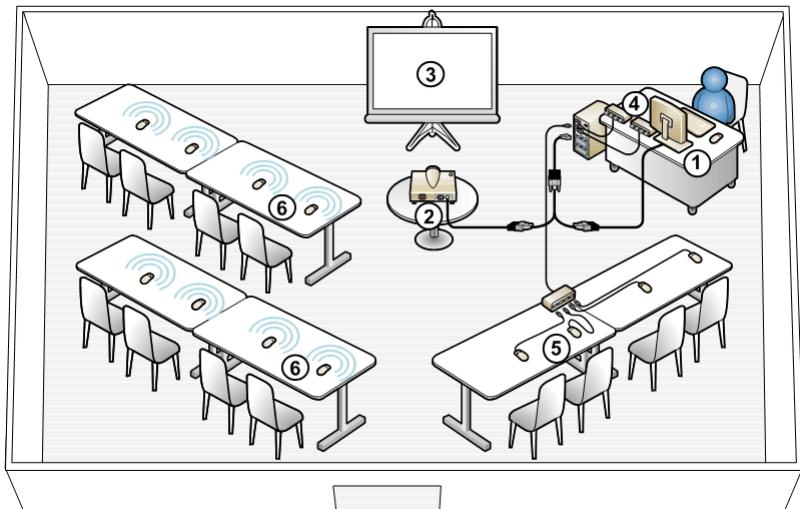


Illustration 6: Einsatz Multipoint Maus - <https://msdn.microsoft.com/en-us/library/windows/desktop/ff853261.aspx>

Dieses SDK bietet genau die nötige Funktion für den Multimausteil der Multi-/Remotemaus. Es ist nicht direkt ein Konkurrenzprodukt und kann evtl. Bei der Windows-Umsetzung genutzt werden, falls der OpenGL Ansatz unter Windows nicht funktioniert. Wenn das SDK aber in der Windows-Lösung eingesetzt wird, so wird wohl unsere grundsätzliche Software-Architektur die Windowslösung nicht mehr korrekt Abbilden.

4.3.2 Multipoint Server

Multipoint Server ist ein Service welcher von Windows-Servern angeboten wird. Dieser Service erlaubt es, dass mehrere Benutzer einen Computer teilen können. Jeder Benutzer soll das Gefühl erhalten, einen eigenen Windowsrechner zur Verfügung zu haben. Das Szenario ist, dass auf einem Server der Multipoint Service läuft. Über USB angeschlossene Mäuse und Tastaturen, sowie die angeschlossenen Bildschirme, können verwendet werden um mehrere Rechner zu simulieren. Die andere Möglichkeit ist, sich mit einer Remote-Desktop-Verbindung auf den Rechner zu verbinden und diesen zu Nutzen.

Diese Funktion unterscheidet sich stark von der Multi-/Remotemaus. Anstatt mehrere Cursor auf einem System zu erlauben, oder einen Cursor Remote steuern zu lassen, so wird es erlaubt, jedem Nutzer eine eigene Windows-Session zu haben. Dies kann seinen Nutzen haben, deckt aber nicht das Problem der Multi-/Remotemaus ab.

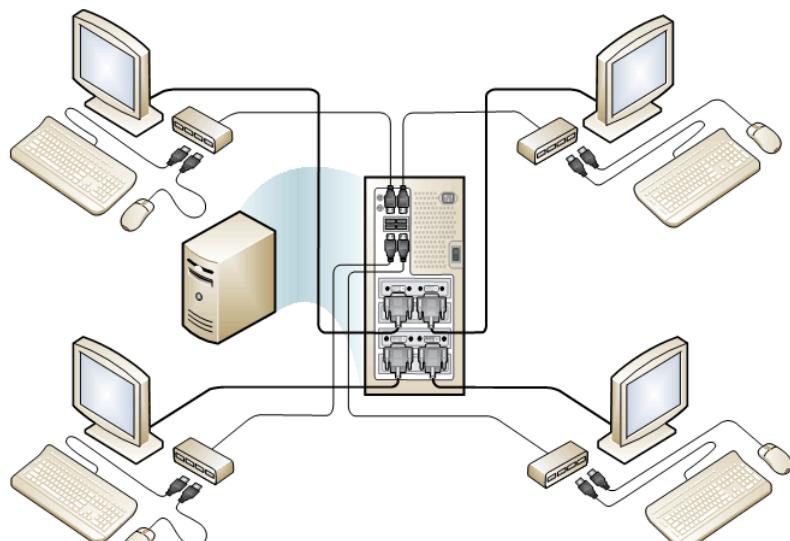


Illustration 7: Einsatz Multipoint Server (Beispiel 1) - <https://docs.microsoft.com/en-us/windows-server/remote/multipoint-services/multipoint-services-stations>

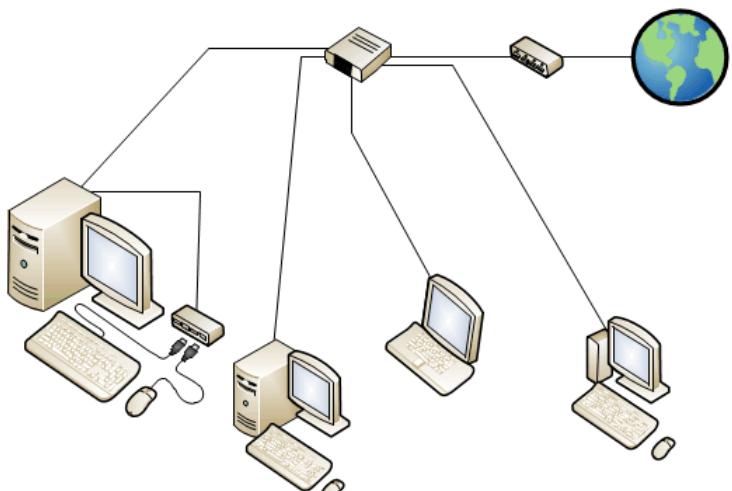


Illustration 8: Einsatz Multipoint Server (Beispiel 2) -
<https://docs.microsoft.com/en-us/windows-server/remote/multipoint-services/multipoint-services-stations>

5 Lösungsansatz

Die Multi-/Remotemaus soll zwei Szenarien umsetzen. Das erste Szenario ist die Multimaus. Auf einem Rechner müssen mehrere Cursor dargestellt werden. Jeder dieser Cursor wird einer lokalen oder einer remoten Maus zugeordnet. Das zweite Szenario ist die Remotemaus. Hier müssen die Daten einer lokalen Maus an einen Remote-Rechner gesendet werden. Der Remote-Rechner muss diese Daten empfangen können und damit einen lokalen Cursor steuern. In beiden Szenarien müssen die Mausevents an den Betriebssystem-Cursor weitergegeben werden können.

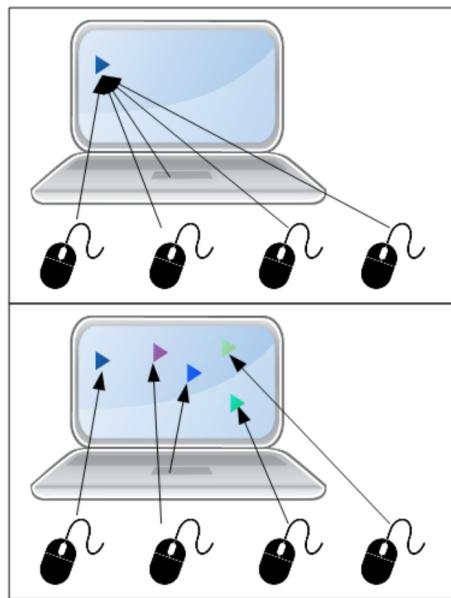


Illustration 9: Multimaus Lösungsansatz

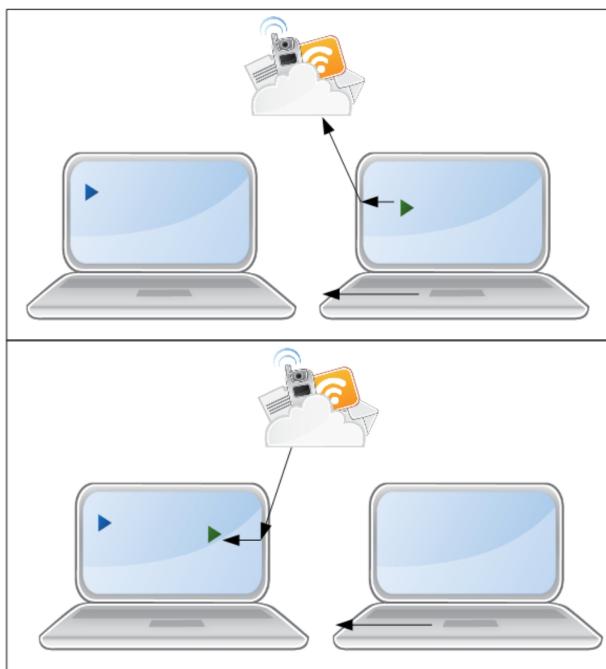


Illustration 10: Remotemaus Lösungsansatz

Die Multi-/Remotemaus verfolgt einen Lösungsansatz nach dem Prinzip eines „man in the middle“. Ein „man in the middle“ stellt ein Prinzip in der IT-Security dar. Wir haben ein Objekt A, welches mit einem Objekt B kommuniziert. Der „man in the middle“ ist jetzt ein Objekt C, welches sich zwischen A und B stellt. Unser „man in the middle“ kann jetzt die Kommunikation von A und B mit lesen und sogar manipulieren.

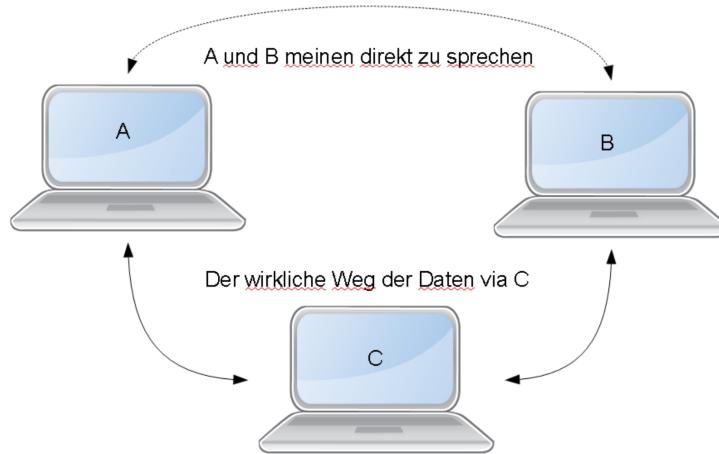


Illustration 11: "man in the middle"

Dieses Prinzip will sich die Multi-/Remotemaus für ihren Lösungsansatz beziehen. Anstatt das sich unser Tool zwischen zwei kommunizierende Computer setzt, wollen wir, dass es sich zwischen das Betriebssystem und den Window-Server platziert. Wenn es diesen Kanal unter Kontrolle hat, kann es alle rohen Mausevents abfangen. Mit den abgefangenen Informationen können wir alle nötigen Sachen machen. Der Mausevent kann gebraucht werden, um unseren Cursor zu bewegen, ohne den Betriebssystem-Cursor zu steuern oder wir können die Daten remote versenden, damit unsere Maus einen Cursor auf einem anderen Rechner steuert.

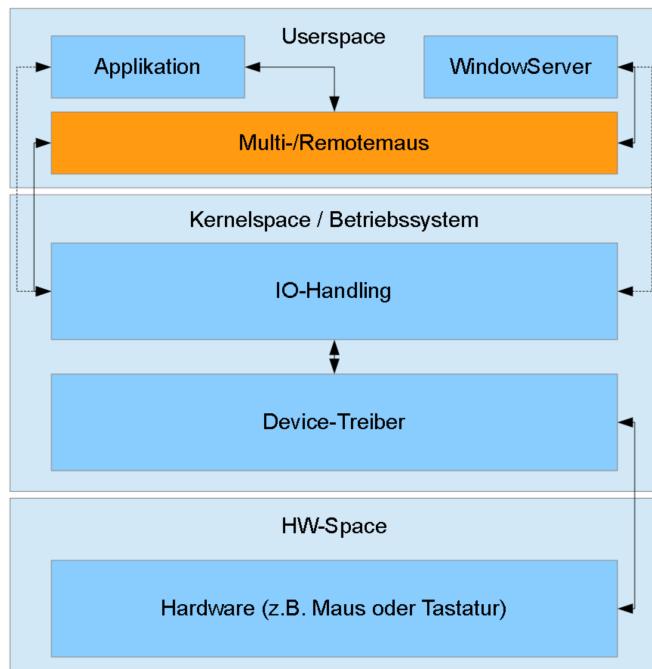


Illustration 12: "man in the middle" Multi-/Remotemaus Lösungsansatz

5.1 Maus-Handling

Für jede vorhandene Hardwaremaus welche konfiguriert ist, wird ein Mausobjekt erstellt. Dieses Mausobjekt liest alle rohen Mausevents von dieser Maus aus und gibt sie weiter an die Multi-/Remotemaus. Die Multi-/Remotemaus verarbeitet nun diese Mausevents. Sie überprüft ob sich der Cursor, welcher zur Maus gehört, sich auf dem lokalen Bildschirm befindet. Wenn ja, wird der Mausevent an den dazugehörigen Cursor übergeben, der Cursor führt nun die nötigen Bewegungen und Aktionen aus. Befindet sich der Cursor der Maus auf einem Remotebildschirm, so wird der Mausevent via Client an den Remotecomputer gesendet. Ist der Cursor gerade dabei den lokalen Bildschirm zu verlassen, so wird zusätzlich zum Mausevent, ein screen-change

Event via Client versendet. Mit diesem screen-change Event wird der Remotecomputer benachrichtigt, das er die Mausevents unserer lokalen Maus verarbeiten soll. In der Remotesituation erhält der Server unseres Tools einen screen-change Event. Durch den screen-change Event beginnt der Server nun, der Multi-/Remotemaus die von einem Client versendeten Mausdaten zu zuspielen. Die Multi-/Remotemaus verarbeitet diese Daten wieder nach den gleichen Kriterien, wie den Mausevent einer lokalen Maus. Der einzige Unterschied ist, dass Mausevents welche zu einem anderen Screen gehören nicht weiter gesendet werden müssen, es muss nur der screen-change Event zum Wechseln des Remoterechners gesendet werden.

5.2 Netzwerk / Kommunikation

Bei der Kommunikation wird auf ein Server/Client Model gesetzt. Jeder Rechner welcher Remotemäuse zulässt, startet einen Server auf, welcher auf einem konfigurierbaren Port arbeitet. Jeder Rechner welcher seine Mäuse Remote verwenden will, benötigt einen Client. Der Client wird verwendet, um Mausevents und screen-change Events zu versenden. Der Server wird verwendet um Mausevents und screen-change Events von Clients zu erhalten.

5.2.1 Sockets

Zur Umsetzung wird wohl eine Socket basierte Lösung angewandt. Sockets sollten unter allen Betriebssystemen vorhanden sein und wir bilden keine unnötigen Abhängigkeiten zu Code von Dritten auf.

5.2.1.1 UDP Client

Der Client-Socket soll ein UDP Format zum Senden der Werte verwenden. Da der Punkt der Security „out of scope“ ist, wird der UDP Client Klartext versenden. Die Events sind als JSON zu formatieren. Der Event wird via Broadcast in das ganze Netzwerk gesendet. Auch hier ist die Security wieder ausser Acht gelassen. Der Client erhält die nötigen Events zum Senden von unserer Multi-/Remotemaus.

5.2.1.2 UDP Server

Der Server-Socket soll auf UDP Pakete hören. Der Server empfängt alle Maus- und screen-change Events. Die erhaltenen Events werden nun gefiltert. Wenn Daten zu einem auf dem Rechner konfigurierten Cursor gehören, so werden die Events an die Multi/Remotemaus weitergeleitet. Alle anderen Daten werden verworfen.

5.2.2 Datenformat

Beim Datenformat sollte auf ein bekanntes Format gesetzt werden. Um den Code zu reduzieren sollte versucht werden, sowohl Konfigurationsinformationen, sowie übertragene Daten im selben Datenformat darzustellen. Vermutlich wird sich eine definiertes Dateiformat wie JSON, XML oder YAML gut eignen. Noch wichtiger als das Dateiformat ist das Datenformat selbst. Wir haben eigentlich zwei Arten von Daten in unserem System. Konfigurationsinformationen, welche helfen unser Tool zu konfigurieren, dazu gehört das Konfigurieren von den lokalen Mäusen, den lokalen Cursors, den remote Cursors und den Remoterechnern. Und es gibt die Übertragungsdaten, hierzu gehören die Events welche wir über das Netzwerk versenden. Bei diesen Events handelt es sich um Mausevents und screen-change Events.

5.2.2.1 Konfigurationsinformationen

Das Programm sollte in mehreren Punkten konfigurierbar sein.

Folgende Information müssen in einem Konfigurationsfile gespeichert werden können:

- Welche Mäuse vom Programm benutzt werden dürfen
 - Eine ID für die Maus
 - Devicefile
 - Lokale Cursorfarbe
 - Und die DeviceNumber vom Window-Server
- Welche Remote-Mäuse Zugriff auf unseren Rechner haben und welche Rechte
 - die ID der Maus
 - die nötigen Rechte-Flags

- Remote Cursorfarbe
- Auf welche Remote Rechner unsere Maus Zugriff hat
 - Position des Rechners (Links/Rechts)
 - IP Adresse

5.2.2.2 Übertragungsdaten

Es gibt zwei wichtige Datensätze, welche mit unserer Software übertragen werden müssen. Dabei handelt es sich um die eigentlichen Mausevents und um die „screen-change“-Events.

Ein Mausevent muss folgende Information enthalten:

- Die ID der Maus
- Der Eventtyp
- Der Eventcode
- Das Eventvalue
- Timestamp

Ein „screen-change“-Event muss folgende Daten enthalten:

- Die ID der Maus
- Der Eventtyp als String „screenchange“
- Die Richtung in welcher der Bildschirm verlassen wird
- Der Bisherige Besitzer der Maus
- Der neue Besitzer der Maus
- Die Y-Position des Cursors
- Timestamp

5.3 Cursor Zeichnen

Um unsere Software umzusetzen müssen ein oder mehrere Cursor gezeichnet werden. Für die Umsetzung dieser Aufgabe konnte ich mich sehr einfach für OpenGL entscheiden. OpenGL bietet eine einfache Schnittstelle zum visuellen Output des Rechners, es können sowohl 2D als auch 3D Objekte im 2D-Raum dargestellt werden und OpenGL ist unter allen Betriebssystemen einfach zu implementieren.

5.3.1 OpenGL

„OpenGL is the name for the specification that describes the behaviour of a rasterization-based rendering system. It defines the API through which a client application can control this system. The OpenGL rendering system is carefully specified to make hardware implementations allowable.“

Hardware vendors, the people who make GPUs, are responsible for writing implementations of the OpenGL rendering system. Their implementations, commonly called “drivers”, translate OpenGL API commands into GPU commands. If a particular piece of hardware is unable to implement all of the OpenGL specification via hardware, the hardware vendor must still provide this functionality, typically via a software-based implementation of the features missing from hardware.“ Zitat von

https://www.khronos.org/opengl/wiki/FAQ#What_is_OpenGL.3F

Übersetzt:

„OpenGL ist der Name der Spezifikation, welche das Verhalten von rasterisierungsbasierten Renderingsystemen beschreibt. Es definiert eine API durch welche eine Applikation dieses System kontrollieren kann. Das OpenGL rendering System ist vorsichtig spezifiziert um Hardware Implementationen zu ermöglichen.“

Hardwarehersteller, welche GPUs machen, sind für das Schreiben der Implementationen von OpenGL Renderingsystemen verantwortlich. Deren Implementationen, gewöhnlich bekannt als „Treiber“, übersetzen die OpenGL API Befehle in GPU Befehle. Wenn eine spezifische Hardware es nicht ermöglicht alle OpenGL Spezifikationen via Hardware zu lösen, so muss der Hardwarehersteller immerhin, diese von der Hardware

fehlende Funktionalität via Software implementieren.“ zitierung von https://www.khronos.org/opengl/wiki/FAQ#What_is_OpenGL.3F übersetzt.

Rasterisierungsbasiertes Rendering ist, wenn aus einem Vektor Objekt ein Raster Bild erstellt wird. Ein Raster Bild ist ein Bild, welches aus einer Reihe von Punkten besteht. Diese Punkte werden auf dem Display wieder zu einem Bild zusammengefügt.

5.3.1.1 Cursor

Der Cursor sollte in OpenGL möglichst simpel gezeichnet werden. Es wird verzichtet einen Pfeil zu zeichnen. Es soll mit einfachen Formen wie Dreiecken gearbeitet werden. Jeder Cursor soll in einem eigenen Fenster gezeichnet werden. Somit muss nur das Fenster und nicht der eigentliche Cursor gehandelt werden.

5.3.2 Window-Server

Über den Window-Server muss für jeden Cursor ein Fenster erstellt werden. Jedes dieser Fenster muss über folgende Eigenschaften verfügen:

- Das Fenster ist transparent und der Hintergrund wird regelmässig neu gezeichnet, damit der transparente Hintergrund auch aktuell bleibt.
- Es dürfen keine Dekorationen des Fenster sichtbar sein
- Das Fenster darf keine Events vom Betriebssystem-Cursor entgegen nehmen

Für jede konfigurierte lokale Maus muss dem Window-Server das Verarbeiten der Mausevents untersagt werden.

Mit Hilfe vom Window-Server wird auch der Betriebssystem-Cursor gehandelt. Damit nicht mehrere Cursor den Betriebssystem-Cursor gleichzeitig bedienen, muss der parallele Zugriff geschützt werden. Ein „first come, first serve“ Prinzip sollte hier genügen. „First come, first serve“ bedeutet, dass wer die Ressource zuerst anfragt, sie auch zuerst erhält. So ein Prinzip wirkt effizient und unterbindet das Stören durch andere Mäuse während einer Klick-Aktion.

6 Architektur

6.1 Scope

Die Software soll folgende Bereiche sicher abdecken:

- Den Multi-/Remotemauscontroller. Das ist das Herzstück der Software. Hier befindet sich der Hauptteil der Business-Logik.
- Einen Konfigurationsparser, zum Einlesen eines Konfigurationsfiles und zum Einstellen unseres Tools.
- Einen Server, zum erhalten von Maus- und screen-change Events
- Einen Client, zum senden von Maus und screen-change Event
- Einen Mauscontroller zum handeln der Mausobjekte und realen Mäusen
- Einen Cursorcontroller zum Handeln der Cursorobjekte und der gezeichneten Cursor

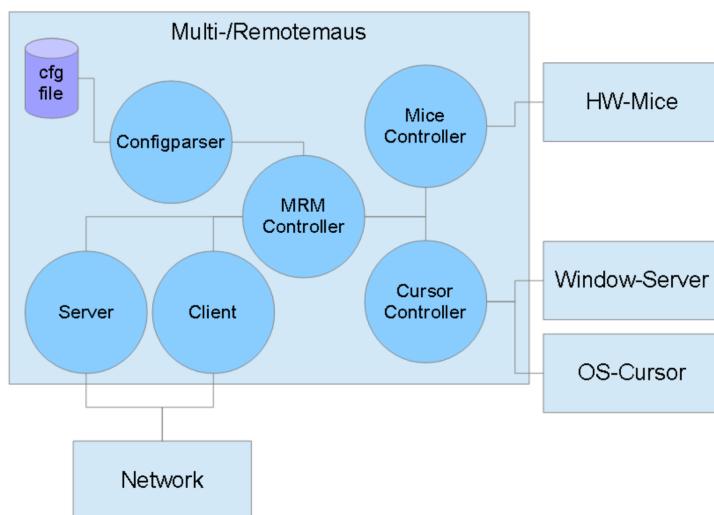


Illustration 13: Softwarescope

6.2 Klassen Diagramme

Unser Tool benötigt folgende, unten aufgeführte, Klassen zum Modellieren unserer Problemlösung. Ein Multi-/Remotemauskontroller. Dieser bildet die Businesslogik unserer Software ab. Damit er seinen Funktionen Nachkommen kann, braucht er Zugriff auf den Konfigurationsparser, den Mäusecontroller, den Cursorcontroller, den Server und den Client. Damit der Mäusecontroller auch echte Mäuse handeln kann, brauchen wir eine Mauskasse. Um dem Cursorcontroller zu ermöglichen, dass er Cursor handeln kann, brauchen wir auch noch eine Cursorklasse. Zu guter Letzt braucht es noch eine Screenklasse welche uns hilft, Remoterechner, also andere Screens für unsere Maus, darzustellen.

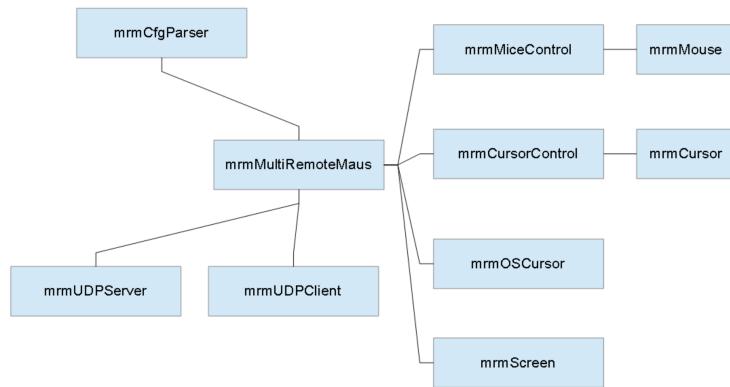


Illustration 14: Klassendiagramm (grob)

Die detaillierten Klassendiagramme sind im Anhang dieser Arbeit zu finden.

6.3 Use-Cases

Es gibt eigentlich drei Use-Cases welche unsere Software abdecken muss.

- Wenn ein Mausevent empfangen wird und lokal genutzt werden kann
- Wenn ein Mausevent empfangen wird und remote genutzt werden kann
- Wenn ein Mausevent einen Screenchange auslöst

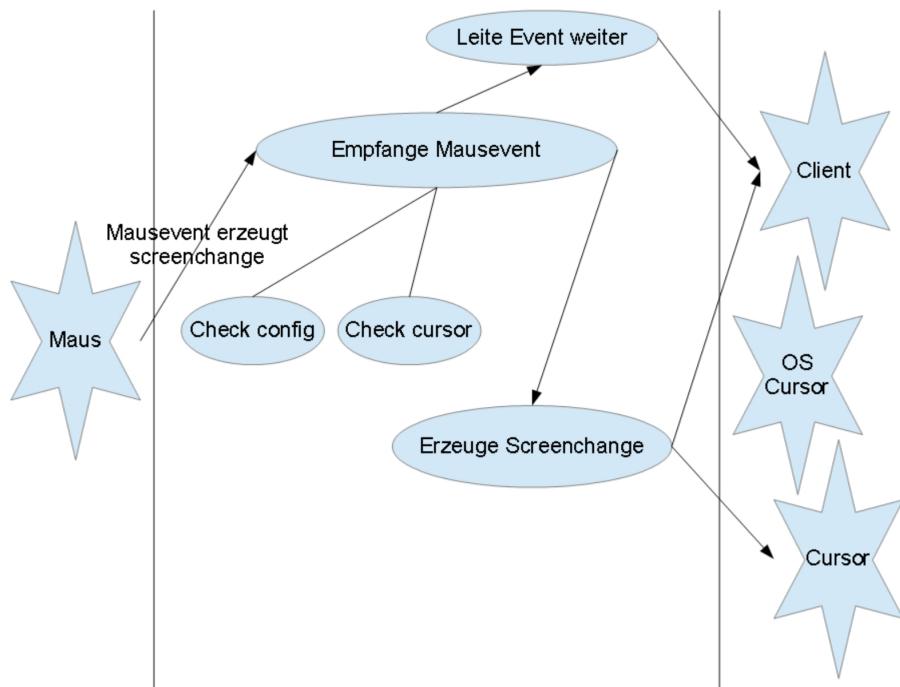


Illustration 15: Beispiel Use-Case: screen-change

Mit diesen drei Use-Cases ist sicher gestellt, dass wir unsere lokalen Cursor mit den lokalen Mäusen steuern können, dass unsere Cursor an Remoterechner weitergegeben werden, wenn man den eigenen Screen verlässt und das Daten welche über den Server erhalten werden, die lokalen Cursor steuern kann.

Alle Use-Cases können im Anhang gefunden werden. Sowie ein Sequenzdiagramm, welches den Ablauf noch genauer beschreibt.

6.4 HW

Grundsätzlich sollen alle Arten von Mäusen und Touchpads unterstützt werden, solange sich diese über eventN Devicefiles lesen lassen und nicht über die alttümlichen mouseN Devicefiles, oder das allgemeinere mice Devicefile. Um noch alte PS/2 Mäuse zu unterstützen werden für Input Devices oft noch ein mouseN Devicefile erstellt und ein allgemeines mice Devicefile, welches alle Events von allen mouseN Devicefiles enthält. So können auch ältere Applikationen noch mit dieser Schnittstelle auf alle Devices zugreifen. Bei der Multi-/Remotemaus werden aber eher neuere Inputdevices angeschaut. Diese können in jedem Fall über die eventN Files ausgelesen werden, welches der aktuelle Weg ist, eine Maus auszulesen.

Es wird sich bei der Lösung nicht auf eine spezielle Hardware festgelegt. Das wichtige für unsere Software ist, dass sich alle Rechner welche zusammen kommunizieren müssen im gleichen Netzwerk sind.

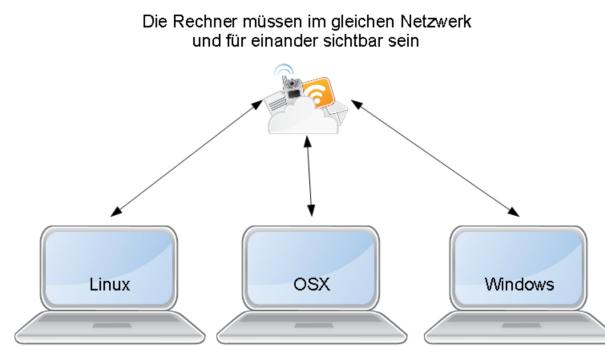


Illustration 16: Netzwerkaufbau für Tests

Zum Testen der SW wird folgende HW verwendet. Ein Linux Laptop, ein MacBook und ein Windows Laptop. Auf jedem dieser Rechner wird eine Multi-/Remotemaus laufen.

7 Implementation

Bei der Implementation geht es darum, den erarbeiteten Lösungsansatz umzusetzen und eine Software Lösung für mindestens eines oder mehrere Betriebssysteme, zu erhalten. Um die Qualität der Software hoch zu halten, habe ich mich dafür entschieden, eine Softwarelösung für *nix-Systeme zu schreiben. Diese Lösung sollte gut unter Linux, als auch unter Mac OS, laufen.

Als Programmiersprache habe ich c++ gewählt. Diese Programmiersprache bietet eine Vielzahl von Compilern unter allen Betriebssystemen an. Dazu ist die Programmiersprache sehr systemnahe, nicht wie etwa Java, welches eine Virtuelle Maschine als Layer zwischen dem Betriebssystem und der ausführbaren Datei benötigt.

Der Sourcecode ist simpel Strukturiert. Im Projektordner ist ein Ordner mit dem Namen „code“ angelegt. In diesem Ordner ist ein „inc“ und ein „src“ Ordner zu finden. Der gesamte selbstgeschriebene Sourcecode findet sich in diesen beiden Ordnern. Die Sourcefiles sind im „src“ Ordner zu finden und die Headerfiles im „inc“ Ordner.

Um das Programm zu Kompilieren und zu Linken wird die g++ Toolchain eingesetzt. Ein Compiler verarbeitet den Sourcecode zu Maschinencode. Der Linker setzt die verschiedenen Maschinencode-Objekte zusammen und verlinkt verwendete Bibliotheken. Zum Handhaben der Makefiles, unter den verschiedenen Betriebssystemen, wird cmake eingesetzt. Ein Makefile ist eine Art von Rezept zum Ausführen von Befehlen anhand von gewissen Regeln. Cmake erstellt die nötigen Regeln für ein Programm und sucht einem die benötigten Bibliotheken und passt die Pfade an. Es kann auch Fehlermeldungen generieren, falls benötigte Tools oder Bibliotheken nicht vorhanden sind. Weitere Informationen zum Kompilieren des Sourcecodes findet man im Anhang.

Um den Sourcecode zu schreiben wurde nach den erstellten UML-Diagrammen, aus dem Kapitel Architektur, vorgegangen. Alle Klassen wurden in Code umgesetzt. Jede Klasse besitzt eine eigene .cpp/.h-Datei. Wenn für ein Betriebssystem spezifischer Code geschrieben wird, so wird dieser in ein #ifdef-Tag verpackt. Das #ifdef-Tag wird vom Pre-Compiler verarbeitet. Der Pre-Compiler ist ein Tool, welches gewisse Tags im Code verarbeitet. Je nach dem werden gewisse Stellen vom Code kompiliert oder nicht. Oder es werden Makros eingesetzt usw. (Kurzinformation zum Pre-Compiler: <https://de.wikipedia.org/wiki/Precompiler>) Der Sourcecode kann unter github eingesehen werden. Im Anhang dieses Dokuments befindet sich die Codedokumentation, sowie ein Link zum github Repository.

7.1 Maus-Handling

Zu Beginn wird dem Window-Server der Zugriff auf die konfigurierten Mäuse genommen. Jede Maus welche einen Cursor zur Verfügung hat, wird vom Window-Server ausgeschlossen. Momentan wird dies noch mit einem Systemcall gelöst. Mit Hilfe des Xinput-Tools werden die nötigen Mäuse im Window-Server deaktiviert.

```
for(int i = 0; i < 5; i++){
    if(devicenumbers[i]!=0){
        stringstream command;
        command << "xinput disable " << devicenumbers[i];
        system(command.str().c_str());
    }
}
```

Illustration 17: Codeauszug - Konfiguriere Window-Server

Das Maus-Handling wird in der mrmMaus-Klasse geregelt. Im Konstruktor, beim Erstellen des Objekts, wird die ID der Maus übergeben, sowie der Pfad des Devicefiles. Das Devicefile wird gleich im Konstruktor geöffnet. Wichtig ist, dass unser Programm als sudo ausgeführt wird, oder der Benutzer unter welchem es ausgeführt wird, Teil der Gruppe Input ist.

```

mrmMouse::mrmMouse(string id, string eventfile) {
    this->id = (char *)malloc(sizeof(char)*(strlen(id.c_str())+1));
    this->eventfile = (char *)malloc(sizeof(char)*(strlen(eventfile.c_str())+1));
    strcpy((char *)this->id, (char *)id.c_str());
    strcpy((char *)this->eventfile, (char *)eventfile.c_str());
    cout << "mrmMouse: " << this->id << ": open file" << this->eventfile << endl;
    if((this->mousefile = open(this->eventfile, O_RDONLY)) == -1) {
        cout << "opening device MOUSEFILE failed" << endl;
        this->mousefile = -1;
    }
}

```

Illustration 18: Codeauszug - Öffne Devicefile

Jetzt muss noch der Thread der Maus gestartet werden, welcher dann die Maus-Events von Devicefile empfängt. Über die Methode start() wird der Mausthread gestartet.. In diesem unabhängigen Thread wird die run() Methode des Maus-Objektes ausgeführt. Diese Methode ist dafür verantwortlich, dass die Maus-Events empfangen werden und packt diese in unser JSON-Objekt.

```

void *ThreadMouse(void *pVoid) {

    mrmMouse* mouse = static_cast<mrmMouse*>(pVoid);
    mouse->run(mouse);

    pthread_exit(nullptr);
}

void mrmMouse::start() {
    cout << "mrmMouse: start " << this->eventfile << endl;
    pthread_t thread;
    // start Client
    pthread_create( &thread, nullptr, ThreadMouse, static_cast<void*>(this));
}

```

Illustration 19: Codeauszug - Erstelle Thread mit Membermethode & Helferfunktion

Mit Hilfe der Hilfsfunktion „ThreadMouse“ wird ein neuer Thread erstellt. Dazu wird die pthread Bibliothek genutzt. Die Hilfsfunktion wird „pthread_create“ übergeben, zusätzlich wird noch ein statisch gecasteter Pointer zu unserem Mausobjekt mitgegeben. Dies wird gemacht, damit die Membermethode run() des Mausobjektes in der Hilfsfunktion aufgerufen werden kann, ohne eine Speicherverletzung zu machen.

```

void mrmMouse::run(mrmMouse* mouse) {
    cout << "mrmMouse: run " << mouse->eventfile << endl;
    struct input_event mouseevent;
    int ret = 0;
    while (ret = read(mouse->mousefile, &mouseevent, sizeof(struct input_event))) {
        if(ret > 0 && !(mouseevent.type == 0 && mouseevent.code == 0 && mouseevent.value == 0)) {
            json eventobj;
            eventobj["id"] = mouse->id;
            eventobj["type"] = mouseevent.type;
            eventobj["code"] = mouseevent.code;
            eventobj["value"] = mouseevent.value;
            eventobj["time_s"] = mouseevent.time.tv_sec;
            eventobj["time_us"] = mouseevent.time.tv_usec;
            mrm->recvMouseEvent(eventobj);
            usleep(50);
        } else {
            usleep(500);
        }
    }
    cout << "error" << endl;
}

```

Illustration 20: Codeauszug - Verarbeite Mausevent zu JSON-Objekt

Sobald wir einen Mausevent erhalten, wird mit Hilfe der nlohmann_json Bibliothek, aus dem erhaltenen Inpuevento unser JSON-Objekt erstellt. Dieses JSON-Objekt wird dem globalen Multi-/Remotemausobjekt mitgeteilt. Dort wird dann entschieden, was mit dem Event passiert.

7.2 Zeichnen eines Cursors

Das Zeichnen des Cursors, sowie das Positionieren des Fensters in dem sich der Cursor befindet, wird in der mrmCursor-Klasse geregelt. Im Konstruktor werden die nötigen Eigenschaften des Cursors mitgegeben. Darunter befinden sich die ID, die Farbe des Cursors, die Rechte des Cursors, sowie die Devicenummer, welche zur Maus des Cursors gehören. Eine Cursor der von einer Remotemaus gesteuert wird, benötigt zwingend die Rechteangaben. Ein Cursor welcher von einer lokalen Maus gesteuert wird, muss die Devicenummer enthalten.

Nach dem gleichen Vorgang wie beim Maus-Handling, wird nun ein neuer Thread erstellt. In diesem Thread wird die run() Methode des Cursor-Objekts aufgerufen. In dieser Methode wird nun ein unsichtbares Fenster, ohne Dekoration, erstellt. In dieses Fenster wird mit Hilfe von OpenGL ein Dreieck gezeichnet. Wenn unser Fenster jetzt einen XEvent vom Window-Server erhält, so wird unser Cursor und unser Fenster neu gezeichnet.

Als erstes muss das Display und das Root-Window vom Window-Server geholt werden. Diese Objekte erhalten wir vom Window-Server mit der Xlib Bibliothek. Mit Hilfe dieser Daten erhalten wir die Framebuffer-Konfigurationen. Wir suchen die für uns passende Konfiguration aus diesem Konfigurations-Pool heraus. Es wird eine Konfiguration gebraucht, welche Alpha-Werte für die Transparenz zulässt. Mit Hilfe der Event- und Attributmaske, sowie einer Shape-Region wird verhindert, dass das Fenster Events vom Betriebssystem-Cursor erhält.

```

attr.colormap = this->cmap;
attr.background_pixmap = None;
attr.border_pixmap = None;
attr.border_pixel = 0;
attr.event_mask =
    StructureNotifyMask |
    EnterWindowMask |
    LeaveWindowMask |
    ExposureMask |
    ButtonPressMask |
    ButtonReleaseMask |
    OwnerGrabButtonMask |
    KeyPressMask |
    KeyReleaseMask |
    FocusChangeMask;

attr_mask =
    CWBackPixmap |
    CWC colormap |
    CWBorderPixel |
    CWO overrideRedirect |
    CWE ventMask;

attr.override_redirect = true;

this->width = 20; //DisplayWidth(Xdisplay, DefaultScreen(Xdisplay));
this->height = 20; //DisplayHeight(Xdisplay, DefaultScreen(Xdisplay));
x=0, y=0;

this->window_handle = XCompositeGetOverlayWindow(this->Xdisplay, this->xroot);
this->window_handle = XCreateWindow( this->Xdisplay,
                                         this->xroot,
                                         x, y, this->width, this->height,
                                         0,
                                         this->visual->depth,
                                         InputOutput,
                                         this->visual->visual,
                                         attr_mask, &attr);

XserverRegion region = XFixesCreateRegion (this->Xdisplay, NULL, 0);

XFixesSetWindowShapeRegion (this->Xdisplay, this->window_handle, ShapeBounding, 0, 0, 0);
XFixesSetWindowShapeRegion (this->Xdisplay, this->window_handle, ShapeInput, 0, 0, region);

XFixesDestroyRegion (this->Xdisplay, region);

```

Illustration 21: Codeauszug - Öffne ein Fenster

In das unsichtbare, eventlose Fenster, muss jetzt unser Cursor-Dreieck gezeichnet werden. Damit dies möglich wird, muss dem erstellten Fenster ein Render-Kontext hinzugefügt werden. Dieser Render-Kontext erlaubt es uns, mit OpenGL in unserem Fenster zu zeichnen

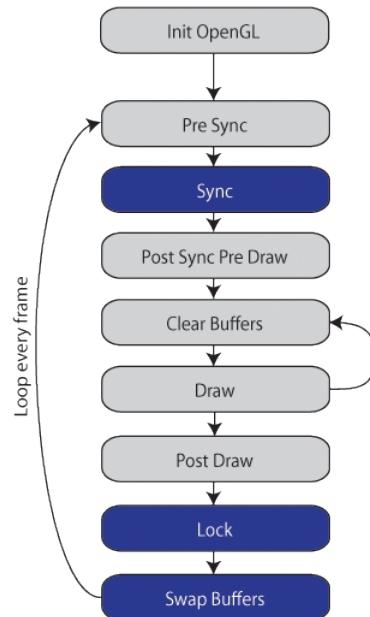


Illustration 22: Funktion von OpenGL -
<https://c-student.itn.liu.se/wiki/develop:sgct:howitworks>

Sobald sich jetzt irgend ein Event vom Window-Server ergibt, welche den Inhalt unseres Fensters angeht, so müssen wir das Fenster und unseren Cursor neu zeichnen. Um unser unsichtbares Fenster weiterhin unsichtbar zu halten, muss die Hintergrundfarbe mit OpenGL gecleared werden. Dazu wird ein 0,0,0 Farbwert und ein null-er Alpha-Wert an OpenGL übergeben. Danach muss OpenGL noch gesagt werden, welche Buffer damit gecleared werden sollen. Um den transparent Effekt zu erhalten müssen wir den GL_COLOR_BUFFER_ und den GL_DEPTH_BUFFER_ clearen.

```

void mrmCursor::redrawTheWindow() {
    //
    XWindowAttributes winattr;

    XGetWindowAttributes(this->Xdisplay, this->glX_window_handle, &winattr);
    glViewport(0, 0, winattr.width, winattr.height);

    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f((1.0f/255)*this->color_r, (1.0f/255)*this->color_g, (1.0f/255)*this->color_b);

    glBegin(GL_TRIANGLES);

    glVertex3f(-1.0f, -1.0f, 0.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    glVertex3f(-1.0f, 1.0f, 0.0f);

    glEnd();

    glXSwapBuffers(this->Xdisplay, this->glX_window_handle);

    XMoveWindow(this->Xdisplay, this->glX_window_handle, this->posx, this->posy);
}

```

Illustration 23: Codeauszug - Zeichne das Fenster und den Cursor

Jetzt ist das Zeichnen des Cursors dran. Um den Cursor zu zeichnen müssen wir zuerst die Farbe des Cursors an OpenGL übergeben. Jetzt starten wir den Zeichnungsprozess von OpenGL und melden der Bibliothek, dass wir ein Dreieck zeichnen wollen. Jetzt müssen wir OpenGL nur noch drei Vektoren übergeben, welche die Position der Ecken unseres Dreiecks definieren.

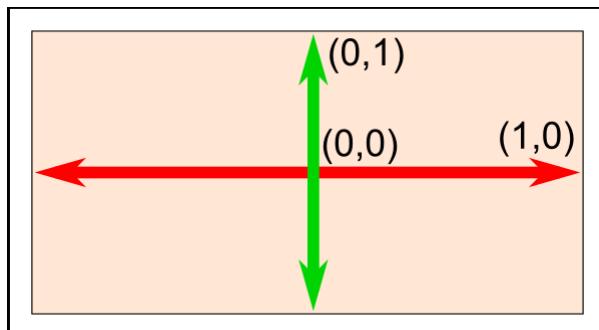


Illustration 24: 2D-Koordinatensystem in OpenGL -
<http://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/>

7.3 Datenverarbeitung

Die mrmMultiRemotemaus-Klasse ist, zusammen mit der mrmMiceControl-Klasse und der mrmCursorControl-Klasse, für die Datenverarbeitung und den Grossteil der Business-Logik verantwortlich. Der MultiRemotemaus-Controller kennt alle Objekte wie den Configurationsparser, den Server und den Client. Über das mrmMiceControl-Objekt und über das mrmCursor-Objekt, hat der MultiRemotemaus-Controller Zugriff auf alle Mäuse und Cursor.

Erhält eines der erstellten Maus-Objekte eine Mausevent, so wird es die Empfangsfunktion des MultiRemotemaus-Controllers aufrufen. Zuerst wird überprüft, ob der Maus-Event an einen lokalen Cursor weiter gegeben werden muss. Ist dies der Fall, so wird der Event dem passenden Cursor übergeben. Führt der Event dazu, dass der Cursor unseren Bildschirm verlässt, so muss unser Controller dem mrmUDPClient mitteilen, dass er einen screen-change Event versenden muss. Nach diesem erstellten Event, wird noch der original Mausevent über den Client versendet. Wird von Anfang an erkannt, dass der Maus-Event nicht zu einem lokalen Cursor gehört, so wird dieser direkt über den Client versendet.

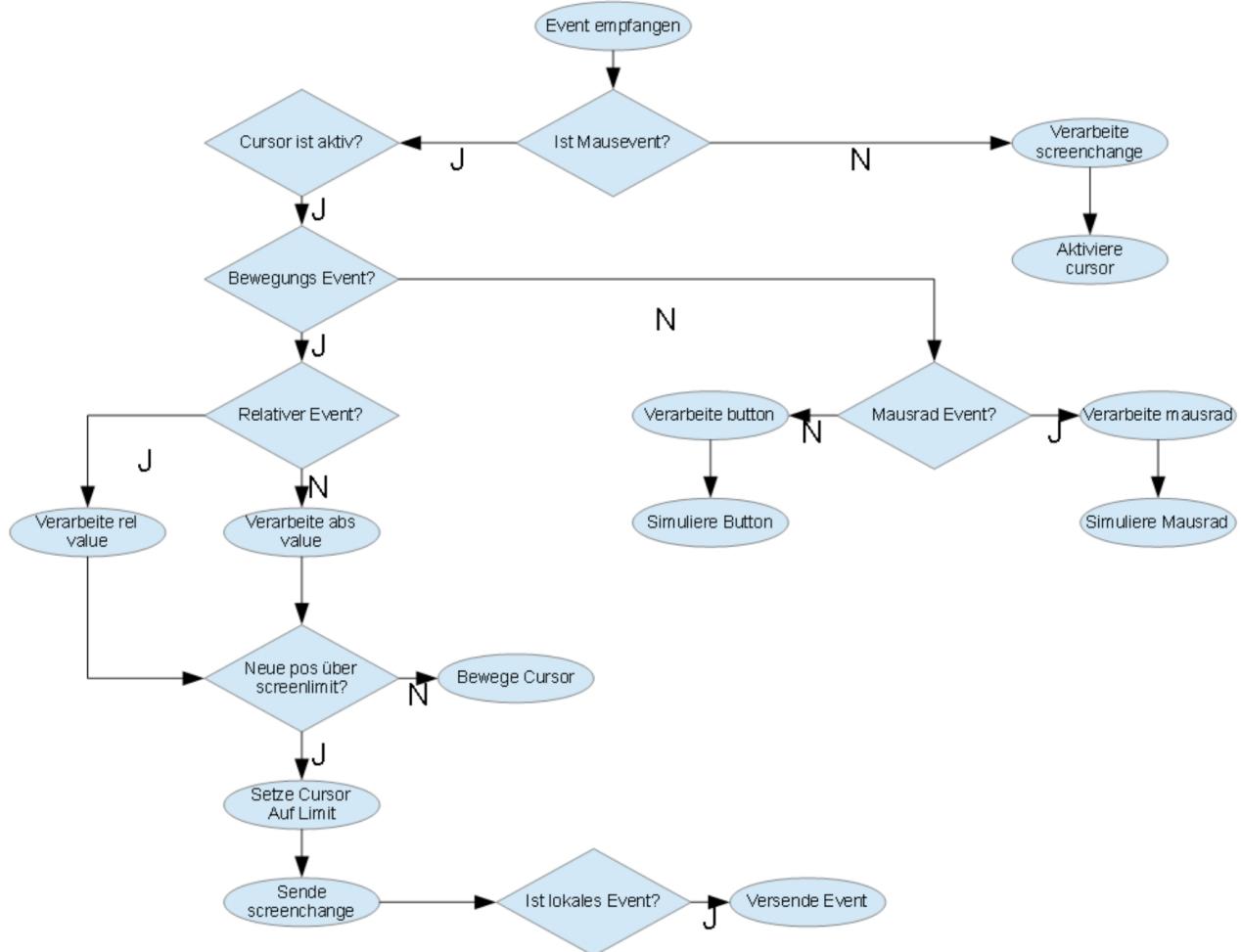


Illustration 25: Flussdiagramm - Eventhandling

Erhält unser mrmUDPServer einen screen-change Event, so informiert er den mrmMultiRemotemaus-Controller über diesen Event. Der Controller verarbeitet diesen Wert und aktiviert, falls nötig, einen der lokal dargestellten Cursor. Wenn der Server jetzt von der Maus hinter dem screen-change Event einen Mausevent erhält, so wird dieser auch an den Controller weiter geleitet. Der Event wird dem passenden Cursor übergeben. Führt der Event dazu, dass der Cursor unseren Bildschirm verlässt, so muss unser Controller dem mrmUDPCClient mitteilen, dass er einen screen-change Event versenden muss. Da es sich um einen Mausevent von einem Remote-Device handelt, muss der Maus-Event nicht weitergeleitet werden.

7.4 Kommunikation

7.4.1 UDP Client

Der UDP Client wird in der mrmUDPCClient-Klasse implementiert. Zur Umsetzung des UDP Clients wird ein UDP-Socket implementiert. Der UDP-Socket wird mit dem Port, welcher dem Konstruktor übergeben wird, geöffnet.

```

mrmUDPClient::mrmUDPClient(int port) {
    this->s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    cout << "mrmUDPClient: socket = " << this->s << endl;
    this->addrout.sin_family = AF_INET;
    this->addrout.sin_port = htons(port);
    this->addrout.sin_addr.s_addr = htonl(INADDR_BROADCAST);
    int ret = 0;
    // activate rights for broadcasting on socket
    int trueflag = 1;
    if (this->s == -1) {
        cout << "mrmUDPClient: socketinit error" << endl;
    }
    if ((ret = setsockopt(this->s, SOL_SOCKET, SO_BROADCAST, &trueflag, sizeof(trueflag))) < 0) {
        cout << "mrmUDPClient: setsockopt " << ret << endl;
    }
}

```

Illustration 26: Codeauszug - öffne einen Socket

Das Versenden der Events kann mit einem einfachen write() Aufruf erledigt werden. Es wird eine Methode zum senden von Mausevents und eine Methode zum senden von screen-change Events implementiert. Beide schreiben via write auf den UDP-Socket.

```
ssize_t ret = sendto(this->s, &sendevent, sizeof(sendevent), 0, (struct sockaddr*) &this->addrout, sizeof(this->addrout));
```

Illustration 27: Codeauszug - Versende einen Event

7.4.2 UDP Server

Der UDP Server wird in der mrmUDPServer-Klasse implementiert. Zur Umsetzung des UDP Servers wird ein UDP-Socket implementiert. Der UDP-Socket wird mit dem Port, welcher dem Konstruktor übergeben wird, geöffnet. Das Öffnen des Sockets funktioniert gleich wie beim UDP-Client. Der Socket muss einfach zum Lesen und nicht zum Schreiben konfiguriert sein.

Wird ein Event empfangen, so wird überprüft, ob der Event für einen unserer System relevant ist. Ein Event ist dann relevant, wenn die Absender IP nicht unsere eigene IP ist. Ist das der Fall, so wird der Event der Multi-/Remotemaus übergeben und wie in der Datenverarbeitung beschrieben, bearbeitet.

```

void mrmUDPServer::run() {
    cout << "mrmUDPServer: run" << endl;
    char dgram[1024];
    for (;;) {
        struct sockaddr_in srcaddr;
        socklen_t addrlen = sizeof(srcaddr);
        ssize_t ret = recvfrom(this->s, dgram, sizeof(char)*1024, 0, (struct sockaddr *)&srcaddr, &addrlen);
        //ssize_t ret = recv(this->s, dgram, sizeof(char)*1024, 0);
        if (ret < 0) {
            cout << "mrmUDPServer: recv error" << endl;
            break;
        } else {
            if(this->hostip.compare(inet_ntoa(srcaddr.sin_addr))!=0) {
                if(dgram[0]!='n' && dgram[1]!='u' && dgram[2]!='l' && dgram[3]!='l') {
                    json temp = json::parse(dgram);
                    mrm->recvMouseEvent(temp);
                    cout << "received from: " << inet_ntoa(srcaddr.sin_addr) << endl;
                }
            }
        }
    }
}

```

Illustration 28: Codeauszug - Empfange Daten

8 Resultat

Aus dem praktischen Teil der Arbeit entstand eine stabile Software welche die Grundfunktionalität der Multi-/Remotemaus umsetzt. Die Software ist unter Linux und Mac OS lauffähig. Unter Mac OS sind nicht alle funktionen verfügbar. Das fertige Resultat wurde auf einem Laptop mit Linux Mint, sowie mit einem MacBook getestet.

Die fertige Software erlaubt es, mehrere Cursor auf einem Computer zu haben. Diese Cursor können entweder von einer lokalen Maus, oder von einer Remotemaus verwendet und gesteuert werden. Solange kein anderer Cursor eine Klick-Aktion ausführt, kann ein Cursor den Zugriff auf den Systemcursor erhalten und einen Klick ausführen. Die Multimausfunktion wurde mit zwei lokalen und einer remote Maus getestet. Im gleichen Zug wurde die Remotemausfunktion getestet, da einer der lokalen Cursor von einer Remotemaus auf einem anderen Rechner gesteuert wurde.

In den letzten Tests unter Linux sind keine groben Bugs mehr aufgetaucht. Es gibt einige Programme, welche es schaffen, gewisse Menüs vor dem Mauscursor der Remotemaus zu zeichnen (z.B. Firefox, ein Web Browser). Dieses Problem ist aber rein optisch, denn der Betriebssystem-Cursor kann trotzdem an die korrekte Stelle gefahren werden und ein Klick kann ausgeführt werden.

8.1 Möglicher Markt

Die Multi-/Remotemaus läuft schon sehr stabil, dennoch sollten einige Punkte vor Marktreife noch angepasst werden oder sogar noch einige Features hinzugefügt werden. Die Software läuft unter GPL Lizenz und ist somit Opensource. Die GPL Lizenz sichert folgende Punkte ab.

Erlaubt:

- Kommerzielle Benutzung
- Modifikationen
- Distributionen
- Benutzung in Patenten
- Private Benutzung

Limitiert:

- Haftung
- Garantie

Bedingungen:

- Lizenz und Copyright-Erklärung muss in der Software mitgegeben werden.
- Änderungen müssen dokumentiert werden
- Sourcecode muss zugänglich gemacht werden, wird die Software vertrieben
- Änderungen müssen unter der selben Lizenz erfolgen

Ein Möglicher Markt für die Software ist zum Beispiel der Ausbildungssektor. Immer mehr und immer früher, werden Computer in der Bildung eingesetzt. Die Multi-/Remotemaus kann einem bei der Teamarbeit am Computer den Arbeitsprozess stark vereinfachen.

Ein weiterer Markt bietet auch die Arbeitswelt. Wer regelmässig mit mehreren Computern am selben Arbeitsplatz arbeitet, könnte auch stark profitieren. Die Multi-/Remotemaus erlaubt es einem, ohne irgendwelche Remote-Desktop-Verbindungen, alle Rechner mit einer Maus zu bedienen. Remote-Desktop würde in so einer Arbeitsumgebung nur unnötig das Netz belasten, da für die Bildübertragung viele Daten gesendet werden. Da unser User auf alle Bildschirme zur selben Zeit sieht, ist die Multi-/Remotemaus das optimale Tool.

Eine Möglichkeit wäre, das Tool als opensource weiter zu führen. Man könnte am Beispiel von Synergy den Sourcecode offen stellen, für fertig kompilierte Binaries des Programms einen gewissen Betrag zur Kostendeckung verlangen. Wie bei vielen Opensource Projekten, könnte daneben noch ein Spendenkonto geführt werden, wo über PayPal oder eine ähnliche Technologie, ein frei wählbarer Betrag überwiesen werden kann.

9 Schlussfolgerungen/Fazit

Über die ganze Arbeit konnte ich sehr viel neues und vertieftes Wissen über die verschiedenen Betriebssysteme lernen. Im Grossen und Ganzen kann sich der erarbeitete Lösungsansatz sehen lassen. Die gewählte Problemlösung so wie das Handeln der Mausevents klappt wie geplant. Das Erfassen der Mausdaten würde ich, vor allem unter Mac OS, heute anders angehen.

Die meisten der technischen Hürden konnten schon im ersten theoretischen Teil der Arbeit gut aufgedeckt werden. Vieler der Herausforderungen waren dadurch bekannt. In der Ausgangslage sind die wichtigsten dieser Erkenntnisse zusammengefasst.

Es konnte ein interessanter Einblick in den aktuellen Markt gewonnen werden. Durch die Recherchen von Konkurrenzprodukten konnte klar aufgezeigt werden, dass ein Tool wie die Multi-/Remotemaus sicher Anklang finden wird.

Die gewählte Architektur zur Umsetzung der Software ist grundsätzlich gut gewählt. Es mussten keine groben Anpassungen an der Software-Architektur gemacht werden. Mit dem gewählten Ansatz, konnten alle nötigen Use-Cases abgedeckt werden.

Die grösste Herausforderung stellte sich in der Implementation der Arbeit. Die Linux-Version unserer Software konnte ohne grosse Umstände umgesetzt werden. Einzig das Ansteuern des Systemcursors hat sich als umständlicher herausgestellt. Doch mit der Hilfe des Xtest-Moduls vom WindowServer, konnte auch das gelöst werden. Da auch MacOS ein Linux ähnliches Betriebssystem ist, habe ich dies als zweite Implementationsvariante gewählt. Nachdem alle nötigen Bibliotheken unter Mac OS installiert waren, konnte ich meine Software auch ziemlich rasch kompilieren und linken. Leider funktionierte aber nicht alles, was unter Linux implementiert war. Das lesen der rohen Mausdaten macht mir noch Schwierigkeiten. Da würde ich in Zukunft einen neuen Ansatz wählen. Auch die Wahl des Xquartz ist hinterher nicht ganz optimal, der Transparenz Effekt konnte nicht genutzt werden.

Mit dem Resultat der Linux-Version kann schon anständig gearbeitet werden. Wird das Projekt weiter gezogen, so macht es Sinn, zuerst die anderen Systemports fertig zu entwickeln und danach weitere Features zu implementieren. Zukünftige Funktionen sind zum Beispiel, die Verschlüsselung der Daten, das Teilen der Tastatur oder eine geteilte Zwischenablage.

10 Abbildungsverzeichnis

Illustration 1: Hardwarehandling eines Betriebssystems.....	8
Illustration 2: Remotemaus-Funktion.....	10
Illustration 3: Timetable - Zeitplanung.....	12
Illustration 4: Screenshot Synergy Homepage - https://symless.com/synergy	13
Illustration 5: Screenshot Logitech Flow Homepage - https://www.logitech.com/en-us/product/options/page/flow-multi-device-control	14
Illustration 6: Einsatz Multipoint Maus - https://msdn.microsoft.com/en-us/library/windows/desktop/ff853261.aspx	
15	
Illustration 7: Einsatz Multipoint Server (Beispiel 1) - https://docs.microsoft.com/en-us/windows-server/remote/multipoint-services/multipoint-services-stations	15
Illustration 8: Einsatz Multipoint Server (Beispiel 2) - https://docs.microsoft.com/en-us/windows-server/remote/multipoint-services/multipoint-services-stations	16
Illustration 9: Multimaus Lösungsansatz.....	17
Illustration 10: Remotemaus Lösungsansatz.....	17
Illustration 11: "man in the middle"	18
Illustration 12: "man in the middle" Multi-/Remotemaus Lösungsansatz.....	18
Illustration 13: Softwarescope.....	22
Illustration 14: Klassendiagramm (grob).....	23
Illustration 15: Beispiel Use-Case: screen-change.....	23
Illustration 16: Netzwerkaufbau für Tests.....	24
Illustration 17: Codeauszug - Konfiguriere Window-Server.....	25
Illustration 18: Codeauszug - Öffne Devicefile.....	26
Illustration 19: Codeauszug - Erstelle Thread mit Membermethode & Helferfunktion.....	26
Illustration 20: Codeauszug - Verarbeite Mausevent zu JSON-Objekt.....	27
Illustration 21: Codeauszug - Öffne ein Fenster.....	28
Illustration 22: Funktion von OpenGL - https://c-student.itn.liu.se/wiki/develop:sgct:howitworks:howitworks	29
Illustration 23: Codeauszug - Zeichne das Fenster und den Cursor.....	29
Illustration 24: 2D-Koordinatensystem in OpenGL - http://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/	30
Illustration 25: Flussdiagramm - Eventhandling.....	31
Illustration 26: Codeauszug - öffne einen Socket.....	32
Illustration 27: Codeauszug - Versende einen Event.....	32
Illustration 28: Codeauszug - Empfange Daten.....	33

11 Glossar

***nix-Systemen**

Unix und unixähnliche Betriebssysteme

6

Window-Server

Tool zur graphischen Darstellung von Fenstern

8

OpenGL

Ein rasterisierungsbasierten Renderingsystemen

20

12 Literaturverzeichnis

Literatureintrag

OpenGL Community, <http://www.opengl-tutorial.org/>

Literatureintrag

Lohmann, Niels, <https://nlohmann.github.io/json/index.html>

Literatureintrag

OpenGL Community, <https://www.khronos.org/opengl/wiki/>

Literatureintrag

X.Org Group, <http://www.opengroup.org/tech/desktop/x-window-system/>

Literatureintrag

Microsoft, <https://docs.microsoft.com/en-us/windows-server/remote/multipoint-services/introducing-multipoint-services>

Literatureintrag

Logitech, <https://www.logitech.com/en-us/product/options/page/flow-multi-device-control>

Literatureintrag

Synergy, <https://symless.com/synergy>

13 Anhang

13.1 Sourcecode & Dokumentation

Der Sorucecode des Projektes kann unter <https://github.com/i3luefirech/mmBTbachr2> gefunden werden.

Die Sorucecodedokumentation des Projektes kann unter

<https://github.com/i3luefirech/mmBTbachr2/documentation/latex/refman.pdf> gefunden werden.

13.2 Journal

W1:

- Grobinformationen zum Ablauf erhalten
- Genaue Aufgabenstellung erhalten
- Beginn studieren der Aufgabenstellung

W2:

- Groben Zeitplan erstellt
- Termin mit Andreas Danuser vereinbart für das Kickoff Meeting

W3:

- Suche nach existierenden Prdukten
- Synergy (<https://symless.com/synergy>)
- Synergy-core (<https://github.com/symless/synergy-core>)
- Microsoft Multipoint

W4:

- 05.10.2018, 17.30, Kickoff-Meeting
- Synergy einigermassen verstanden
- Lässt eine Maus übers Netzwerk auf einem anderen PC verwenden
- Benutz den selben Cursor, die externe Maus ist einer zweiten lokalen gleichgesetzt
- Einige Ansätze könnten nützlich sein, versucht aber nicht ganz das selbe Problem zu Lösen, die externe Maus verwendet den selben Cursor
- Bestellung von Mac mini ausgelöst, wird benötigt für tests unter OSX
- Grobplanung versendet

W5:

- Lösungsansatz für Linux und OSX definiert
- Lösungsansatz: Direkter zugriff auf /dev/input/mice etc. evtl. kombiniert mit Xserver
- Weitere Infos zu Lösungsansatz
 - <https://unix.stackexchange.com/questions/25601/how-do-mouse-events-work-in-linux>
 - <https://www.kernel.org/doc/Documentation/input/input.txt>
 - <http://cmp.felk.cvut.cz/~pisa/linux/mousedrivers.pdf>
 - <https://unix.stackexchange.com/questions/388053/disable-mouse-pointer-but-read-the-mouse-events>
 - <https://askubuntu.com/questions/1043832/difference-between-dev-input-mouse0-and-dev-input-mice>
- Ersatzlösung (evtl. Teillösung von Lösungsansatz):
 - Xserver
 - siehe P2
 - <https://unix.stackexchange.com/questions/397339 xcb-asynchronous-pointer-grab-not-propagating-events>
 - <https://www.systutorials.com/docs/linux/man/3-XSendEvent/>
 - <https://www.x.org/wiki/guide/debugging/>
- Ersatzlösung:

- ibgpm <https://www.linuxjournal.com/article/4600>
- <https://github.com/telmich/gpm>
- Lösung Windows noch offen! Noch keine schlaue Lösung gefunden

W6:

- Lösung Windows noch offen! Noch keine schlaue Lösung gefunden
 - <https://docs.microsoft.com/en-us/windows-hardware/drivers/samples/input-driver-samples>
 - <https://github.com/Microsoft/Windows-driver-samples/blob/master/input/moufiltr/moufiltr.c>
- Scope für Projekt erstellt
- Grobe Klassendiagramme für Projekt erstellt
- Erster PoC-Code

W7:

- Mehr PoC Code
- Wegen Arbeit, leider keine weitere Zeit investiert in die Lösungssuche für Windows
- Wegen Arbeit, leider nur wenig Zeit in PM investiert
- Windows Lösung aus Scope gestrichen!

W8:

- Wähle definitiven Lösungsansatz
- Beginne mit dem richtigen Software Code für Linux

W9:

- Alle nötigen Klassen erstellt, oft noch mit Dummyfunktionen
- Beginne damit Poc-Code ins Projekt zu implementieren
- Verfeinere Lösungsansatz im Detail für Linux

W10:

- Treffen mit dem Experten Thomas Jäggi
 - Erwähnt Logitech Flow
 - Logitechflow wird in Stand der Tehcnik aufgenommen
- Mehr Poc-Code (opengl x11)
- Mehr Linux implementationscode

W11:

- Mehr Poc-Code (opengl x11)
- Der Grossteil der Funktionalität in der Linuximplementation läuft

W12:

- Der Linux Port läuft. Sowohl Multi- wie Remotemaus!
- Portiere den code auf Mac OS

W13:

- bekomme Code unter Mac OS zum Kompilieren und Linken
- Funktionen gehen nur eingeschränkt unter Mac OS
- Optimiere Linux code

W14:

- Übernehme erarbeite Informationen in Thesis
- Versuche mehr funktionalität unter Mac OS zu erhalten
- Optimiere Linux Code
- Optimiere Mac OS Code
- Erstelle Bookeintrag
- Erstelle Ausstellungsposter

W15:

- Thesis Inhalt beinahe komplett erstellt

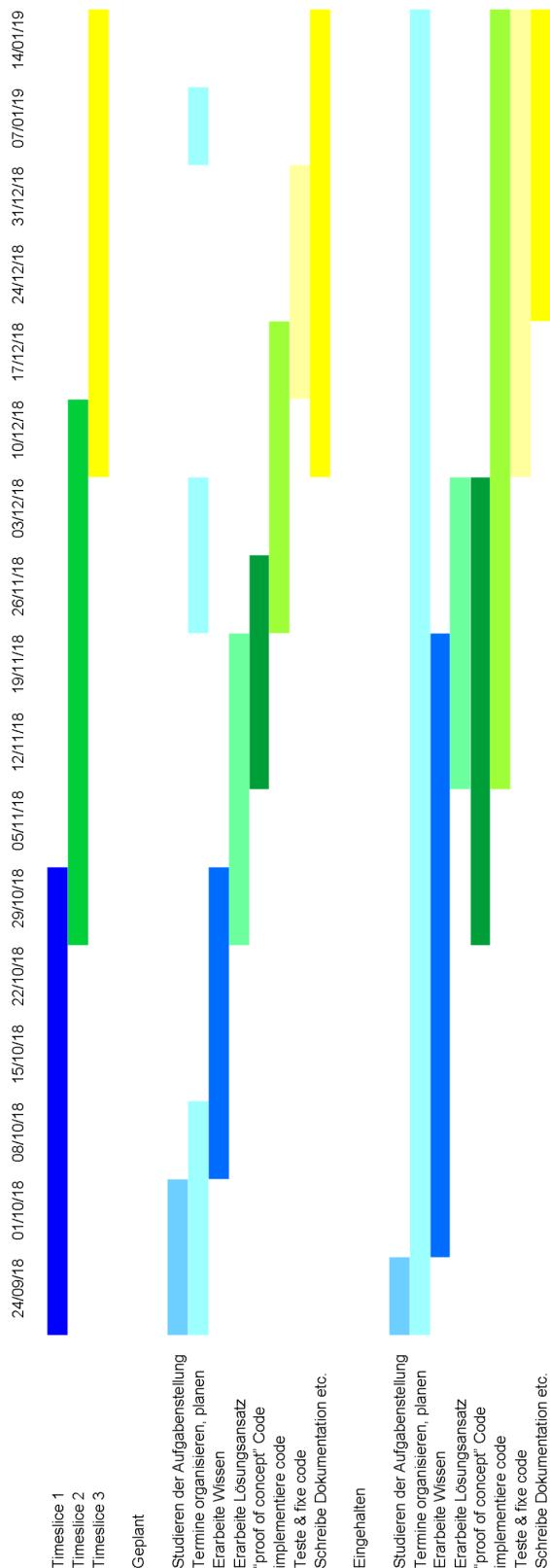
- Versuche mehr funktionalität unter Mac OS zu erhalten
- kleinere Bugfixes im allgemeinen Code

W16:

- Finale Verison der Thesis fertig gestellt
- Feinschliff der finalen Thesis Version
- Abgabereife Thesis erstellt
- Nötige Präsentationsfolien erstellt
- Präsentation Hauptprobe
- Überarbeiten der Präsentation

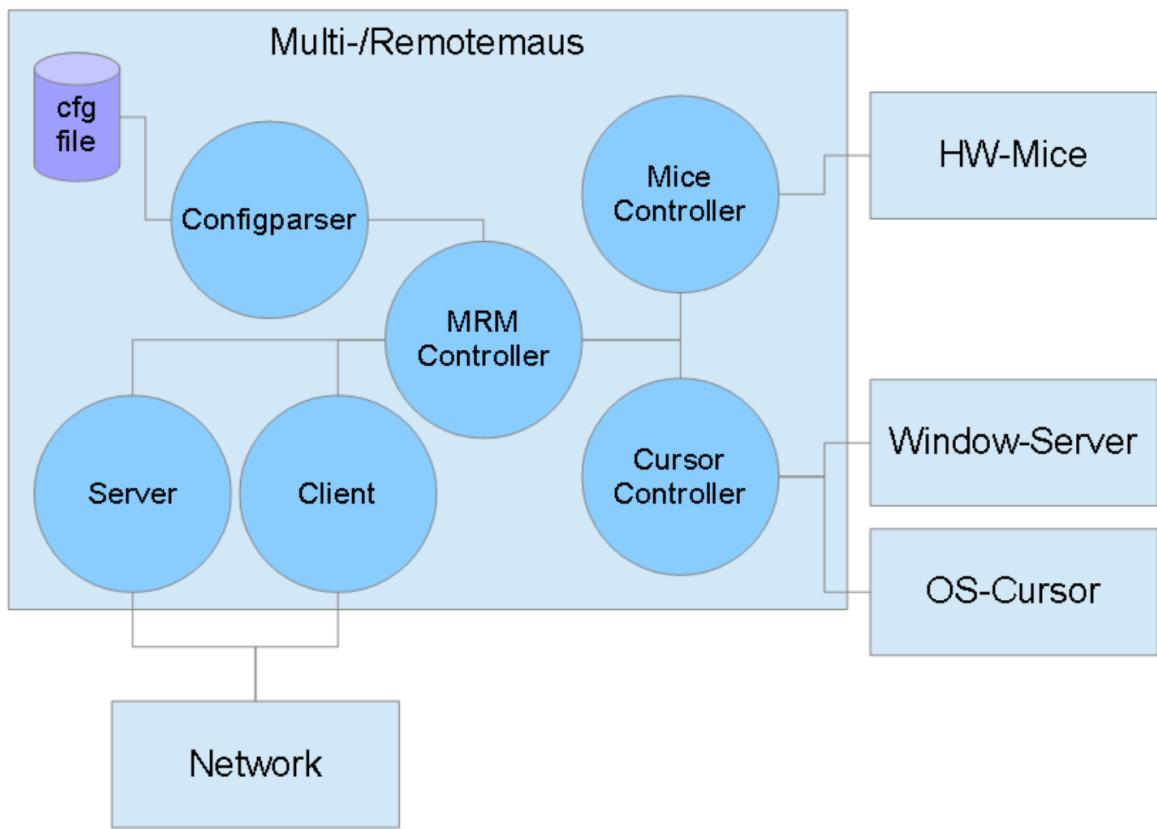
13.3 Planungsdiagramme

13.3.1 Zeitplanung

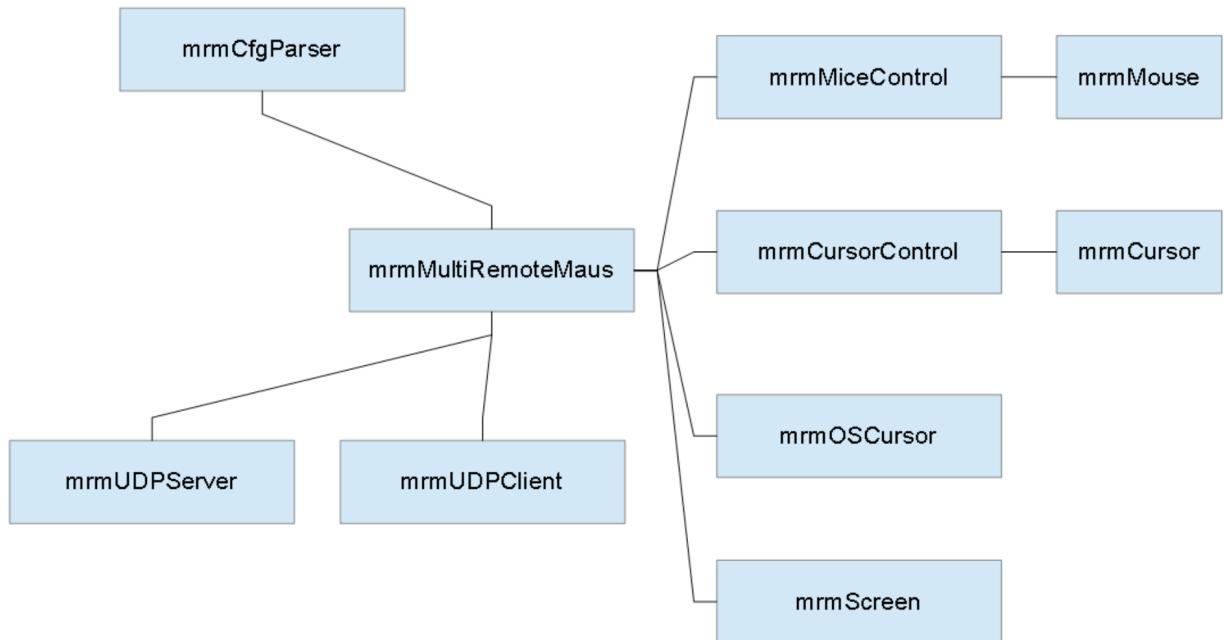


13.3.2 Softwareentwicklung

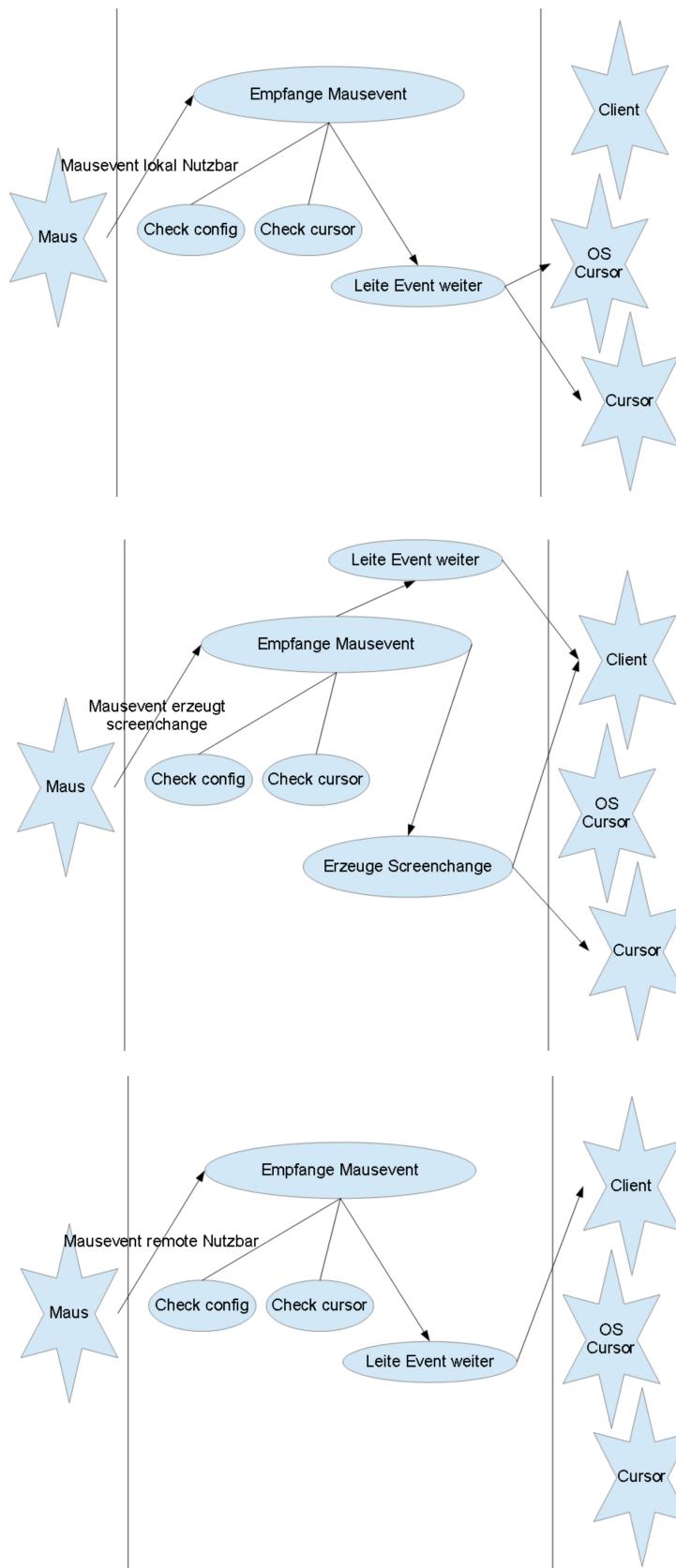
13.3.2.1 Scope



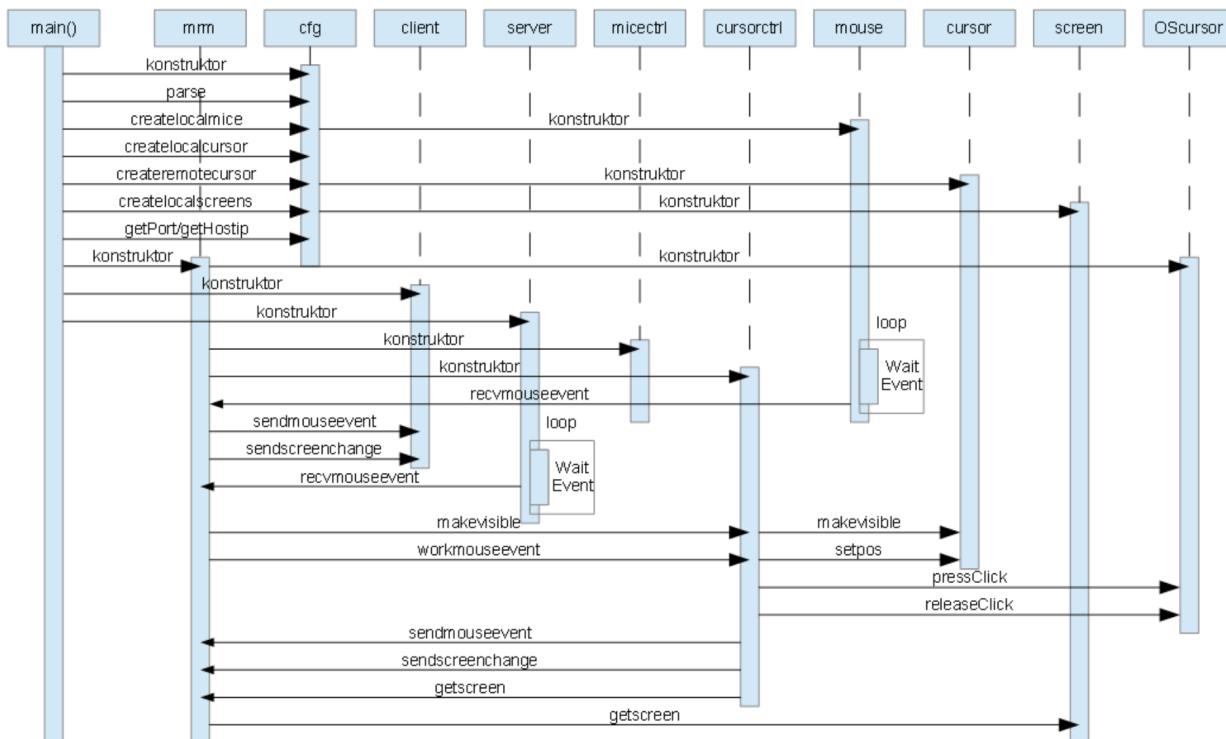
13.3.2.2 Klassendiagramme



13.3.2.3 Use-Cases



13.3.2.4 Sequenz-Diagramme



13.4 Anleitungen

13.4.1 Compilieren vom Sourcecode

Der Sourcecode ist momentan geeignet um unter Linux oder Mac OS zu kompilieren.
Um den Sourcecode kompilieren zu können wird cmake eingesetzt.

1. Laden sie das Projekt von github herunter.
2. Wechsle in das Projektverzeichnis „code“
3. Führe cmake aus
4. Wenn alle nötigen Bibliotheken installiert sind wird das Programm kompiliert
 1. Bei Fehlermeldungen bitte die fehlenden Bibliotheken als Entwicklerversion installieren
5. Durch das Kompilieren wurde das Verzeichnis „cmake-build-debug“ erstellt.
6. Das Programm ist jetzt unter dem Namen „code“ im Verzeichnis „cmake-build-debug“ abgelegt.

13.4.1.1 Konfigurieren der Multi-/Remotemaus

Zum Konfigurieren der Multi-/Remotemaus wird ein JSON-File eingesetzt.
Ein Beispieldatei befindet sich im Verzeichnis „cfg“ unter dem Namen „config.cfg“
Die Anleitung bezieht sich auf *nix-Systeme.

1. Hole die nötigen Daten der lokalen Mäuse
 1. Öffne ein Terminal
 2. Gebe diesen Befehl ein „cat /proc/bus/input/devices“
 3. Suche den gewünschten Device in der Liste
 4. Notiere den Eintrag „eventN“
 5. Gebe diesen Befehl ein „xinput –list“
 6. Notiere alle IDs des gewünschten Devices
2. Für jede lokale Maus wird ein Eintrag in „localmice“ gemacht
 1. Unter „devicefile“ wird „/dev/input/eventN“ eingetragen
 2. Unter „id“ wird eine eindeutige ID für die Maus definiert. (Achtung globale ID)

3. Für den Cursor wird ein RGB-Farbwert gespeichert
4. Unter „devicenumbers“ werden die notierten IDs von „xinput –list“ eingetragen.
 1. Das Array hat immer fünf Einträge
 2. „leere“ Einträge werden auf 0 gesetzt
3. Unter „screen“ werden die benachbarten Remotecomputer eingetragen
 1. Unter „pos“ wird eingetragen ob der Rechner links (1) oder rechts (-1) unseres Rechners steht.
 2. Unter „ip“ wird die IP des Rechners eingetragen
4. Unter „remotemice“ werden fremde Cursor erlaubt
 1. Unter „id“ wird die ID der Maus eingetragen
 2. Für den Cursor wird ein RGB-Farbenwert gespeichert
 3. Die nötigen Rechte-Flags werden gesetzt. 0 = Keine Rechte, 1 = Hat Rechte.
5. Unter „port“ wird der Kommunikationsport für das Netzwerk definiert.
6. Unter „hostip“ ist die IP des Hostrechners einzutragen.

14 Selbständigkeitserklärung

Ich bestätige, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe. Sämtliche Textstellen, die nicht von mir stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum:

Unterschrift: