FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

MTP-Lynx

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Martin Währer
Matrikelnummer 1225177

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Ing. Andreas Candido

Wien, 11. Dezember 2015

Martin Währer	Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Martin Wührer
Lazarsfeldgasse 1/6/2, 1210 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. Dezember 2015

Martin Wührer

Inhaltsverzeichnis

1. Kurzfassung

Ziel von *MTP-Lynx* ist die Entwicklung eines Touchpanels zur Steuerung von *LYNX Series / 5000*-Geräten über das *LYNX-RCP* Netzwerkprotokoll. *MTP-Lynx* besteht aus zwei Teilen: der Software *MTPSW-Lynx* und der Hardware, auf der die Software betrieben wird.

Die Hardware besteht aus einem *SBC* mit LVDS-Anschluss, einem 11" Display mit Touchscreen und einer selbstentwickelten Adapterplatine, die das Videosignal von LVDS auf parallel RGB umsetzt. Durch die geringen Abmessungen von Display, *SBC* und Adapterplatine ist es möglich, diese Komponenten in einem 19" 1HE Einschub unterzubringen.

Die Software wurde in C++14 entwickelt und soll auf der soeben vorgestellten Hardware im Vollbild betrieben werden. *MTPSW-Lynx* ist eine per XML konfigurierbare, über den Touchscreen steuerbare und auf *gtkmm*-basierte GUI.

2. Problemstellung

Im Rahmen der LVA *Bachelorarbeit für Informatik und Wirtschaftsinformatik* soll ein per Touchscreen bedienbares Steuersystem entwickelt werden. Dabei soll die Hardware für verschiedenste Steuersysteme verwendet werden können. SPS-Steuerungen mit *Proview* oder *XBMC/Kodi*-basierte Multimedia Server/Clients in Schulen oder Konferenzzentren sind nur einige wenige Anwendungsfälle. Daher auch der Name *Multipurpose Touchpanel*, für den die Abkürzung *MTP* steht.

Als Beispielimplementierung sollen jedoch im Rahmen dieser Arbeit die von der Firma *LYNX Technik AG* (im folgenden *LYNX*) entwickelten *Series / 5000*-Geräte gesteuert und überwacht werden (daher der Name *MTP-Lynx*).

Series / 5000

LYNX entwickelt und vertreibt Systemkomponenten für Audio-, Video- und Signalverarbeitung. In der Produktkategorie *Series / 5000* finden sich Rack- und Steckkartenbasierte Lösungen für Studioinstallationen jeglicher Größe. Diese Komponenten werden zentral über das netzwerkbasierte *APPolo / Control System* gesteuert und überwacht [vgl. 36].

Jedes vom *APPolo / Control System* (im folgenden *LYNX-Server*) gesteuerte *Series / 5000*-Gerät besitzt eine serverweit eindeutige Adresse und viele gerätespezifische Parameter. Diese Parameter können vom *LYNX-Server* über diese eindeutige Adresse abgefragt und ggf. geändert werden. Dazu wird das von *LYNX* entwickelte Netzwerkprotokoll *RCP* verwendet [vgl. 1].

Mit *MTP-Lynx* soll es möglich sein, diese Parameter mit GUI-Widgets zu visualisieren und über den Touchscreen zu verändern. Über eine XML-Datei soll dabei konfigurierbar sein, wie diese Parameter-Widgets am Display angezeigt werden sollen.

In Abbildung 2.1 auf der nächsten Seite ist der Systemaufbau des Gesamtsystems dargestellt. In den folgenden Abschnitten wird auf die genaueren Hard- und Softwareanforderungen näher eingegangen.

2.1. Anforderungen an die Hardware

Grundsätzlich muss die gesamte Hardware von *MTP-Lynx* in einem $19''$-Einschub mit $1HE$ Platz finden.

Durch diese geringen Abmessungen (vor allem in der Höhe) können nur Displays verwendet werden, die weniger als $40mm$ hoch sind. Um dennoch genügend Displayfläche bereitstellen zu können, sollten die Displays möglichst breit sein und gleichzeitig über einen Touchscreen verfügen.

Als Recheneinheit muss ein *ARM*-basierter *SBC*[1] im Kreditkartenformat verwendet werden (z.B. *Raspberry PI*, *Cubieboard*, *Banana PI*, usw.). Dieser soll das Display als Ausgabegerät verwenden und über Ethernet mit dem *LYNX-Server* verbunden werden.

[1] Single Board Computer

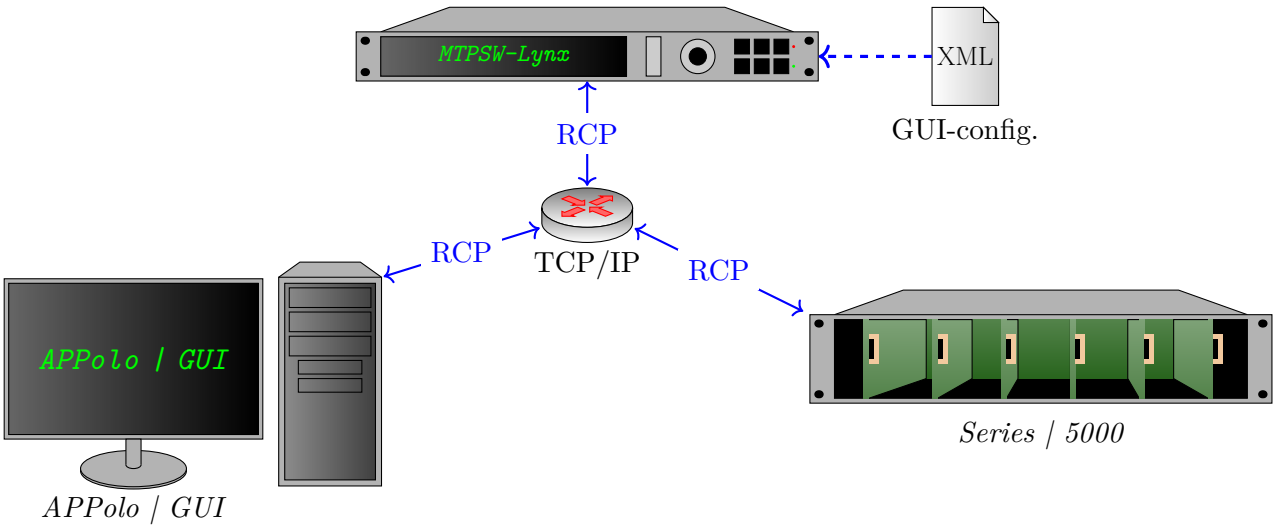


Abbildung 2.1.: Systemaufbau mit *Series | 5000*

2.2. Anforderungen an die Software

Die Software *MTPSW-Lynx* muss auf dem *SBC* mit einem auf *Debian*-basierten Betriebssystem und Paketen aus *Debian 7 (Wheezy)* oder neuer betrieben werden können (z.B. *Raspbian*[L24] am *Raspberry PI* oder *armbian*[L54] am *Banana PI/Cubieboard*). Allerdings sollte es zusätzlich möglich sein, *MTPSW-Lynx* auf *x86*- bzw. *amd64*-CPUs unter *Debian* zu betreiben.

Die Software wird im Vollbildmodus gestartet, um die gesamte Displayfläche möglichst effizient verwenden zu können. Es sind keine speziellen Anforderungen an die GUI gestellt, jedoch muss sichergestellt sein, dass durch die im Vergleich zu Desktopsystemen geringe Rechenleistung der *SBCs* die UX[2] der GUI nicht abnimmt.

Die Kommunikation mit dem *LYNX-Server* sollte über eine unverschlüsselte *TCP/IP*-Verbindung und zumindest mit *IPv4* erfolgen. Zur Kommunikation mit dem *LYNX-Server* muss das *LYNX-RCP*[1] verwendet werden.

Die Konfiguration der GUI muss über XML-Dateien erfolgen. *LYNX* hat jedoch für die *APPolo | Control*-GUI bzw. das *Apple-iPad* ein XML-Format für einen ähnlichen Zweck entwickelt. Dieses Format sollte soweit möglich beibehalten und ggf. minimal erweitert werden [vgl. 2].

[2] User Experience

3. Hardware

Die Hardware des *MTP* sollte im Endausbau über einen Touchscreen, Taster, LEDs und einem Touchregler bzw. einem Touchwheel bestehen. Alle diese Komponenten müssen an der Front eines 1HE-Einschubs untergebracht werden (siehe Abbildung 3.1).

Dadurch, dass sowohl die Software, als auch die Hardware im Rahmen der LVA *Bachelorarbeit für Informatik und Wirtschaftsinformatik* entwickelt wurden, lag die Priorität bei der Entwicklung des Geräts beim Touchscreen und bei der Software *MTPSW-Lynx*. Für die zusätzlichen Taster, LEDs und Touchsensoren war der Entwicklungszeitrahmen zu kurz. In zukünftigen Weiterentwicklungen des Geräts werden diese Komponenten aller Vorraussicht nach noch eingebaut werden.

Die größte Einschränkung der Hardware betraf das verwendete Display, denn für die Front eines 1HE Einschubs (siehe Abschnitt 2.1 auf Seite 2) war es nicht sehr einfach, ein passendes Display zu finden.

Abhängig vom Display konnte anschließend festgelegt werden, welcher *SBC* verwendet wird und ob noch eine Zusatzplatine entwickelt werden muss.

3.1. Display

Für *MTPSW-Lynx* wurde das $11.1''$ *Truly TFT1280120-1-E* verwendet. Mit einer Breite von $289mm$ und einer Höhe von $38.6mm$ passt dieses genau in das Anforderungsprofil,[55] denn die Höhe eines 1HE Einschubs beträgt $44.5mm$ und die Breite ist mit $465mm$ für das Display mehr als ausreichend.[L6] Außerdem besitzt das Display bereits einen resistiven Touchscreen.

Dieses $1280x120$-Pixel-Display kann über eine *parallel-RGB*-Schnittstelle angeschlossen werden. Dabei können 6Bit pro Farbkanal (gesamt also 18Bit Farbtiefe) angesteuert werden [vgl. 55, S. 4].

Die Versorgungsspannung des Displays beträgt $3V3$, während die Hintergrundbeleuchtung mit $9V$ bis $10V5$ betrieben werden kann [vgl. 55, S. 6].

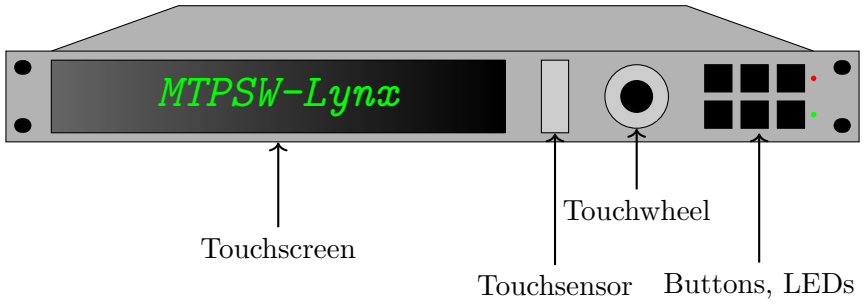


Abbildung 3.1.: Front des *MTP* im Endausbau

Alternatives Display

Im August 2014 wurde von *EarthLCD* der *Pi-Raq* vorgestellt [vgl. 16]. Dabei handelt es sich um ein zu *MTP* ähnliches Produkt, das ebenso in einem 1HE 19″-Einschub Platz finden soll. Allerdings besitzt der *Pi-Raq* im Gegensatz zum *MTP* keinen Touchscreen.

Anstelle des 11.1″-*Truly*-Displays könnte also auch das im *Pi-Raq* verbaute 10.1″-Display verwendet werden (allerdings müsste dafür noch ein kapazitiver oder induktiver Touchscreen verbaut werden). Das *EarthLCD-10.4-1024100* Display besitzt leider nur eine Auflösung von $1024x100$-Pixel, dennoch kann es entweder direkt über eine *parallel-RGB*-Schnittstelle oder über die mitgelieferte Adapterplatine betrieben werden. Die Adapterplatine wurde für einen Betrieb am *Raspberry PI* konzipiert [vgl. 24].

Ebenso wie das *TFT1280120-1-E* wird dieses über *parallel-RGB* angesteuert, allerdings nicht wie das *TFT1280120-1-E* mit 18Bit, sondern nur mit 16Bit. Beide Displaymodule besitzen exakt dieselbe Breite, allerdings ist das *EarthLCD-10.4-1024100* mit einer Höhe von $41.5mm$ etwas höher, als das vorhin beschriebene *TFT1280120-1-E* [vgl. 15, S. 4].

3.2. Recheneinheit

Wie bereits in Abschnitt 2.1 auf Seite 2 beschrieben, sollte als Recheneinheit ein *SBC* verwendet werden.

Displayschnittstelle

Sehr viele *SBCs* wie *Raspberry PI*, *Banana PI* oder *Cubieboard* besitzen eine *HDMI*- oder *miniHDMI*-Buchse.[L25,L5,L23]

HDMI wird mittlerweile von sehr vielen Geräten im Desktop-PC-Bereich und im Multimediabereich verwendet. Einige der Funktionen, wie Audiosignalübertragung über *HDMI* oder die Verwendung der *EDID*-Daten[L69], sind zwar gerade im privaten Bereich sehr hilfreich, für viele Embedded-Geräte, die über die Displayschnittstellen keine Audiosignale übertragen und fixe Displayauflösungen verwenden, jedoch viel zu komplex.

Daher bieten einige *SBCs* auch Schnittstellen an, mit denen es möglich ist, die Displaymodule direkt (ohne *HDMI*) anzusprechen.

Die meisten *Raspberry PIs* (ausgenommen *Raspberry PI Zero*) besitzen aus diesem Grund einen *DSI*-Port.[L25] Die *DSI*-Schnittstelle wurde von der *MIPI-Alliance* für Mobilgeräte entwickelt.[L39] Einige Displaymodule können daher direkt über diese *DSI*-Schnittstelle betrieben werden.

Sehr viele Displays bieten jedoch keinen *DSI*-Port, sondern werden über eine *LVDS*- oder *parallel-RGB*-Schnittstelle (wie auch die in Abschnitt 3.1 auf der vorherigen Seite vorgestellten Displays) angebunden.

Cubieboard 2

Für *MTP-Lynx* wurde daher das *Cubieboard 2* verwendet, das sowohl *LVDS* als auch *parallel-RGB* direkt unterstützt.[L23]

Außerdem sind die weiteren Komponenten des *Cubieboard 2* mit dem Dualcore *Allwinner A20* (jeder Kern Taktet mit $1GHz$) und $1GB$-DDR3 Speicher für die Software *MTPSW-Lynx* (siehe Kapitel 4 auf Seite 9) mehr als ausreichend. Auch der bereits im Chip integrierte *PWM*-Generator[1] kann sehr einfach für die Displaybeleuchtung verwendet werden.

Zusätzlich bietet das *Cubieboard 2* mit zwei *USB*-Ports, *SATA* und I^2C noch genug Erweiterungsmöglichkeiten für weitere Steuer- und Peripheriegeräte.

[1] Pulsweitenmodulation

Cubie-Baseboard

IO Technologies hat für das *Cubieboard 2* ein *Cubie-Baseboard* entwickelt, das die *LVDS*-Leitungen ebenso wie die Pins für den resistiven Touchscreen aus den *GPIO*[2]-Pins auf einen separaten Stecker führt [vgl. 54].

Das *Cubie-Baseboard* muss mit einer Spannung zwischen $8V$ und $20V$ versorgt werden und kann (nach einer kleinen Modifikation[L58]) auch das *Cubieboard 2* versorgen. Somit sind keine weiteren Versorgungsspannungen notwendig, denn sowohl das *TFT1280120-1-E*-Display, die Beschaltung am Adapterboard, als auch das *Cubieboard 2* können über das *Cubie-Baseboard* mit dem $12V$-Netzteil versorgt werden.

Kernelsupport

Sehr viele der genannten Features und Schnittstellen des *Cubieboard 2* besitzen einen großen Nachteil: Die Unterstützung der Hardware in modernen Linux-Kerneln ist relativ gering.

Beim *Cubieboard 2* kann, wie bei allen anderen *Allwinner-SoCs*[3], die Hardwarekonfiguration über *FEX*-Dateien erfolgen. Beispielsweise kann so dem *Allwinner-SoC* mitgeteilt werden, welche Auflösung das *LVDS*-Display besitzt, welche Timings dabei eingehalten werden müssen, mit welcher Frequenz die PWM für die Hintergrundbeleuchtung arbeiten soll und dass ein resistiver Touchscreen mit einer Diagonale von $11''$ verwendet werden soll [vgl. 17].

Da es sich bei diesen *FEX*-Dateien jedoch um ein proprietäres Hardwarebeschreibungsformat handelt, das nicht im Rahmen des *Mainline*-Linux-Kernels gepflegt wird, muss, wenn dieses verwendet werden soll, eine *sunxi*-unterstützte Kernelversion verwendet werden. Die aktuellste Version davon ist leider nur die Version *sunxi-3.4*, die auf dem im Mai 2012[L80] veröffentlichten Linux-Kernel 3.4 basiert [vgl. 32].

Im Linux-Kernel existiert mit dem *Device-Tree*[L57] jedoch eine Alternative zu den *FEX*-Dateien der *Allwinner-SoCs*, die auch im *Mainline*-Linux-Kernel gepflegt wird. Aktuell werden jedoch vom *Cubieboard* sehr viele Komponenten nicht unterstützt, wenn neuere Kernelversionen (ab Linux-Kernel v.3.8) mit *Device-Tree*-Unterstützung auf den *Allwinner-SoCs* betrieben werden [vgl. 33]. Daher wurde auch für *MTPSW-Lynx* der *sunxi-3.4*-Kernel verwendet. Allerdings werden mit jeder neuen Linux-Kernel Version ab 3.8 immer mehr Komponenten der *Allwinner-SoCs* unterstützt.

Alternative

Aufgrund der sehr unbefriedigenden Hardwareunterstützung für alle *Allwinner-SoCs* sind nicht nur die *Cubieboards*, sondern auch sehr viele Alternativen, die ebenfalls *Allwinner-SoCs* einsetzen, betroffen. Beispielsweise bringt zwar der *Banana PI m2* ebenfalls *LVDS*-Unterstützung mit und wäre auch bzgl. der Rechenleistung eine Alternative zum *Cubieboard 2*, jedoch verwendet dieser *SBC* mit dem *A31s* ebenfalls einen *Allwinner-SoC*[L4] und es gelten dieselben Einschränkungen bzgl. des Linux-Kernels, die auch für das *Cubieboard 2* gelten.

Daher könnte statt dem *TFT1280120-1-E* das in Abschnitt 3.1 auf Seite 4 vorgestellte *EarthLCD-10.4-1024100* am *Raspberry PI* verwendet werden, denn die *Raspberry PIs* verwenden keine *Allwinner-SoCs*.

3.3. Adapterplatine

Die Adapterplatine ist, wie in Abschnitt 3.2 auf der vorherigen Seite beschrieben, notwendig, um das Display *TFT1280120-1-E* über die *LVDS*-Schnittstelle vom *Cubie-Baseboard* betreiben zu können.

[2] general purpose input/output
[3] System-on-a-Chip

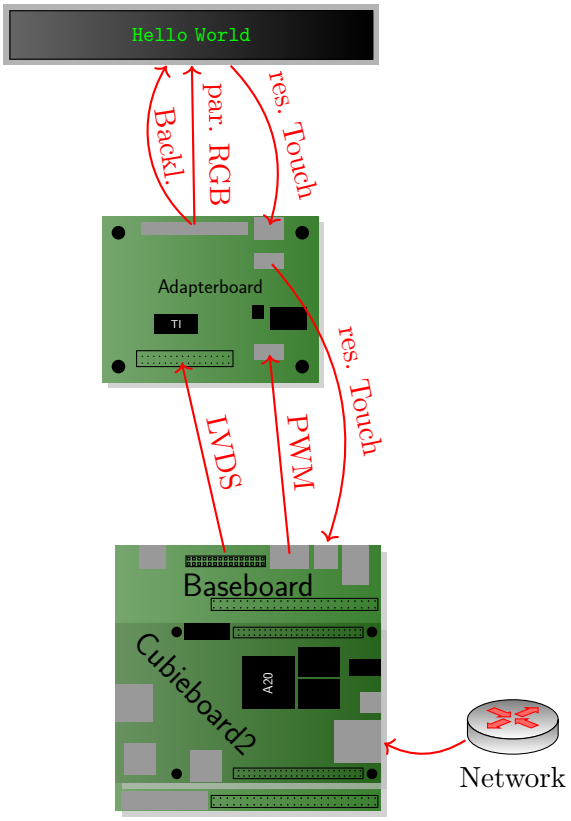


Abbildung 3.2.: Hardwarekomponenten und -schnittstellen des *MTP*

Neben der Umsetzung des *LVDS*-Signals auf *parallel-RGB* gehören zu den weiteren Aufgaben der Adapterplatine die Spannungsversorgung des Displays, die Beschaltung für die Hintergrundbeleuchtung und die Weiterleitung des Signals vom Touchscreen (siehe Abbildung 3.2).

Im Folgenden wird näher auf die einzelnen Teile der Adapterboard-Schaltung eingegangen:

3.3.1. *LVDS* zu *parallel-RGB*

Da das *Cubie-Baseboard* bereits die für *LVDS* notwendigen Pins in einer zweireihigen Stiftleiste anbietet, kann das *LVDS*-Signal vom *Cubieboard 2* sehr einfach weiterverarbeitet werden, denn diese *LVDS*-Pins können über eine Steckverbindung mit dem Adapterboard verbunden werden.

Um die *LVDS*-Signale vom *Cubie-Baseboard* anschließend in *parallel-RGB*-Signale umzuwandeln, wurde der *LVDS Konverter DS90CF86* verwendet. Dieser erlaubt es, *parallel-RGB*-Displays mit bis zu 8Bit pro Farbkanal über eine *LVDS*-Schnittstelle anzusteuern [vgl. 27, S. 1]. Dazu sind vier *LVDS*-Signalleitungspaare und ein *LVDS*-Taktleitungspaar als Eingangssignale notwendig. Ausgangsseitig werden die jeweils sechs höherwertigsten Bits pro Grundfarbe, *HSYNC*, *VSYNC* und das *DCLK*-Signal direkt mit dem Display verbunden.

Weitere Informationen zur Schaltung und ein Schaltplan sind in Abschnitt A.2.3 auf Seite 45 zu finden.

3.3.2. Spannungsversorgung

Die Versorgungsspannung des Displays (ohne Hintergrundbeleuchtung, dafür siehe Abschnitt 3.3.3) und des *DS90CF86* liegt lt. Datenblatt bei jeweils $3V3$ [vgl. 55, S. 6, bzw. 27, S. 2]. Grundsätzlich bietet das *Cubieboard 2* auch eine $3V3$-Versorgung für externe Komponenten an. Um diese Versorgung zu entlasten, wurde zusätzlich noch die $12V$ Spannung vom *Cubie-Baseboard* in eine $3V3$ umgewandelt.

Am Adapterboard kann zwischen diesen beiden $3V3$-Spannungsversorgungen (jener vom *Cubieboard 2* und jener, die aus der $12V$-Versorgung vom *Cubie-Baseboard* erzeugt wurde) mittels Jumper umgeschaltet werden.

In Abschnitt A.2.2 auf Seite 45 ist die Schaltung kurz beschrieben und aufgezeichnet.

3.3.3. Hintergrundbeleuchtung (PWM)

Zum Steuern der Hintergrundbeleuchtung (Helligkeit) wurde der im *Cubieboard 2* bereits vorhandene PWM-Generator verwendet [vgl. 4, S. 101 ff.]. Das daraus generierte *PWM*-Signal und das ebenfalls aus dem *Cubieboard 2* kommende *BL_EN*-Signal steuern zwei selbstsperrende *n-Kanal-MOSFETs*[L63], die die Versorgungsspannung der Hintergrundbeleuchtung kontrollieren.

Die Spannung der Hintergrundbeleuchtung des *TFT1280120-1-E* sollte zwischen $9V$ und $10V5$ liegen (bei $180mA$) [vgl. 55, S. 6].

Da vom *Cubie-Baseboard* jedoch nur die bereits erwähnte $12V$ Spannungsversorgung angeboten wird, muss diese Spannung noch an die des Displays angepasst werden.

Für die Schaltungsbeschreibung und -skizze, siehe Abschnitt A.2.1 auf Seite 45.

3.3.4. Touchsensorverbindung

Das Touchsensorsignal wird vom Sensor am Display über das Adapterboard an das *Cubie-Baseboard* weitergeleitet.

Das *Cubie-Baseboard* mit dem Tochsensor über das Adapterboard zu verbinden, ist nötig, da am *Cubie-Baseboard* keine Buchse für das Folienkabel des Touchsensors vorhanden ist.

Daher wurde am Adapterboard eine Buchse für das vieradrige Folienkabel und eine für die Verbindung zum *Cubie-Baseboard* montiert. Beide Buchsen sind direkt miteinander verbunden.

4. Software

4.1. Betriebssystem

In den Softwareanforderungen in Abschnitt 2.2 auf Seite 3 wurde definiert, dass zumindest ein zu *Debian 7 (Wheezy)* kompatibles Betriebssystem verwendet werden muss. Für *MTPSW-Lynx* wurde diese Anforderung jedoch auf *Debian 8 (Jessie)* als Mindestvoraussetzung für den Betrieb erhöht. Der Grund dafür ist, dass *Debian 8 (Jessie)* im April 2015[L72] und *Debian 7 (Wheezy)* bereits im Mai 2013 veröffentlicht[L73] wurde. Alleine durch die Tatsache, dass bei *Debian 7 (Wheezy)* bereits 2012 die meisten Pakete gefreezt[L73] wurden, sind die meisten Softwarepakete in *Debian 7 (Wheezy)* aktuell bereits über 3 Jahre alt, während die Pakete in *Debian 8 (Jessie)* erst im November 2014 gefreezt[L72] wurden (in beiden Fällen wird von Bugfix-Releases abgesehen).

Zusätzlich unterstützen die Compiler *g++* bzw. *clang++* in *Debian 8 (Jessie)* bereits (teilweise) den neuen C++14-Standard[L13] (siehe Abschnitt 4.2.2 auf Seite 11).

Auch das von *Debian 8 (Jessie)* verwendete *GNOME* bietet bereits experimentelle *Wayland*-Unterstützung an. Dadurch können *GTK+*-Anwendungen bereits unter *Wayland* anstelle von *X11* betrieben werden (siehe Abschnitt 4.9.1.2 auf Seite 27).

4.2. Wahl der Programmiersprache

Neben dem Betriebssystem ist die Wahl der Programmiersprache im Embedded-Bereich ebenso wie im HPC[1] essentiell. Beispielsweise werden Skriptsprachen im Normalfall zur Laufzeit mit einem Interpreter aus dem Programmcode ausgeführt. Im Gegensatz zu C-Code, der vor der Ausführung durch den Compiler in Maschinencode übersetzt wird und direkt ausgeführt werden kann, ist dadurch bei Skriptsprachen ein gewisser Overhead zu erwarten.

Während beim HPC durch effizientere Programme und Algorithmen die Anzahl der benötigten Server reduziert werden kann, hängt im Embedded-Bereich von der Softwareperformance die verwendete Hardware und die damit einhergehende Leistungsaufnahme ab.

Ein weiteres Problem von Embedded-Geräten ist, dass oftmals kein Betriebssystem am Zielsystem läuft. Dann muss die Anwendung Aufgaben wie die Speicherverwaltung selbst implementieren, die sonst das Betriebssystem übernehmen würde. Für manche Sprachen ist jedoch ein Betriebssystem Grundvoraussetzung.

Außerdem ist es oft nicht so einfach, Compiler oder Interpreter für die gewünschte Prozessorarchitektur zu finden, denn unterschiedliche Architekturen besitzen unterschiedliche Befehlssätze. Solange es für diese Architektur keinen Compiler/Interpreter für die gewünschte Sprache gibt, kann diese nicht verwendet werden [vgl. 65, S. 175].

„Deshalb hat sich C als allgemeine Sprache für Embedded Systems durchgesetzt.“ [vgl. 65, S. 175].

Ein weiterer Grund dafür ist auch, dass C (bzw. C++) ohne Betriebssystem verwendet werden kann.

[1] High Performance Computing

4.2.1. Gründe für C++

Doch unabhängig davon werden aktuell sehr viele weit verbreitete Anwendungen zur Gänze, als auch in Teilen, in C++ entwickelt.[L48]

Dazu ein Zitat eines Facebook-Mitarbeiters, das von Sutter im Rahmen der *//build/*-Konferenz 2011 erwähnt wurde: "The going word at Facebook is that reasonably written C++ code just runs fast, which underscores the enormous effort spent at optimizing PHP and Java code. Paradoxically, C++ code is more difficult to write than in other languages, but efficient code is a lot easier." [vgl. 59, S. 4]

C++ ist bzw. war, sofern dem *Tiobe Index* Glauben geschenkt werden kann, seit 2001 im Durchschnitt immer die Nummer 3 der populärsten Programmiersprachen (nach Java und C) [vgl. 8].

Alleine durch die höhere Verbreitung von C gerade im Embedded-Bereich kommt es oft vor, dass viele Bibliotheken nur in C verfügbar sind. Aber wie Stroustrup im Jahr 1999 schrieb, können in C++-Code auch C-Bibliotheken verwendet werden: "C++ was designed to be source-and-link compatible with C […]" [vgl. 56]

Diese Funktionalität ist sehr wichtig, denn dadurch kann auch bestehender C-Quellcode in den meisten Fällen durch einen C++-Compiler kompiliert werden.

"With minor exceptions, C++ is a superset of C […]." [vgl. 57, S. 1271] Das ist auch der Grund, warum oftmals von C/C++ als eine Sprache gesprochen wird.

Weitere Gründe, warum in diesem Projekt auf C++ anstelle von C gesetzt wurde, betrafen die folgenden Erweiterungen gegenüber C:

RAII [2] In C++ ist es möglich, angeforderte Ressourcen (z.B. Speicher) über Objekte zu verwalten. Diese Ressourcen können beispielsweise beim Konstruieren des Objektes angefordert werden. Sobald das Objekt zerstört wird, wird automatisch über den Destruktor die angeforderte Ressource wieder freigegeben [vgl. 42, S. 85-86].

Wenn dieses Konzept durchgehend eingehalten wird, können sehr viele Speicherzugriffsfehler und Speicherlecks vermieden werden.

C++ Standardbibliothek(*STL*) [3] In der C++ Standardbibliothek finden sich Stringklassen[L19], Containerklassen[L14], Algorithmen[L12], Funktionen zur Ein-/ und Ausgabe[L17] und vieles mehr. Diese Bibliothek ist standardisiert und wird von jedem standardkonformen Compiler mitgeliefert. Daher ist es nicht mehr notwendig, verkettete Listen, Suchbäume usw. selbst zu implementieren bzw. aus externen Bibliotheken einzubinden.

Generische Programmierung (Templates) „Generische Programmierung besitzt eine lange Tradition in C++, erlaubt sie es doch, Funktions- und Template-Klassen zu definieren, die über Typen parametrisiert werden." [vgl. 20, S. 177] Viele Funktionen der C++ Standardbibliothek basieren auf dem Template-Konzept, d.h., sie eignen sich für beliebige Typen. Dadurch ist es beispielsweise möglich, mit nur einer Listenimplementierung Listeninstanzen für unterschiedliche Datentypen zu erzeugen.

Funktionsüberladung Auch die Tatsache, dass in C++ mehrere Funktionen mit demselben Namen, jedoch unterschiedlichen Parametern erlaubt sind, ist eine wichtige Erweiterung von C++ [vgl. 51, S. 155].

[2] Resource Acquisition Is Initialization
[3] Standard Template Library

So ist es beispielsweise möglich, dieselbe Funktionalität einer Funktion/Methode unter dem selben Namen, jedoch für unterschiedliche Datentypen bereitzustellen. Dadurch kann individuell auf Typspezifika reagiert werden.

4.2.2. Gründe für C++ 14

Mit C++11 wurden zahlreiche Erweiterungen der C++ Standardbibliothek eingeführt, so wurden unter anderem reguläre Ausdrücke[L18] (siehe Abschnitt 4.7 auf Seite 20), Threads[L20] (siehe Abschnitt 4.4 auf Seite 13), eine Zeitbibliothek[L15] und einige neue Container[L14] eingeführt. Auch die Smartpointerbibliothek[L16] wurde verbessert.

Zusätzlich ist die C++11-Standardbibliothek einfacher zu benutzen und bietet bessere Performance, als das Pendant aus dem C++98 Standard [vgl. 58].

Ein weiterer großer Vorteil von C++11 ist jedoch die Spracherweiterung, so wurde unter anderem *Move-Semantik* und *Perfect-Forwarding* neu eingeführt. Die *Move-Semantik* ermöglicht es den Compilern ggf. teure Kopieroperationen durch weniger teure Verschiebeoperationen zu ersetzen und neue *Move-Only*-Typen zu erzeugen. Während *Perfect-Forwarding* es ermöglicht, Funktionstemplates mit beliebigen Argumenten zu erzeugen und an andere Funktionen weiterzuleiten [vgl. 41, S. 157].

Grundsätzlich kann gesagt werden, dass alleine durch die Verwendung eines C++11-Compilers selbst bestehender (C++98) Code schneller wird. Stroustrup schrieb dazu 2013 folgendes:

> C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster. [vgl. 57, S. v]

Im Dezember 2014 wurde mit C++14 bereits der Nachfolger von C++11 veröffentlicht.[L70]

„C++14 hat gegenüber C++11 wenige Änderungen. Es handelt sich um Korrekturen und Ergänzungen.“ [vgl. 7]

Zu einigen dieser Korrekturen gehören beispielsweise Vereinfachungen bei den mit C++11 eingeführten `constexpr`-Funktionen. So ist es in C++11 nicht möglich, in diesen mehr als eine `return`-Anweisung zu verwenden. In C++14 wurden diese Anforderungen etwas gelockert. Was spürbar zur besseren Lesbarkeit des Programmcodes beitragen kann [vgl. 41, S. 99-100].

Außerdem wurde analog zu `std::make_shared` die Template-Funktion `std::make_unique` eingeführt.

Erst ab C++14 lässt sich somit Tipp 21 von Meyers Buch *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14* auch standardkonform verwenden. “Prefer `std::make_unique` and `std::make_shared` to direct use of `new`.” [vgl. 41, S. 139]

Zusammenfassend

Alle diese Neuerungen rechtfertigen aus Sicht des Autors, über eine Verwendung von C++14 oder zumindest C++11 für Software in Embedded-Geräten nachzudenken. Die Zusatzfunktionen die modernes C++ gegenüber C bietet, machen das Programmieren einfacher und effizienter. Zumal C++11 bzw. C++14 bereits in vielen modernen Compilern unterstützt wird (C++14 wird bereits komplett von *g++* 5.0 unterstützt[L13]).

Außerdem kann durch die erweiterte (standardisierte) C++ Standardbibliothek die Abhängigkeit zu externen Bibliotheken verringert werden und durch die neuen Sprachfeatures ist es möglich, Software noch performanter zu entwickeln.

4.2.3. Build-Tools

An die Build-Tools wurden keine speziellen Anforderungen gestellt, daher fiel die Entscheidung auf das *GNU Build System*[L27]. Stattdessen ist jedoch durchaus auch der Einsatz von *CMake*[L9] denkbar.

4.3. Testgetriebene Entwicklung

Alle Teile von *MTPSW-Lynx* (mit Ausnahme der GUI) wurden testgetrieben (TDD[4]) entwickelt.

Argumente für testgetriebene Entwicklung

Bei testgetriebener Entwicklung wird der Programmcode so entwickelt, dass im Hinterkopf immer die Testbarkeit der einzelnen Programmkomponenten steht. Dadurch wird zwangsläufig das Design anders entwickelt, als ohne testgetriebener Entwicklung, es wird ein besseres Design [vgl. 29, S. 16].

Zusätzlich können die Tests als Teil der Dokumentation und Spezifikation gesehen werden. Sollte es im Zuge einer Erweiterung der Software notwendig werden, das Verhalten der Software zu verändern, beispielsweise um neue Funktionen einzubauen, kann mit den alten Testfällen einfach verifiziert werden, ob die Änderungen ein Verletzen der alten Test-Spezifikation bedeuten. Natürlich ist es in diesem Fall auch sinnvoll, die Zusatzfunktionen ebenfalls testgetrieben zu entwickeln.

Handelt es sich bei der zu entwickelnden Anwendung beispielsweise um eine Multithreadanwendung mit asynchronen Events, kann es vorkommen, dass die Bedingungen, die zu Fehlern geführt haben, nicht mehr reproduziert werden können. Das Debugging des Programmcodes wird dadurch viel schwieriger. Umso wichtiger ist es in diesen Fällen, gut getestete Programmkomponenten zu besitzen.

Funktionsweise

Die grundlegende Funktionsweise hat Langr in seinem Buch *Testgetriebene Entwicklung mit C++: Sauberer Code. Bessere Produkte.* über einen sich immer wiederholenden Zyklus (siehe Abbildung 4.1 auf der nächsten Seite) beschrieben:

1. Sie schreiben einen Test („rot“).
2. Sie sorgen dafür, dass der Test bestanden wird („grün“).
3. Sie optimieren das Design („Refactoring“).

[vgl. 29, S. 86]

Wichtig dabei ist, dass jeder Test möglichst einfach sein sollte und erst inkrementell die Komplexität des Testobjekts und der Testfälle gesteigert werden.

Zu testgetriebener Entwicklung gibt es mittlerweile sehr viele (aktuelle) Bücher. Daher wird an dieser Stelle nicht weiter darauf eingegangen.

Verwendete Bibliotheken

Um *MTPSW-Lynx* testgetrieben zu entwickeln, wurde das Unit-Testing-Framework *Boost-Test*[L64] verwendet, weil es wie die anderen *BOOST*-Bibliotheken sehr einfach in vielen aktuellen Linux-Distributionen über die Paketverwaltung installiert werden kann.

Als Mocking-Framework wurde *Turtle* verwendet, das lt. eigenen Angaben für die Verwendung mit den *BOOST*-Bibliotheken entwickelt wurde.[L7] *Turtle* ist eine header-only Bibliothek, die nicht installiert werden muss. Zum Einbinden in den Programmcode sind lediglich `#include`-Direktiven notwendig.

[4] Test Driven Development

Abbildung 4.1.: Entwicklungszyklus bei TDD.

4.4. Multithreading

In sehr vielen *SBCs* (abgesehen vom sehr populären *Raspberry PI 1*)[L49] kommen mittlerweile zwei oder mehr Prozessorkerne vor. z.B. *Raspberry PI 2*[L50] mit vier Kernen oder das *Cubieboard 2*[L1] mit zwei Kernen.

Um diese Mehrkernprozessoren besser nutzen zu können, ist es sinnvoll, eine multithreadingfähige Anwendung zu entwickeln. Dabei halfen unter anderem einige Neuerungen, die in den *C++11*-Standard eingeflossen sind. Diese hat Grimm kurz zusammengefasst:

- eine standardisierte Threading-Schnittstelle, unabhängig von Betriebssystem und Compiler,
- ein definiertes Speichermodell und atomare Datentypen,
- mehrere Techniken zum Schutz der Daten vor konkurrierendem Zugriff,
- Bedingungsvariablen, um Threads durch Events zu synchronisieren,

...

[20, S. 46]

Da die mit *C++11* eingeführten Änderungen auch unter *C++14* weiterhin unterstützt werden, konnte die Anwendung bereits von Beginn an so entwickelt werden, dass für einzelne Teilbereiche der Anwendung einzelne `std::thread`-Threads zuständig sind.

Daher ist ein Thread, nur für die GUI verantwortlich, einer nur für die unterste Ebene der Netzwerkkommunikation, einer kümmert sich um empfangene Nachrichten, usw.

4.5. Logging

Hardwarelimitierungen

SBCs haben im Vergleich zu Desktop-Computern nicht nur eine vergleichsweise geringe Rechenleistung (siehe dazu auch Kapitel C auf Seite 52), auch die Speicherkapazität des persistenten Speichers, auf dem das Betriebssystem und die installierten Anwendungen gespeichert sind, ist meist nicht mit der von modernen Desktop-Festplatten vergleichbar.

Zwar wäre es grundsätzlich möglich, eine externe Festplatte über *USB* zu betreiben, allerdings spricht dagegen eine erhöhte Leistungsaufnahme, der zusätzliche Platzbedarf und eine weitere Fehlerquelle.

Auch der Unterschied der Performance von modernen *HDDs* oder gar *SSDs* lässt sich nur schwer mit der Performance der *SBC*-Geräte vergleichen.

Das liegt einerseits daran, dass konventionelle *HDDs* mittlerweile bereits mit 180MB/s lesen, bzw. schreiben können[L35] und *SSDs* bereits das mit *SATA-3* gesetzte theoretische Limit von 6 Gbit/s überschreiten,[L67] während aktuell erhältliche Micro-SD-Karten mit *UHS-I*-Typ nicht mehr als die theoretischen 105MB/s lesen bzw. schreiben können.[L2]

Andererseits werden von aktuellen *SBCs* zurzeit selbst die schnellen *UHS-I*-Karten nicht ausgereizt, so unterstützt beispielsweise der *Raspberry PI 2* aktuell nur den *High Speed*-Bus, der ein theoretisches Limit von 25MB/s besitzt. Der in derselben Preisklasse befindliche *ODROID-C1* unterstützt zwar bereits *UHS-I*, jedoch nur mit einem Bus, der mit maximal 50MB/s betrieben werden kann.[L33]

Logging-Bibliothek

Da in *MTPSW-Lynx* sehr viele asynchrone Events, wie beispielsweise Netzwerkkommunikation oder Benutzereingaben, über mehrere Threads verteilt verarbeitet werden, wird eine Logging-Bibliothek verwendet, um das Debugging etwas zu vereinfachen und auf Fehler der Software auch im Nachhinein besser reagieren zu können.

Der Logging-Teil der Anwendung ist jedoch der einzige Bereich, der im laufenden Betrieb ständig mit dem persistenten Speicher zu tun hat. Daher sollten folgende Anforderungen erfüllt werden:

1. Filtern der Logeinträge, um die Anzahl der Logeinträge und somit die Log-Datei klein zu halten. Die Filterung sollte auf dem Log-Level basieren (z.B. *debug*, *info*, *warning*, *error*, ...).

2. Automatisiertes Löschen von alten Logeinträgen. Durch die Anforderung, *MTPSW-Lynx* durchgehend zu betreiben, sind im Normalbetrieb Logeinträge aus den vergangenen Monaten meist nicht mehr notwendig.

3. Möglichst wenig Overhead, den das Logging verursacht, um nicht zu viel Rechenleistung zu verschwenden.

Boost.Log v2

Boost.Log v2 unterstützt bereits von sich aus den Punkt 1, also ein Filtern von Logeinträgen ist jederzeit möglich. Auch Punkt 2 ist kein Problem, denn *log rotation* ist für *Boost.Log v2* ebenfalls kein Problem.[L68]

Allerdings ist dafür nicht unbedingt die Unterstützung der Log-Bibliothek notwendig, beispielsweise könnte auch das Paket `logrotate`, das üblicherweise zur Grundausstattung von *Debian* gehört, dafür verwendet werden [vgl. 5, S. 366]. In `\etc\logrotate.conf` befindet sich die Konfiguration und in dieser kann auch konfiguriert werden, wie oft die Logdateien rotiert und ob sie automatisch gepackt werden [vgl. 5, 366ff].

Das ist auch der Grund, warum in *MTPSW-Lynx* auf die von *Boost.Log v2* angebotene Bibliotheksunterstützung von *Log rotation* verzichtet wurde.

Eines der Ziele von *Boost.Log v2* ist lt. eigenen Angaben: “The library should have as little performance impact on the user’s application as possible.” [vgl. 53] Demnach wäre der 3. Punkt ebenfalls erfüllt.

Da die Bibliothek jedoch auch zum Debuggen des Programmcodes verwendet werden kann, sind manche Einträge im Produktivbetrieb nicht relevant. Daher können die Einträge, die zwar zum

Debuggen sinnvoll sind, jedoch im Produktivbetrieb nicht benötigt werden über das Makro `NDEBUG` beim Kompilieren der Anwendung deaktiviert werden. Übrig bleiben somit nur noch Logeinträge, die auch im Produktivbetrieb relevant sind.

Wird beim *g++* die dritte Optimierungsstufe (`-O3`) verwendet, haben die Debug-Logeinträge keine Auswirkung mehr auf das kompilierte Programm (siehe Abschnitt B.1 auf Seite 50).

4.6. Netzwerk

Für die Netzwerkprogrammierung wurde die Bibliothek *Boost.Asio* verwendet. Dafür gab es mehrere Gründe:

- Es ist eine Bibliothek, die neben synchronen auch asynchrone Operationen erlaubt.[5]
- Es wird das C++ Iostreams-Konzept auf Netzwerkebene unterstützt.[L44]
- Aktuelle *Boost.Asio*-Versionen setzen außerdem bereits das in C++11 neu eingeführte `std::future`-Konzept um.[L42]
- Client-Anwendungen können völlig transparent für IPv4 als auch IPv6 betriebene Server benutzt werden.[L41] Somit ist *MTPSW-Lynx* bereits für IPv6 gerüstet.
- Auch SSL-Unterstützung ist über *Boost.Asio* grundsätzlich möglich,[L45] dennoch wird es in *MTPSW-Lynx* noch nicht verwendet, da aktuell die *LYNX-Server* noch ohne SSL-Unterstützung betrieben werden.

  Um *MTPSW-Lynx* auf SSL umzurüsten, sollten allerdings nur geringe Änderungen im Programmcode notwendig sein.

Boost.Asio wird jedoch nicht nur für die Netzwerkprogrammierung verwendet, in *MTPSW-Lynx* werden auch die asynchronen `boost::asio::steady_timer` verwendet.[L46]

Neben dem Timer kann auch Signalbehandlung von UNIX-Signalen über *Boost.Asio* realisiert werden.[L43]

Design

Der Teil der Software, der für die Verbindung zum Server zuständig ist, wurde in mehrere Klassen aufgeteilt. Das hat einerseits den Grund, dass dadurch die Komplexität abstrahiert wird, andererseits eignen sich einzelne Klassen, die jeweils nur Teilbereiche der notwendigen Arbeiten erledigen, besser für Unit-Tests, was bei TDD sehr wichtig ist.

Daher wurde ein dynamisches Interfacemodell mit rein virtuellen Interfaces verwendet. Grundsätzlich hätte das Problem aber auch mit einem Template-Ansatz angegangen werden können, wie Stroustrup in [57, S. 777] beschreibt. Templates besitzen jedoch einige Nachteile, beispielsweise müssen sie im Header implementiert werden. Das führt zwangsläufig zu häufigerem Neukompilieren von Code, der diesen Header einbindet.

Außerdem wird der Code aus den Templates generiert, was oft zu mehr Programmcode führt, speziell dann, wenn es viele Spezialisierungen gibt.

Zusätzlich ist es im aktuellen C++14-Standard noch kaum möglich, bestimmte Anforderungen an die Template-Parameter zu stellen. Erst mit dem für zukünftige C++-Versionen geplanten Feature *Concepts*[L85] wird es, ähnlich den rein virtuellen Klassen, möglich sein, Typanforderungen an Templates zu stellen.

[5] *Asio* steht für “asynchronous input/output”

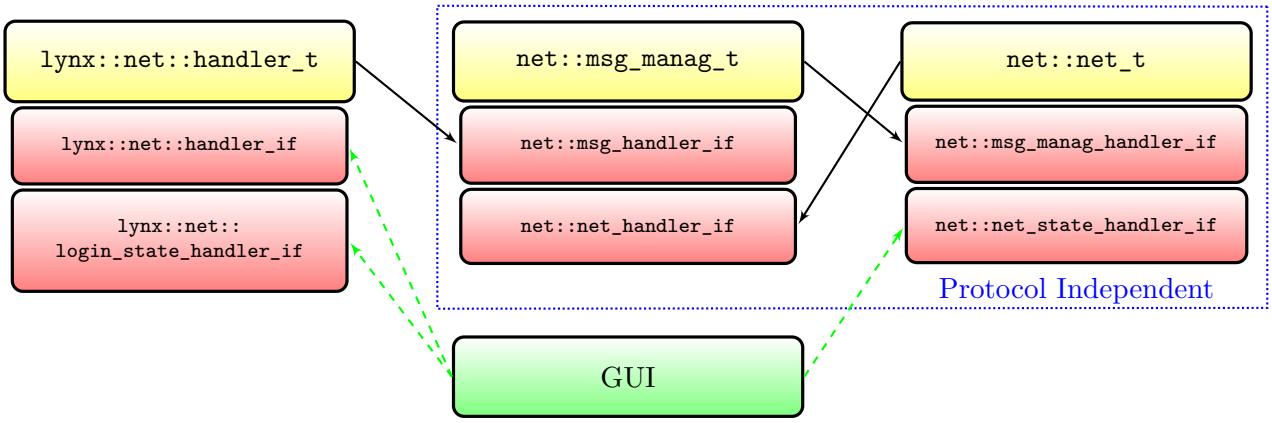


Abbildung 4.2.: Beziehung der Netzwerkklassen und -interfaces. Alle Klassen innerhalb des blauen Rahmens sind protokollunabhängig.

In Abbildung 4.2 sind die Netzwerkklassen und ihre Beziehung zueinander kurz skizziert. Im Folgenden wird auf die einzelnen Klassen näher eingegangen:

4.6.1. `net::net_t`

Diese Klasse ist von allen Netzwerklassen jene mit dem niedrigsten Abstraktionslevel. In dieser Klasse wird die Namensauflösung und der Verbindungsaufbau zum Server durchgeführt. Wenn eine aktive Verbindung von einer der beiden Seiten beendet wird, versucht `net::net_t` die Verbindung neu aufzubauen.

Auch neuerliche Verbindungsversuche nach einem bereits fehlgeschlagenen Verbindungsaufbau werden hier eingeleitet. Dadurch abstrahiert diese Klasse alle verbindungstypischen Eigenschaften und bietet den darauf aufbauenden Klassen eine verbindungslose Schnittstelle an.

Die darüber liegenden Klassen werden nur benachrichtigt, wenn die aktive Verbindung zum Server geschlossen wurde, denn in einigen Protokollen ist es nötig, nach einem neuerlichen Verbindungsaufbau erneut die Authentifizierungssequenz zu senden.

Diese Klasse ist protokollunabhängig, d.h., sie enthält keine *LYNX-RCP*-spezifischen Implementierungsdetails.

Zustände

`net::net_t` implementiert eine State-Machine mit folgenden Zuständen:

„Init“ Dieser Zustand ist der Startzustand, dieser ist aktiv, nachdem die `net::net_t`-Klasse konstruiert wurde. Ein Verbindungsaufbau zum Server ist jederzeit möglich, indem `net::net_t::connect()` aufgerufen wird.

„Connecting“ Dieser Zustand wird eingenommen, wenn die Funktion `net::net_t::connect()` aufgerufen wurde. `net::net_t::connect()` ist dabei ein blockierender Aufruf, d.h., der Thread, der diese Funktion aufgerufen hat, wird solange blockiert, bis `net::net_t::disconnect()` aufgerufen wurde und sich die `net::net_t`-Instanz wieder im Zustand „Init“ befindet.

„Reconnect“ Dieser Zustand wird erreicht, wenn entweder die Verbindung nicht hergestellt werden konnte, oder die Verbindung zum Server beendet wurde. In diesem Zustand wird versucht, erneut eine Verbindung herzustellen.

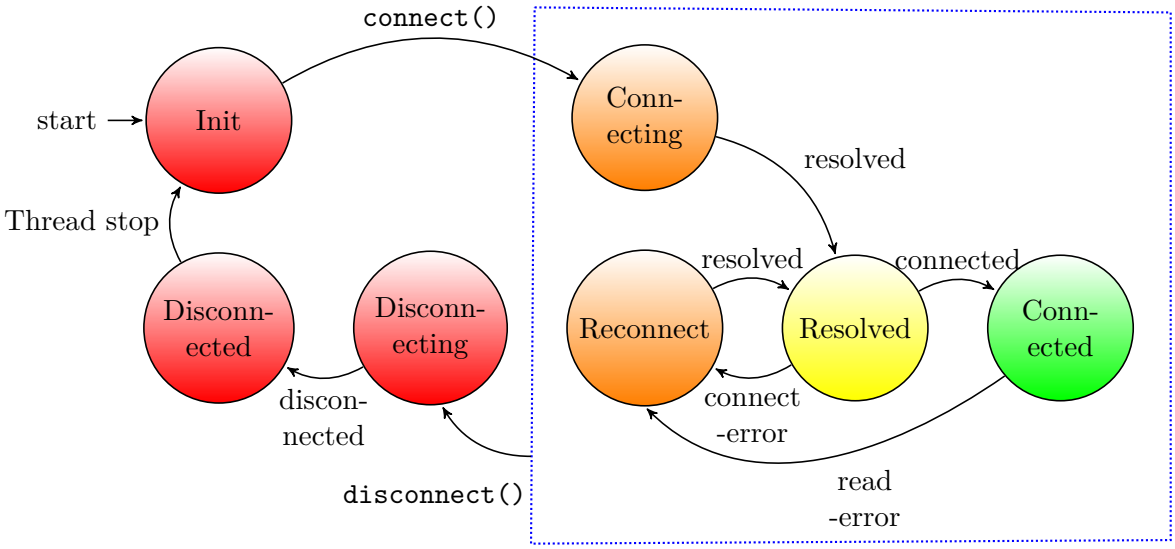


Abbildung 4.3.: Mögliche Zustandsübergänge der Zustände von `net::net_t`. Der Übergang nach „Disconnecting“ ist von allen Zuständen möglich, die innerhalb des blauen Rahmens liegen.

Über C-Makros kann bestimmt werden, wie oft die Verbindung neu aufgebaut wird, bevor eine *Exception* geworfen wird und wie viel Zeit zwischen einem fehlgeschlagenen Verbindungsaufbau und einem neuen Versuch liegen muss.

„Resolved“ Dieser Zustand wird erreicht, wenn der Verbindungsaufbau erfolgen kann.

„Connected“ In diesem Zustand ist es möglich, dem Server Nachrichten zu senden und selbige zu empfangen.

„Disconnecting“ Dieser Zustand kann nur erreicht werden, wenn die Anwendung sich in den Zuständen „Connecting“, „Reconnect“, „Resolved“ oder „Connected“ befindet und die Funktion `net::net_t::disconnect()` aufgerufen wurde.

„Disconnected“ Sobald `boost::asio` zurückmeldet, dass die Netzwerkverbindung erfolgreich beendet wurde, wird dieser Zustand erreicht. Der letzte Befehl in der `net::net_t::connect()`-Funktion ist der Zustandswechsel von „Disconnected“ auf „Init“.

In Abbildung 4.3 ist der Zustandsübergang beim Verbindungsaufbau eingezeichnet.

Interfaces

Wie in Abbildung 4.2 auf der vorherigen Seite zu sehen, implementiert `net::net_t` das Interface `net::msg_manag_handler_if`, das eine Schnittstelle definiert, die verwendet werden kann, um Nachrichten zum Server zu senden, der mit dieser Instanz verbunden ist.

`net::net_t` kann dagegen empfangene Nachrichten, Sendebestätigungen und den Verbindungszustand an die Klasse, die das Interface `net::net_handler_if` implementiert, senden.

Das `net::net_state_handler_if` wird von der GUI verwendet, um Benachrichtigungen für Zustandsänderungen zu erhalten. Dadurch kann visualisiert werden, wenn eine Verbindung zum Server hergestellt wurde. Diese Callbacks werden entkoppelt vom Hauptthread aufgerufen (der Thread der

`lynx::net_t::connect()` aufgerufen hat). Dadurch wird sichergestellt, dass diese den Hauptthread nicht beeinflussen.

4.6.2. `net::msg_manag_t`

Auch diese Klasse ist protokollunabhängig. Ihre Aufgabe ist es, ein- und ausgehende Nachrichten zu puffern und andere Klassen zu informieren, sobald die Verbindung zum Server unterbrochen wurde.

Zusätzlich bietet diese Klasse die Funktion, jede Nachricht, die ihr zum Senden übergeben wurde so oft zu versenden, bis das Senden ohne Fehler erfolgt ist. Anschließend wird die Klasse, die diese Nachricht gesendet hat, benachrichtigt, dass diese erfolgreich gesendet wurde.

Dass die Verbindung zum Server unterbrochen wurde, kann von den darüber liegenden, protokollabhängigen Klassen dafür genutzt werden, Nachrichten an den Server zu senden, die Teil der Authentifizierungssequenz sind (sofern dies vom Protokoll definiert ist).

Jede ausgehende Nachricht besitzt einen Nachrichtentyp. Abhängig von diesem Typ können die Nachrichten dadurch priorisiert werden. Zusätzlich ist es möglich, bestimmte Nachrichtentypen kurzzeitig zu sperren. Dadurch können temporär alle Nachrichten verzögert werden, die nicht zur Authentifizierungssequenz gehören.

Thread-Signale

`net::msg_manag_t` bietet ein auf Thread-Signalen basiertes Interface an (`sig::signal_t`). D.h., wenn eine Nachricht empfangen wurde, wird dies einem Thread, der zum Bearbeiten von empfangenen Nachrichten verantwortlich ist, über Condition-Variablen (`std::condition_variable`) mitgeteilt. War die Ausführung des Threads zu der Zeit unterbrochen, wird die Ausführung des Threads fortgesetzt und die Nachricht bearbeitet. Arbeitet der Thread bereits eine Nachricht ab, geht die Nachricht nicht verloren, sondern wird solange gespeichert, bis dieser Thread diese Nachricht abarbeiten kann.

Durch dieses Modell werden empfangene Nachrichten in speziellen Threads abgearbeitet. Daher ist es möglich, mehrere Threads mit der Abarbeitung der Nachrichten zu betrauen und somit bei großem Datenaufkommen diese schneller zu bearbeiten.

4.6.3. `lynx::net::handler_t`

Diese Klasse abstrahiert das im *LYNX REMOTE CONTROL PROTOCOL 1.9* (im folgenden *LYNX-RCP*) beschriebene Protokoll [vgl. 1]. D.h., alle Programmkomponenten, die diese Klasse verwenden, müssen sich nicht um die Protokolleigenheiten kümmern. Sie senden nur `lynx::net::msg_t`-Nachrichten und können davon ausgehen, dass diese Nachrichten auch am Server ankommen. Im Folgenden wird kurz auf die Aufgaben dieser Klasse eingegangen.

Authentifizierung

Eine der Aufgaben von `lynx::net::handler_t` ist die Authentifizierung am *LYNX-Server*. Die Authentifizierung ist lt. *LYNX-RCP* [vgl. 1, S. 9] das erste Kommando, das nach dem Verbindungsaufbau an den Server gesendet werden muss.

Anschließend wird an den Server eine „uptime“-Anfrage gesendet. Beide Kommandos bilden zusammen die Authentifizierungssequenz.

Wird beispielsweise die Verbindung beendet, oder der Server neu gestartet (wird erkannt durch regelmäßige „uptime“-Anfragen), werden alle Nachrichten, die aktuell in der Warteschleife auf den Versand warten so lange blockiert, bis eine neue Authentifizierungssequenz gesendet und vom Server bestätigt wurde.

Parameter-Abos („subscription“-Befehle)

Im *LYNX-RCP* ist es möglich, bestimmte Parameter am Server zu abonnieren. Diese Abos unterscheiden sich von „normalen“ Nachrichten durch den Nachrichtentyp. Ändert sich der Wert eines abonnierten Parameters am Server, wird automatisch eine Nachricht an den Abonnenten mit dem neuen Wert gesendet.

Die Abos gelten jedoch nur solange, bis die TCP-Verbindung zum *LYNX-Server* geschlossen, der Server neu gestartet, oder eine neue Authentifizierungssequenz gesendet wird.

Eine der Aufgaben von `lynx::net::handler_t` ist es daher, alle Abos zu speichern und nach dem Senden der Authentifizierungssequenz diese wieder an den Server zu senden.

Regelmäßige „uptime“-Überprüfung

In der *LYNX-RCP*-Beschreibung wird empfohlen, den „uptime“-Parameter regelmäßig vom *LYNX-Server* abzufragen und zu überprüfen, ob die erhaltenen Werte aufsteigend sind [vgl. 1, S. 12]. Dadurch kann erkannt werden, ob der *LYNX-Server* neu gestartet wurde und eine erneute Authentifizierung am *LYNX-Server* notwendig ist.

Sendebenachrichtigung

Im *LYNX-RCP* sind mehrere Nachrichtentypen definiert. So gibt es unter anderem „Read“-, „Write“- und, wie bereits erwähnt, auch „Subscription“-Befehle [vgl. 1, S. 4-8]. Beim Senden der Nachricht kann eine Callback-Funktion angegeben werden, die aufgerufen wird, wenn der *LYNX-Server* auf die Nachricht geantwortet hat. Bei „Write“-Nachrichten antwortet der Server immer mit dem aktuellen Parameterwert.

Soll beispielsweise der Wert eines Parameters geändert werden, ist erst nach Empfangen der Antwortnachricht sichergestellt, dass der Server diese erhalten hat.

Erneutes Senden der Nachrichten

Über ein C-Makro kann festgelegt werden, wie groß der Zeitraum ist, innerhalb der eine Antwort vom Server empfangen werden muss.

Wird innerhalb dieses Limits keine Antwort empfangen, wird die Nachricht automatisch neu gesendet.

Unterscheidung „normale“ Nachrichten und Subscriptions

Wie bereits erwähnt, müssen beiden Nachrichtentypen unterschiedlich behandelt werden. Dafür sind die Klassen `lynx::net::handler_t::message` bzw. `lynx::net::handler_t::subscriptions` zuständig.

Zustände

Auch `lynx::net::handler_t` implementiert eine State-Machine mit folgenden Zuständen:

„Stopped“ Dieser Zustand ist der Startzustand und ist aktiv, wenn die `lynx::net::handler_t`-Klasse konstruiert wurde. Zu diesem Zeitpunkt ist keiner der Threads mehr aktiv. Sobald `lynx::net::handler_t::run()` aufgerufen wurde, wird in den Zustand „Starting“ gewechselt.

„Stopping“ Wird aktiv, wenn die Methode `lynx::net::handler_t::stop()` aufgerufen wurde und sich die Instanz dieser Klasse in den Zuständen „Starting“, „Initialized“ oder „Authenticated“ befindet. In diesem Zustand wird versucht, alle aktiven Threads zu stoppen.

„Starting“ Dieser Zustand wird erreicht, wenn die Funktion `lynx::net::handler_t::run()` aufgerufen wurde. `lynx::net::handler_t::run()` ist dabei ein blockierender Aufruf, der solange blockiert, bis `lynx::net::handler_t::stop()` aufgerufen wurde. Außerdem werden alle benötigten Threads in diesem Zustand gestartet.

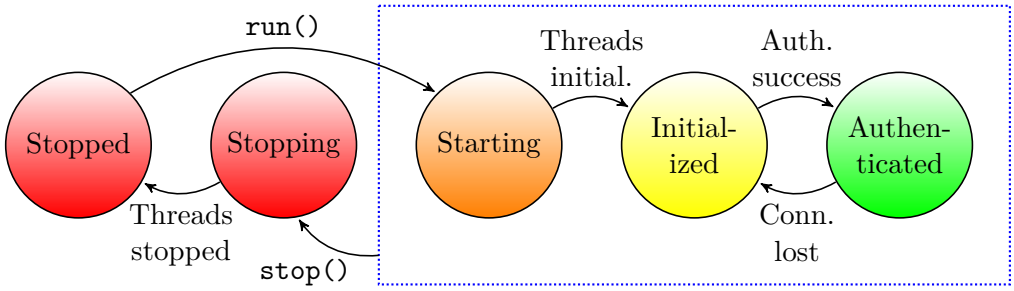


Abbildung 4.4.: Zustände von `lynx::net::net_handler_t`. Der Übergang nach „Stopping“ ist von allen Zuständen möglich, die innerhalb des blauen Rahmens liegen.

„Initialized“ Nachdem alle Threads erfolgreich gestartet wurden, ist dieser Zustand aktiv.

„Authenticated“ Wenn von `net::net_t` die Benachrichtigung kommt, dass eine TCP-Verbindung mit dem Server hergestellt werden konnte, wird die Authentifizierungssequenz an den Server gesendet. Nachdem diese vom Server bestätigt wurde, wird dieser Zustand erreicht.

In Abbildung 4.4 sind die Zustände aufgezeichnet, die `lynx::net::handler_t` annehmen kann.

Interfaces

Wie in Abbildung 4.2 auf Seite 16 zu sehen, werden die Interfaces `lynx::net::handler_if` bzw. `lynx::net::login_state_handler_if` implementiert.

Ersteres definiert eine Schnittstelle, um Nachrichten an den *LYNX-Server* zu senden, während letzteres ähnlich der Klasse `net::net_t` auch die Möglichkeit bietet, Benachrichtigungen über Zustandsänderungen zu erhalten.

4.7. Reguläre Ausdrücke zum Parsen

Zum Datenaustausch mit der Software werden (wie in Abbildung 2.1 auf Seite 3 dargestellt) 2 Schnittstellen verwendet:

- Die XML-Schnittstelle: Sie wird in Abschnitt 4.8 auf Seite 23 näher betrachtet.
- Die *LYNX-RCP*-Schnittstelle [vgl. 1]: Sie wird in diesem Abschnitt näher behandelt.

LYNX-Remote Control Protocol (RCP)

Das *LYNX-RCP* ist ein textbasiertes (ASCII) Protokoll, das dem Client-Server Modell folgt. Die Nachrichten zum und vom Server können wahlweise über UDP/IP oder TCP/IP übertragen werden. Zusätzlich ist es möglich, Nachrichten über die serielle Schnittstelle (RS232) zu senden/empfangen [vgl. 1, S. 2].

Ein *LYNX-Server* steuert dabei unterschiedliche *Series / 5000*-Geräte. Jedes dieser Geräte besitzt eine serverweit gültige Adresse, über die Eigenschaften und Parameter des Geräts gelesen und teilweise auch geschrieben werden können.

Nachrichtenformat

Alle Nachrichten sind textbasiert und in diesen können nur bestimmte ASCII-Zeichen vorkommen. Das Format der Nachrichten ist ähnlich dem CSV-Format: Es wird ein Zeichen als Feldtrenner und

eines als Nachrichtentrenner verwendet. Kommt eines der Trennzeichen im Feld-Text vor, muss es mittels Backslash codiert werden [vgl. 1, S. 2].

Manche Felder enthalten jedoch mehrere Werte, diese sind mit anderen Trennzeichen getrennt. Ob und wie viele Unterfelder in einem Feld untergebracht sind, hängt unter anderem vom Nachrichtentyp ab.

Mit regulären Ausdrücken können sehr präzise alle gültigen Zeichen angegeben und die Nachricht auf Gültigkeit überprüft werden. Außerdem ist es möglich, die Nachricht sehr einfach an den Trennzeichen zu trennen. Zusätzlich kann die Behandlung von codierten Trennzeichen in der Nachricht damit sehr einfach ausgedrückt werden.

Evaluierung der RegEx[6]-Parser

Zum Verwenden von regulären Ausdrücken unter C++ können unterschiedliche Bibliotheken verwendet werden: *Boost.Regex*[L52], *Posix Regex*[L86], *PCRE*[L34] oder `std::regex`[L18] sind nur einige davon.

Um aus den genannten Bibliotheken die passendste auszuwählen, wurden sie einem Benchmark unterzogen. Getestet wurde auf einem PC mit *Intel i7 3770k*-Prozessor, dem *Raspberry PI 2* und dem *Cubieboard 2*. Die Bibliotheken wurden in der Standardkonfiguration mit unterschiedlichen Versionen von *g++* und *clang++* getestet. Der Test (siehe Kapitel C auf Seite 52) enthält dabei den regulären Ausdruck, der zum Parsen der *LYNX-RCP*-Nachrichten vom *LYNX-Server* verwendet wird und die Operationen, um eine neue Objektinstanz mit mehreren `std::string`-Member aus den geparsten Teilen zu erzeugen.

Evaluierung am PC

Das Ergebnis für den PC unter *Ubuntu 15.10 (Wily Werewolf)* (siehe Abschnitt C.1 auf Seite 52) zeigt, dass sich unabhängig vom verwendeten Compiler *g++* bzw. *clang++* die Ausführungszeiten der externen Bibliotheken *Boost.Regex*, *PCRE* und *Posix Regex* kaum unterscheiden. Einzig der von *g++*-v.4.9 übersetzte Test ist insgesamt einen Tick langsamer, als der Test der anderen Compiler. Grundsätzlich kann gesagt werden, dass *PCRE* die performanteste der auf dem PC getesteten Bibliotheken ist. Etwa die eineinhalbfache Zeit von *PCRE* benötigt die C++-Bibliothek *Boost.Regex* und weit abgeschlagen mit der vier- bis fünffachen Zeit liegt die Implementierung von *Posix Regex*.

Ein ähnliches Bild bietet sich für den PC auch unter *Debian 8 (Jessie)*, wobei hier die Streuung der Ergebnisse zwischen den Compilern etwas größer ist, als unter *Ubuntu 15.10 (Wily Werewolf)* und die Laufzeit von *Posix Regex* etwas gedrückt werden konnte.

Im Gegensatz dazu hängt die Performance von `std::regex` jedoch sehr stark vom verwendeten Compiler bzw. von der konkreten Implementierung der Standardbibliothek ab. Unter *Ubuntu 15.10 (Wily Werewolf)* und *clang++*(v.3.5 oder höher) in Kombination mit *libstdc++* wird etwa die Performance erreicht, die auch mit *PCRE* erreicht wird. Während *clang++* mit *libc++* für den Test `std::regex` mehr als zehnmal so viel Zeit benötigt, als die *PCRE*-Bibliothek. Auch unter *Debian 8 (Jessie)* wird die Laufzeit, die mit *Boost.Regex* gemessen wurde bei weitem nicht erreicht.

Evaluierung am *Cubieboard 2*

Selbst die Tests am *Raspberry PI 2* (siehe Abschnitt C.2 auf Seite 55) bzw. am *Cubieboard 2* (siehe Abschnitt C.3 auf Seite 56) ergeben ein Bild, das zum PC-Ergebnis sehr ähnlich ist: Zuerst allerdings fällt auf, dass sich die Ergebnisse zwischen *Raspberry PI 2* und *Cubieboard 2* kaum unterscheiden, außerdem überholt *Boost.Regex* in jedem Test auf beiden *arm*-Geräten die Bibliothek *PCRE*, die sich bei jedem Test am PC an die Spitze gestellt hatte.

[6] Regular Expression - Regulärer Ausdruck

Fazit

Am Zielsystem war die Bibliothek *Boost.Regex* die schnellste Bibliothek. Allerdings ergaben die Tests auch, dass selbst die Performance der langsamsten Bibliothek *Posix Regex* mehr als ausreichend für *MTPSW-Lynx* ist, denn in der Praxis werden kaum mehr als hunderte Nachrichten innerhalb weniger Sekunden vom Server verschickt. Daher ist die Performance kein Ausschlusskriterium.

Ein Kriterium war daher, dass *PCRE* bzw. *Posix Regex* C-Bibliotheken sind, und keine C++-Interfaces anbieten. Um C-Bibliotheken in C++ sicher verwenden zu können, muss darauf geachtet werden, wann C++-*Exceptions* auftreten können. Im ungünstigsten Fall kann es sonst passieren, dass angeforderte Ressourcen nicht mehr freigegeben, oder das Programm zu undefiniertem Verhalten führt (beispielsweise wenn in der Bibliothek `longjmp`-Anweisungen verwendet werden).

> Similarly, `longjmp()` from `<csetjmp>` is a nonlocal goto that unravels the stack until it finds the result of a matching `setjmp()`. It does not invoke destructors. Its behavior is undefined if a destructor would be invoked by a throw from the same point of a program. Never use `setjmp()` in a C++ program. [vgl. 57, S. 1264]

Aus diesen Gründen ist es nicht nur aufgrund der meist einfacheren Interfaces, sondern auch aufgrund der Sicherheit oftmals einfacher, C++ Bibliotheken zu verwenden.

Ein anderes Kriterium ist die Standardkonformität. Bibliotheken, wie *Posix Regex* oder `std::regex`, sind standardisiert, während *PCRE* bzw. *Boost.Regex* nicht standardisiert sind. Idealerweise sollten die Schnittstellen in standardisierten Bibliotheken relativ stabil sein und sich nicht sehr häufig ändern. Außerdem sollte es relativ selten vorkommen, dass plötzlich die Entwicklung eingestellt wird. Daher sind nach Meinung des Autors standardisierte Bibliotheken den nicht standardisierten vorzuziehen.

`std::regex` - C++11-RegEx

Aufgrund des C++-Interfaces und der Standardisierung von `std::regex` fiel daher in *MTPSW-Lynx* die Entscheidung auf die in C++11 eingeführten regulären Ausdrücke der Standardbibliothek. Gerade die Ergebnisse mit dem sehr aktuellen *Ubuntu 15.10 (Wily Werewolf)* zeigen, dass aktuellere Implementierungen der Standardbibliothek meist sehr viel bessere Performance liefern, als die bereits etwas betagten Implementierungen in *Debian 8 (Jessie)*.

Sollte es also mit den Standardcompilern in *Debian 8 (Jessie)* zu Performanceproblemen kommen, könnte ein neuerer Compiler samt modernerer Standardbibliothek durchaus ein Schlüssel für bessere Performance sein.

4.7.1. `lynx::net::msg_t`

Alle Nachrichten, die in *MTPSW-Lynx* empfangen oder gesendet wurden, sind vom Typ `lynx::net::msg_t`. Diese Klasse repräsentiert eine RCP-Nachricht als C++ Datentyp. Daher wurde der RegEx-Parser in den Konstruktor der Klasse eingebaut. Wird eine neue Instanz von `lynx::net::msg_t` aus einer ASCII-Zeichenkette erzeugt, wird automatisch der Inhalt geparst und die Nachricht kann im Folgenden einfach weiterverarbeitet werden (z.B. von der GUI).

Umgekehrt können alle Instanzen dieser Klasse mit der `to_string()`-Methode wieder in eine ASCII-Zeichenkette umgewandelt und an den *LYNX-Server* gesendet werden.

Gleichzeitig ist es möglich, mittels Getter-Methoden auf die einzelnen geparsten Felder der Nachricht zuzugreifen und andererseits mit Setter-Methoden die Werte von bestimmten Feldern zu manipulieren.

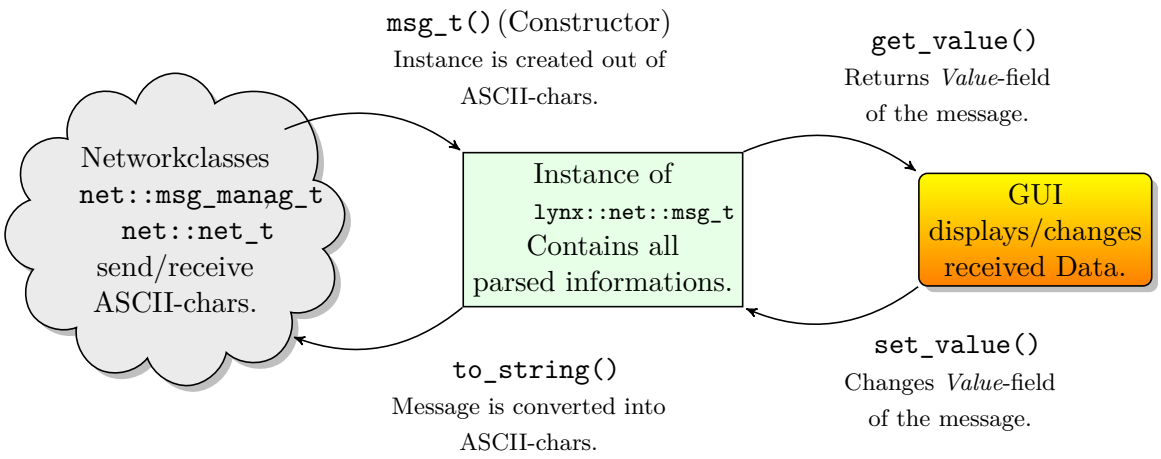


Abbildung 4.5.: Anwendungsbeispiel von `lynx::net::msg_t`.
Dieses Beispiel wäre möglich im folgenden Szenario: Der Parameterwert einer Nachricht vom Server wird in der GUI angezeigt. Später wird der Wert in der GUI über den Touchscreen verändert und schließlich wird diese veränderte Nachricht wieder an den Server gesendet.

In Abbildung 4.5 ist skizziert, wie die GUI, `lynx::net::msg_t` und die Netzwerkklassen zusammenarbeiten, wie das *Value*-Feld in der Nachricht gelesen, verändert und wie die veränderte Nachricht wieder an den *LYNX-Server* gesendet werden kann.

Fehlerbehandlung

Im *LYNX-RCP* ist ebenfalls definiert, wie Fehlermeldungen vom *LYNX-Server* an die Clients gesendet werden: Fehler werden durch ein zusätzliches letztes Feld in der Nachricht an den Empfänger identifiziert [vgl. 1, S. 9]. Beispiele für Fehler sind Syntaxfehler, Fehler bei der Authentifizierung, usw. Wird beim Parsen der Nachricht dieses zusätzliche Feld erkannt, sollte dieses umgehend behandelt werden.

Die Klasse `lynx::net::msg_t` kann zwar Fehlermeldungen parsen, jedoch werden diese anschließend nicht in der Instanz der Nachricht gespeichert, stattdessen wird beim Parsen der ASCII-Zeichenkette eine *Exception* abhängig vom empfangenen Fehler geworfen. Dadurch kann sehr einfach auf unterschiedliche Fehler reagiert werden.

4.8. XML-Parser

Wie in Abschnitt 2.2 auf Seite 3 beschrieben, wurde für die *APPolo* / *Control*-GUI bzw. das *Apple-iPad* von *LYNX* ein spezielles XML-Format entwickelt, das es erlaubt, bestimmte Steuerelemente (im folgenden Widgets) auf einer Arbeitsfläche (basierend auf einem Koordinatensystem) zu positionieren. Diesen Widgets können bestimmte *LYNX-Server*-Parameter zuordnet werden. D.h., diese Widgets können über den Touchscreen gesteuert werden und die Parameterwerte am *LYNX-Server* verändern [vgl. 2].

Durch dieses XML ist es daher möglich, eine virtuelle Steuerung der am *LYNX-Server* angeschlossenen *Series* / *5000*-Geräte zu bauen. Diese Steuerung kann anschließend sowohl im *Apple-iPad*, als auch in der *APPolo* / *Control*-GUI verwendet werden.

Modifiziertes XML-Format

MTPSW-Lynx verwendet als Grundlage ebenfalls dieses Format. Allerdings unterscheiden sich die Anforderungen von *MTPSW-Lynx* gerade in Bezug auf die Displaygröße stark von denen in der *APPolo | Control*-GUI bzw. dem *Apple-iPad*. Daher waren einige Modifikationen notwendig. So verwendet das Format von *LYNX* wie bereits erwähnt, ein Koordinatensystem, auf dem die Widgets beliebig platziert werden können. Auch Überlappungen von Widgets sind dabei möglich. Dieses Format wurde für *MTPSW-Lynx* so modifiziert, dass nun ein Gitterlayout verwendet wird.

Dadurch ergibt sich eine Vereinfachung in der Bedienung: Um im Originalformat beispielsweise die Widgets eines Containers anzuzeigen, musste ggf. in den Container gezoomt werden und je nach Zoomstufe wurden die Widgets im Container unterschiedlich groß angezeigt oder gar ausgeblendet. Im neuen Format gibt es keine Zoomstufen mehr, stattdessen werden alle Widgets im selben Container gleichzeitig angezeigt, wenn der übergeordnete Container selektiert wurde.

Dadurch wird einerseits verhindert, dass Elemente außerhalb des aktuellen Anzeigebereichs liegen und somit der Anzeigebereich über den Touchscreen verschoben werden muss, um mit allen Elementen interagieren zu können, andererseits ist die Möglichkeit, den Anzeigebereich zu zoomen, gar nicht mehr notwendig. Dieses Gitterlayout eignet sich somit ideal für Geräte, die einen kleinen Bildschirm besitzen.

Für genauere Informationen, welche Änderungen konkret gemacht wurden, siehe [66].

Bibliothek

Zur Evaluierung der XML-Bibliothek wurden keine Performancetests durchgeführt, denn im Gegensatz zu den Nachrichten vom Server, die zur Laufzeit immer wieder empfangen und geparst werden müssen, muss die XML-Datei nur beim Starten der Anwendung geparst werden. Daher spielt die Laufzeit der XML-Parser-Komponente nur eine untergeordnete Rolle.

Da dem Autor keine standardisierte XML-Bibliothek für C oder C++ bekannt war, fiel die Wahl auf *libxml++*, das einen C++-Wrapper für die unter anderem für das *GNOME*-Projekt entwickelte *libxml2* darstellt [vgl. 47, bzw. 64].

libxml++ bietet unter anderem einen *DOM*- und *SAX*-Parser an [vgl. 47]. Die Entscheidung für den *SAX*- und gegen den *DOM*-Parser fiel, weil das XML-Dokument sehr viele Tags besitzt, die in der Anwendung nicht benötigt werden. Beim *DOM*-Parser würden auch diese Tags zusätzlichen Speicher belegen. Beim *SAX*-Parser kann jedoch beim Parsen für jedes XML-Tag entschieden werden, ob es gespeichert oder verworfen werden soll. Außerdem können beim *SAX*-Parser die Daten beim Auslesen bereits sehr einfach in eigene Strukturen gepackt werden, während beim *DOM*-Parser alle Daten in dynamischen Strukturen gespeichert werden, die anschließend vom Programm abgefragt werden.

State-Machine + Element-Stack

Wie in [66] erklärt wird, besitzt die XML-Datei unter anderem mindestens ein `<scene>`-Tag und beliebig viele (auch verschachtelte) `<element>`-Tags.

Jedes Element, das mit einem `<element>`-Tag beschrieben wird, wird durch mindestens ein Widget in der GUI repräsentiert. Einer *Scene* (beschrieben durch `<scene>`-Tag) können beliebig viele Elemente (`<element>`-Tags) zugeordnet werden. Dabei sind die bereits erwähnten Container eine Spezialform der Elemente und somit ebenfalls durch ein `<element>`-Tag beschreibbar.

Zum Parsen der XML-Datei wurde eine State-Machine in Kombination mit einem Element-Stack verwendet: Jedes Tag, das von *MTPSW-Lynx* ausgewertet werden kann, stellt einen Zustand in der XML-State-Machine dar. Aufgrund der vielen möglichen Tags (und somit möglichen Zustände) wurde an dieser Stelle auf ein Zustandsdiagramm verzichtet.

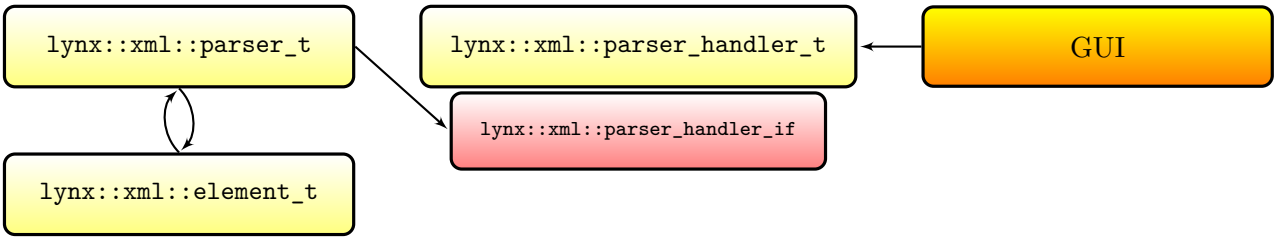


Abbildung 4.6.: Beziehung der XML-Parser-Klassen

Allerdings ist es, wie bereits erwähnt, in der XML-Datei möglich, innerhalb von Containern wiederum beliebig viele `<element>`-Tags (auch Container) zu positionieren. Die Struktur der XML-Datei kann somit nicht mehr in einer einfachen State-Machine abgebildet werden. Daher wurde ein Element-Stack eingeführt, der die einzelnen Rekursionsstufen abbildet.

Wird beispielsweise ein `<element>`-Tag geparst, wird auf dem Stack ein neues Element angelegt. Innerhalb dieses `<element>`-Tags werden alle Tags geparst und bei Erreichen des `<element>`-End-Tags wird der Eintrag vom Stack genommen. Befindet sich jedoch innerhalb des vorigen `<element>`-Tags ein `<children>`-Tag mit einem weiteren `<element>`-Tag, so wird für dieses wieder ein neuer Eintrag am Stack angelegt. Erst wenn das innere `<element>`-End-Tag geparst wurde, wird der letzte Eintrag vom Stack genommen. Kommen anschließend das `<children>`- und das äußere `<element>`-End-Tag, wird auch der erste Eintrag vom Stack genommen.

Layout

Zum Parsen und Aufbereiten der XML-Datei sind drei Klassen verantwortlich: `lynx::xml::parser_t`, `lynx::xml::element_t` und `lynx::xml::parser_handler_t`. In Abbildung 4.6 sind diese und ihre Beziehung zueinander kurz skizziert.

Diese Klassen spielen nur beim Programmstart eine Rolle, denn das XML wird nur beim Start geparst. Ein weiteres Parsen von XML-Dateien im laufenden Betrieb ist nicht möglich und nicht vorgesehen, stattdessen kann die Anwendung einfach neu gestartet werden.

Im Folgenden wird auf die drei XML-Klassen näher eingegangen.

4.8.1. `lynx::xml::parser_t`

Diese Klasse ist die bereits erwähnte SAX-Parser-Klasse, die mittels Vererbung von der *libxml++*-Basisklasse `xmlpp::SaxParser` erbt.

`lynx::xml::parser_t` ist dafür verantwortlich, die XML-Datei zu parsen und zu validieren.

Im Gegensatz zu den GUI- und Netzwerkklassen werden die XML-Parser-Klassen nur beim Programmstart benötigt. Daher wird zum Parsen auch kein eigener Thread verwendet, stattdessen wird das Parsen in dem Thread-Kontext ausgeführt, der es angestoßen hat. Der Aufruf blockiert somit solange, bis das XML geparst wurde, oder ein Fehler aufgetreten ist.

Wurde ein `<scene>`-Tag vollständig geparst, wird an `lynx::xml::parser_handler_t` das geparste Scene-Objekt übergeben.

Die Validierung der XML-Datei wird nicht mittels XSD durchgeführt, sondern während des Parsens durch den SAX-Parser übernommen. Dadurch ist kein zweiter Parser-Durchlauf mit dem XSD notwendig.

Da allerdings einige Tags, die von *LYNX* definiert wurden, in *MTPSW-Lynx* nicht benötigt und somit vom Parser ignoriert werden, kann es vorkommen, dass diese ignorierten Tags fehlerhaft sind,

obwohl die XML-Datei korrekt geparst werden konnte. Für die Funktionalität und die Stabilität von *MTPSW-Lynx* sind diese fehlerhaften Tags jedoch unerheblich, daher werden diese Fehler ignoriert.

4.8.2. `lynx::xml::element_t`

In der XML-Datei können unterschiedliche `<element>`-Tags angegeben werden. Zu jedem dieser `<element>`-Tags muss ein Typ angegeben werden. Abhängig von diesem Typ können dabei unterschiedliche Eigenschaften und Parameter spezifiziert werden. Beispielsweise ein oberes und unteres Limit beim *Slider*-Widget. [7] Jedoch werden nicht alle Eigenschaften und Parameter in allen Widgets benötigt. So macht beispielsweise ein oberes und ein unteres Limit beim Ein-/Aus-Button keinen Sinn.

Während ein Element geparst wird, werden alle geparsten Eigenschaften in einer Instanz von `lynx::xml::element_t` gespeichert. D.h., eine Instanz von `lynx::xml::element_t` ist ein universelles Element-Objekt und kann alle möglichen Parameter (unabhängig vom Typ) aufnehmen.

Sobald die `lynx::xml::parser_t`-Instanz jedoch ein `<element>`-End-Tag findet, wird versucht, aus den geparsten Informationen der `lynx::xml::element_t`-Instanz ein in der GUI visualisierbares Objekt zu erzeugen. Dieser Schritt funktioniert jedoch nur, wenn alle für diesen Elementtyp notwendigen Tags und Parameter in der XML-Datei angegeben wurden. Ist das nicht der Fall, wird eine *Exception* geworfen und das Programm beendet sich, denn die XML-Datei ist nicht vollständig.

4.8.3. `lynx::xml::parser_handler_t`

In der XML-Datei ist es möglich, mehrere `<scene>`-Tags anzugeben. Allerdings wird von *MTPSW-Lynx* nur das erste `<scene>`-Tag verwendet. Diese Klasse ist dafür verantwortlich, dass alle anderen `<scene>`-Tags ignoriert werden.

Die Klasse `lynx::xml::parser_handler_t`, die das Interface `lynx::xml::parser_handler_if` implementiert, wird als Abstraktionsschicht zwischen Parser und GUI eingeführt. Sobald der Parser ein Scene-Tag vollständig geparst hat, erhält die vom Parser verwendete Instanz von `lynx::xml::parser_handler_t` das Scene-Objekt.

Die GUI-Klassen können anschließend das Scene-Objekt von der `lynx::xml::parser_handler_t`-Instanz abfragen (siehe Abbildung 4.6 auf der vorherigen Seite).

4.9. GUI

In embedded Umgebungen und für *SBCs* werden andere Anforderungen an die GUI-Bibliotheken gestellt, als in reinen Desktop-Umgebungen. Beispielsweise werden oftmals Touchscreens anstelle von Maus und Tastatur eingesetzt. Auch bei den Displayauflösungen und der Farbwiedergabe sind die Displays in embedded und *SBC*-Umgebungen nicht mit denen in aktuellen Desktop-Systemen oder Notebooks vergleichbar. Wie in Abschnitt 3.1 auf Seite 4 beschrieben, bietet beispielsweise das für *MTPSW-Lynx* verwendete Display *TFT1280120-1-E* von *Truly* nur eine Auflösung von 1280x120 bei 18Bit Farbtiefe.[55]

[7] Das *Slider*-Widget wird verwendet, um Fließkomma-Parameter zu visualisieren.

4.9.1. Verfügbare Grafik-APIs

Um unter Linux eine grafische Oberfläche für Programme entwickeln zu können, gibt es unterschiedliche Möglichkeiten. Im Folgenden wird auf einige Grafikschnittstellen, die unter Linux eine große Rolle spielen, näher eingegangen und die Vor- bzw. Nachteile davon erörtert.

4.9.1.1. *X11*-Server

Der Mitte der 80er entwickelte und noch immer weit verbreitete *X-Server* erlaubte die Entwicklung von portablen GUIs [vgl. 37, S. 312-313]. Bereits im September 1987 wurde nach *X10* (genauer *X10R4*) das bis heute verwendete *X11* herausgegeben [vgl. 6, S. 75-76]. *X11* (aktuell in der Version *X11R7*[L26]) spielt in modernen Linux-Distributionen eine große Rolle als Fenstersystem. So wird *X11* beispielsweise als Grundlage für die Desktops *GNOME* oder *KDE* verwendet [vgl. 6, S. 77]. Daher ist *X11* auch in aktuellen Versionen der Distributionen *Debian*, *Ubuntu* oder *Fedora*, um nur einige zu nennen, enthalten und kann somit auch auf *SBCs* verwendet werden.

Ein *X11*-Server ist in einem Linux-System jedoch nicht nur für die Anzeige am Display verantwortlich, sondern kümmert sich auch um die Eingabegeräte wie Touchscreens, Maus oder Tastatur. Das vereinfacht einerseits den Betrieb von Displays und Eingabegeräten, da diese meist *plug & play* funktionieren[L81], andererseits werden mittlerweile sehr viele Funktionen des *X-Servers* nicht mehr benötigt, da diese Funktionen entweder vom Kernel, oder von den GUI-Bibliotheken und Desktops übernommen werden [vgl. 31].

Aufgrund der Funktionsfülle ist der *X-Server* mittlerweile sehr schwer zu warten. Beispielsweise wurde 2013 ein Fehler im Programmcode entdeckt, der bereits seit 20 Jahren im Quellcode des *X-Servers* enthalten war.[L31]

„Der Ansatz von X11 gilt schon lange als unzeitgemäß, da Computer und ihre Grafikchips heute ganz andere Aufgaben übernehmen, als bei der Entstehung von X vor dreißig Jahren. Nur durch umfangreiche und permanente Um- und Ausbauten ist der X-Server überhaupt für moderne Einsatzzwecke geeignet.“ [vgl. 31]

Außerdem ist für den Betrieb der Anwendungen neben dem *X-Server* auch ein *Windowmanager* notwendig, der für die Positionierung und teilweise für das Aussehen des Fensters verantwortlich ist (z.B. Titelleiste) [vgl. 60]. Populäre *Windowmanager* sind beispielsweise *KWin* (Standard bei *KDE*)[L84], *mutter* (Standard bei *GNOME*)[L40], *Compiz* (Standard bei *Ubuntu*)[L11] und viele mehr.

Es gibt jedoch unter Linux neben dem *X11*-Server noch andere Grafikschnittstellen (die teilweise sogar performanter sind).

4.9.1.2. *Wayland*

Ein Beispiel dafür ist das im Vergleich zum *X11*-Server noch sehr junge *Wayland*-Protokoll, das von vielen Distributoren und Desktops mittlerweile als Nachfolger von *X11* gehandelt wird [vgl. 31].

„Die Bildausgabe mit Wayland ist um einiges schlanker und sicherer, weil Wayland die Anwendungen isoliert. Der Wayland-Ansatz vermeidet zudem viel Overhead, der mit X11 einhergeht. Dadurch verspricht Wayland, die Performance zu verbessern.“ [vgl. 31]

Vereinfacht gesagt, werden mit *Wayland* der *Displayserver* und der *Windowmanager* zur neuen Komponente *Wayland-Compositor* zusammengefasst.[L28] Die Anwendungen sprechen somit nicht mehr direkt mit dem *Displayserver*, sondern mit dem *Wayland-Compositor*. Dadurch ist es allerdings auch notwendig, dass sich der *Wayland-Compositor* um die Eingabegeräte kümmert, denn dafür war lange Zeit der *Displayserver* zuständig.

> With the Wayland protocol the matter of handling input device processing is left up to the compositors themselves. To ease the development process and ensuring compositors have good input support, a common input device library has been proposed that compositors can utilize for handling their input events from the Linux kernel. [vgl. 30]

Daher wurde von Jonas Ådahl die *libinput*-Bibliothek entwickelt, die aus dem *Wayland* Referenz-Compositor *Weston* hervorging [vgl. 44]. Die aktuellen *Wayland-Compositors* von *GNOME*[L8] bzw. *KDE*[L30] verwenden diese bereits.

Interessanterweise gibt es mit *xf86-input-libinput* mittlerweile einen Wrapper, der es sogar erlaubt, *libinput* im *X-Server* als generischen Treiber für Eingabegeräte zu verwenden [vgl. 43].

Aktuell wird *Wayland* bereits seit längerem im mobilen Betriebssystemen SailfishOS eingesetzt.[L51] Auch das Linux-basierte Betriebssystem Tizen verwendet seit der Version 3.0 ebenfalls statt dem betagten *X-Server* das neue *Wayland*-Protokoll. Dabei soll sowohl die Startzeit der Anwendungen, als auch die Ressourcenanforderung um jeweils 30% verringert worden sein [vgl. 21]. Gerade bei Mobiltelefonen oder Smartwatches, in denen Tizen unter anderem eingesetzt wird, sind das durchaus wichtige Verbesserungen.

Auch das in vielen Distributionen als Standarddesktop verwendete *GNOME*[L78] bietet bereits seit Version 3.10 vom 25. Sept. 2013 experimentelle Unterstützung für *Wayland* [vgl. 45]. Daher lassen sich bereits sehr viele *GNOME*-Anwendungen unter *Wayland* betreiben.[L60] *Fedora* ist mittlerweile sogar dazu übergegangen, im Entwicklungszweig für *Fedora 24* ganz auf *GNOME* über *Wayland* anstelle von *X11* umzusteigen [vgl. L61].

4.9.1.3. *Linux-Framebuffer*

"The Linux kernel provides an abstraction for the graphical hardware in the form of framebuffer devices. These allow applications to access the graphics hardware through a well-defined API." [vgl. 38] Unter Linux gibt es mit *Linux-Framebuffer* (im Folgenden *Framebuffer*) eine zusätzliche Grafikschnittstelle.

Dabei benutzen die Anwendungen, die den *Framebuffer* verwenden, eine API, die vom Kernel bereitgestellt wird. Zusätzliche Userspace-Programme, wie *X11* oder der *Wayland-Compositor*, sind dafür nicht mehr erforderlich, stattdessen wird die gewünschte Anzeige direkt in das *FB-Device* geschrieben.

Dadurch, dass für den *Framebuffer* die Grafiktreiber sehr einfach zu schreiben sind, sind die Möglichkeiten mit dem *Framebuffer* auch sehr limitiert [vgl. 39, S. 31].

"In short, framebuffer drivers are especially interesting for embedded systems, where memory footprint is essential, or when the intended applications do not require advanced graphics acceleration." [vgl. 39, S. 31]

DirectFB

Auch für Anwendungen, die den *Framebuffer* verwenden, gilt, dass sie für die Eingabegeräte des Systems selbst verantwortlich sind. *DirectFB*, das direkt auf dem *Framebuffer* aufsetzt, bietet allerdings neben der Möglichkeit der hardwarebeschleunigten Grafikausgabe auch eine Unterstützung für Eingabegeräte.[L83]

Tatsächlich ist es so, dass *DirectFB* oftmals in Spielen oder Embedded-Systemen verwendet wird, da es sich dabei um eine schlanke Alternative zu *X.org* handelt [vgl. L79].

4.9.1.4. *OpenGL*

OpenGL[8] ist die Spezifikation einer Grafikbibliothek, die sowohl Hardwarebeschleunigt von den Chips unterstützt wird, aber auch rein in Software implementiert werden kann [vgl. 39, S. 51]. Die Spezifikation ist dabei plattform- und sprachenunabhängig. Eine Opensource-Implementierung von *OpenGL* unter Linux ist das unter der MIT-Lizenz stehende *Mesa 3D*, das (passende Treiber vorausgesetzt) allerdings auch Hardwarebeschleunigung für moderne GPUs bietet.[L53]

OpenGL ES

Gerade im Bereich der Smartphones, Tablets oder *SBCs* ist jedoch *OpenGL ES*[9] wesentlich bedeutsamer [vgl. 13, S. xxi]. So unterstützt beispielsweise der im *Raspberry PI*[L65] bzw. *Raspberry PI 2*[L66] verwendete Grafikchip *Videocore IV* die Schnittstelle *OpenGL ES* in der Version 2.0.

Auch das *Cubieboard 2* mit dem Grafikchip *Mali400MP2* unterstützt *OpenGL ES* in der Version 2.0.[L21]

OpenGL ES bzw. *OpenGL* werden meist für 3D-Anwendungen verwendet, die auf Hardwarebeschleunigung angewiesen ist. Im Gegensatz zu *OpenGL* liegt bei *OpenGL ES* jedoch der Fokus darauf, mit maximaler Funktionalität auf die veränderten Anforderungen der Hardware und der Leistungsaufnahme von mobilen- und embedded Geräten zu reagieren [vgl. 13, S. xxi].

Außerdem handelt es sich bei *OpenGL ES* nicht um eine völlig neue Spezifikation, denn die neueste Version von *OpenGL ES* ist syntaktisch sehr ähnlich mit der neuesten Version von *OpenGL* [vgl. 22, S. xxii].

Auch wenn viele Anwendungen *OpenGL* vor allem wegen der 3D-Fähigkeiten nutzen, eignet sich *OpenGL* auch für GUIs, allerdings sind die Anwendungen für das Handling der Eingabegeräte (Touchscreens, Maus oder Tastatur) selbst verantwortlich. Daher wurden die Bibliotheken *GLUT* bzw. *FreeGLUT* entwickelt. Während erstere bereits seit 1998 nicht mehr weiterentwickelt wird, wurde von *FreeGLUT* erst im März 2015 die Version 3 veröffentlicht [vgl. 18].

EGL, GLX

Allerdings können die Vorteile von *OpenGL* oder *OpenGL ES* nicht nur verwendet werden, wenn diese direkt verwendet werden. Auch *Wayland* bzw. die *Wayland* Clients können auf *OpenGL* bzw. *OpenGL ES* über die Schnittstelle *EGL* zurückgreifen. *EGL* verbindet dabei das Fenstersystem mit den *OpenGL*-Schnittstellen, ähnlich der *GLX*-Schnittstelle, die den *X-Server* mit den *OpenGL*-Schnittstellen verbindet.

Vulkan

Im März 2015 wurde im Rahmen der GDC[10] von der *Khronos Group* die API *Vulkan* als "next generation Graphics API" vorgestellt. Noch im Jahr 2015 soll die Spezifikation von *Vulkan* als *OpenGL* Ersatz veröffentlicht werden [vgl. L29, S. 4].

Allerdings soll diese neue API inkompatibel zu *OpenGL* sein und somit eine vollständige Neuentwicklung als saubere und moderne Architektur erlauben. Weitere Ziele sollten erweitertes Multithreading, reduzierter CPU-Overhead und architekturunabhängigkeit sein [vgl. L29, S. 7].

Werden alle diese Ziele erreicht, ist *Vulkan* nach Meinung des Autors nicht nur im Gaming-Bereich, sondern auch für GUIs in *SBCs* in Zukunft sicherlich eine Alternative zu *OpenGL* bzw. *OpenGL ES*.

[8] Open Graphics Library
[9] Open Graphics Library for Embedded Systems
[10] Game Developers Conference

4.9.2. GUI-Bibliotheken

4.9.2.1. *GTK+*

"GTK+, or the GIMP Toolkit, is a multi-platform toolkit for creating graphical user interfaces." [vgl. 62] *GTK+* ist eine am Linux-Desktop sehr verbreitete Bibliothek, so sind im aktuellen *Debian 8 (Jessie)* über 650 Pakete abhängig vom Paket `libgtk2.0-0`[11] und mehr als 220 Pakete abhängig von `libgtk-3-0`[12]. Werden sowohl die *GTK+2*- als auch *GTK+3*-Abhängigkeiten addiert, verwenden fast 900 Pakete aus *Debian 8 (Jessie)* ein *GTK+*-Paket.

Auf *GDK*[13], das unter anderem als Wrapper für die low-level Funktionen des *X windows Systems* (*Xlib*) diente, wurde *GTK+* gebaut [vgl. 63]. Allerdings lässt sich *GTK+* nicht nur unter Unix am *X-Server* betreiben, auch unter Windows oder MacOS können *GTK+*-basierte Anwendungen gestartet werden, die *GTK+* verwenden.[L77]

Wie bereits unter Abschnitt 4.9.1.2 auf Seite 27 beschrieben, unterstützen neuere *GNOME*-Versionen mittlerweile neben dem *X-Server* auch *Wayland* als Backend. Dies ist nur möglich, weil die *Wayland*-Unterstützung in *GTK+3* bereits sehr weit fortgeschritten ist.[L59,L60] Außerdem ist es meist nicht notwendig, die Anwendungen speziell auf *Wayland* zu portieren:

"If your application is not including any backend-specific header such as gdkx.h or gdkwin32.h, it will most likely just work under Wayland without any porting." [vgl. 48]

Dadurch ist es sogar möglich, mit dem selben Binary die Anwendung sowohl auf *X11* als auch auf *Wayland* zu betreiben, ohne den Programmcode neu zu kompilieren.

GTK+ bietet zusätzlich den Betrieb der Anwendungen als HTML5-Anwendung über das *Broadway*-Backend an. D.h., grundsätzlich kann jede *GTK+*-Anwendung ebenfalls über einen modernen HTML5-Browser bedient werden [vgl. 49]. Auch dazu sei erwähnt, dass es dafür ebenfalls nicht notwendig ist, die Anwendung neu zu kompilieren.

GTK+ ist eine Bibliothek, die in der Programmiersprache C geschrieben wurde. Wie in Abschnitt 4.2.1 auf Seite 10 beschrieben, könnte *GTK+*, da es in C entwickelt wurde, in C++ Projekten ebenfalls verwendet werden. Allerdings existieren für *GTK+* sehr viele Anbindungen an andere Programmiersprachen (z.B. *PyGObject* für *Python*, *GTK#* für *C#*, *gtkmm* für C++ uvm.) [vgl. 61].

gtkmm

Im Folgenden sind einige der Vorteile von *gtkmm* gegenüber *GTK+* hervorgehoben:

- Kapselung, Vererbung und Polymorphismus: Objekte in *gtkmm* können mit der C++ Vererbung erweitert werden. Dadurch können sehr schnell eigene Widgets aus bestehenden erzeugt werden, die dank Polymorphismus mit bestehenden Widgets interagieren können. Auch eine Kapselung von Eigenschaften und Methoden wird dadurch ermöglicht.
- Typsicherheit: Typfehler in *gtkmm*-Code können schon während des Kompilierungsvorgangs gefunden werden, während diese bei Verwendung von *GTK+* in C erst zur Laufzeit auftreten würden.
- Speicherverwaltung: Dank RAII werden in *gtkmm* durch den Konstruktor die Widgets dynamisch erzeugt und durch ihren Destruktor wieder gelöscht.
- Namespaces: *gtkmm* unterstützt die in C++ verfügbaren Namespaces direkt. Im Gegensatz zu *GTK+* sind die Funktionsnamen in *gtkmm* wesentlich kürzer.

[11] `apt-cache rdepends libgtk2.0-0 | sort | uniq -u | wc -l`
[12] `apt-cache rdepends libgtk-3-0 | sort | uniq -u | wc -l`
[13] GIMP Drawing Kit

- Keine Makros: Für *gtkmm* sind keine Makros notwendig.

[vgl. 46]

Somit ist ersichtlich, dass *gtkmm* die Sprachfeatures von C++ wesentlich besser abdeckt, als es mit der C-API in *GTK+* möglich ist. Aus Sicht des Autors sollte somit immer, wenn C++-Anwendungen entwickelt werden, über den Einsatz von *gtkmm* anstelle von *GTK+* nachgedacht werden, denn durch *gtkmm* kann kompakter und sicherer programmiert werden, als es mit *GTK+* möglich ist.

Da *gtkmm* nur eine Sprachanbindung von *GTK+* für C++ ist, gelten zudem alle bereits genannten Eigenschaften von *GTK+* auch für Anwendungen, die in *gtkmm* entwickelt wurden.

Selbst die Lizenz dürfte sich in den wenigsten Fällen als ein Problem darstellen, denn sowohl *gtkmm*[L32] als auch *GTK+*[62] wurden in der *LGPL*[14] v.2.1[L37] veröffentlicht.

4.9.2.2. *QT*

"Qt is a cross-platform application development framework for desktop, embedded and mobile. Supported Platforms include Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS and others." [vgl. 35] Auch *QT* wird in sehr vielen GUI-Anwendungen am Linux-Desktop verwendet. So sind im aktuellen *Debian 8 (Jessie)* 620 Pakete abhängig vom Paket `libqtgui4`[15] und mehr als 390 Pakete abhängig vom Paket `libqt5gui5`[16]. Werden sowohl die *QT4*-, als auch die *QT5*-Abhängigkeiten addiert, verwenden über 700 Pakete aus *Debian 8 (Jessie) QT*.

Ebenso, wie *GTK+*-Anwendungen lassen sich *QT*-Anwendungen mittlerweile unter Linux ebenfalls mit *Wayland* betreiben. Auch für *QT*-Anwendungen ist es dabei meist nicht notwendig, die Anwendungen neu zu kompilieren [vgl. 19]. Dadurch kann die Anwendung sowohl unter *X11*, als auch mit *Wayland* als Backend betrieben werden.

Allerdings kann *QT* gerade in embedded Anwendungen wesentlich vielfältiger eingesetzt werden, als *GTK+*, denn während *GTK+* zum Betrieb ein Windowsystem benötigt (also *X11* oder *Wayland*) kann *QT* auch ohne diesem verwendet werden. Das kann beispielsweise über *EGL*, den Linux-*Framebuffer* oder über *KMS*[17] in Kombination mit *DRM*[18] erfolgen [vgl. 10].

Neben *GTK+* bietet auch *QT* Sprachanbindungen für andere Sprachen an, z.B. für Python, C# oder Java [vgl. 34].

Das Ziel hinter der Entwicklung des *GNOME*-Desktops war die Entwicklung von benutzerfreundlichen Anwendungen und Desktoptools, die ähnlich denen in *KDE*, bzw. *CDE* sein sollten, jedoch vollständig auf *Freier Software* basieren [vgl. 23]. Das Problem war, dass *QT* damals nicht unter einer freien Lizenz zur Verfügung stand [vgl. 23].

Das hat sich jedoch mittlerweile geändert, *QT* wurde zwar im Laufe der Jahre unter den unterschiedlichsten (teilweise proprietären) Lizenzen veröffentlicht, aktuell werden aber sowohl proprietäre- als auch Community-Lizenzen angeboten [vgl. 11, 28]. Unter den Community-Lizenzen sind die *LGPL* v.2.1[L37], *LGPL* v.3[L38] und die *GPL* v.3[L36].

QT und die Kompatibilität zu C++

Obwohl sehr oft von *QT* als C++-Framework die Rede ist, ist es für den Autor wichtig, zu erwähnen, dass Programmcode, der in *QT* entwickelt wurde im Normalfall kein standardisierter C++ Code ist. *QT* erweitert die ohnehin schon sehr mächtige Sprache C++ abermals um nicht standardisierte

[14] GNU Lesser General Public Licence
[15] `apt-cache rdepends libqtgui4 | sort | uniq -u | wc -l`
[16] `apt-cache rdepends libqt5gui5 | sort | uniq -u | wc -l`
[17] Kernel Modesetting
[18] Direct Rendering Manager

Konzepte, wie *Signals & Slots* [vgl. 12]. Daher wurde ein eigener Precompiler (MOC[19]) entwickelt, der es erlaubt, C++ mit eben diesen neuen Konzepten zu verwenden.

Sehr viele Klassennamen in *QT*-5 beginnen mit `Q` (z.B. `QWidget`, `QLabel`, `QString` uvm.) und liegen im globalen Namespace [vgl. 9]. Der Autor ist allerdings der Meinung, dass diese Klassen idealerweise in einem eigenen Namespace für *QT* definiert und dafür das `Q`-Präfix entfernt werden soll.

4.9.2.3. *wxWidgets*

"wxWidgets is a C++ library that lets developers create applications for Windows, Mac OS X, Linux and other platforms with a single code base." [vgl. 69]

wxWidgets wird ebenfalls relativ häufig verwendet, wenn auch nicht so häufig, wie *QT* oder *GTK+*. In *Debian 8 (Jessie)* finden sich aktuell knapp 100 Pakete, die von `libwxgtk3.0-0`[20] abhängen.

Je nach Plattform werden bei *wxWidgets* unterschiedliche Komponenten genutzt. *wxGTK* ist beispielsweise der für Linux empfohlene Port [vgl. 68]. Im Unterschied zu *QT* oder *GTK+*, die direkt mit den Fenstermanagern (*X11* oder *Wayland*) kommunizieren, abstrahiert *wxGTK* dabei *GTK+*-2 und verwendet die Fenstermanager indirekt über *GTK+*-2.

Es existiert zwar mit *wxX11* auch ein Port, der direkt *X11* verwendet und *wxMotif*, der über die Bibliothek *Motif* indirekt *X11* verwendet, aber unter Linux ist es, wie bereits erwähnt, empfohlen, *wxGTK* zu verwenden [vgl. 68].

Da jedoch *GTK+*-2 verwendet wird, die *Wayland*-Unterstützung allerdings erst in den neueren *GTK+*-3-Versionen eingeführt wurde (siehe Abschnitt 4.9.1.2 auf Seite 27), wird *Wayland* mit *wxGTK* nicht unterstützt. Es gibt auch aktuell keinen offiziellen *wxWidgets*-Port für *Wayland*.

Mit *wxDFB* existiert zwar auch ein *wxWidgets*-Port für *DirectFB*, allerdings ist dieser nicht vollständig [vgl. `wxWidgets-3.0.2/docs/dfb/install.txt` in 70].

„Die wxWidgets-Lizenz ist eine leicht modifizierte LGPL und erlaubt daher die freie Verwendung in kommerzieller und freier Software und den weiteren Vertrieb unter einer selbst gewählten Lizenz." [vgl. 25]

Auch *wxWidgets* bietet Anbindungen für andere Sprachen an, z.B. für Perl, Python oder Haskell.[L87] Jedoch werden einige Anbindungen seit Jahren nicht mehr weiterentwickelt (z.B. die Anbindung für Java).[L56]

4.9.2.4. *CEGUI*

CEGUI steht für *Crazy Eddi's GUI* und ist eine Bibliothek, die sich ebenfalls unter Windows, Linux und MacOS X betreiben lässt.[L74]

Wobei die Bibliothek unter anderem *OpenGL* oder *Direct3D* (nur Windows) zum Rendern verwendet.[L75]

CEGUI wird von einigen Spielen verwendet. z.B. *Secret Maryo Chronicles*, bietet aber auch die typischen Widgets wie Buttons, Sliders, Checkboxen, usw. an, die zum Entwickeln von GUIs benötigt werden.

Allerdings wird *CEGUI* bei weitem nicht so häufig verwendet, wie *GTK+*, *QT* oder *wxWidgets*. Unter *Debian 8 (Jessie)* hängen sogar weniger als 10 Pakete von `libcegui-mk2-0.7.6` ab.[21]

Dennoch wurde *CEGUI* hier erwähnt, denn gegenüber den bisher vorgestellten GUI-Bibliotheken bietet sie den Vorteil, dass sie unter der *MIT*-Lizenz steht.[L76] D.h., es kann uneingeschränkt sowohl

[19] Meta Object Compiler
[20] `apt-cache rdepends libwxgtk3.0-0 | sort | uniq -u | wc -l`
[21] `apt-cache rdepends libcegui-mk2-0.7.6 | sort | uniq -u | wc -l`

privat als auch kommerziell, sowohl in Binär- als auch in Quellcodeform benutzt oder vertrieben werden.

Unter den vorgestellten Bibliotheken bietet *CEGUI* die wenigsten Sprachanbindungen, mit *pyCEGUI* wird aktuell nur Python unterstützt.[L82]

4.9.2.5. Weitere GUI Bibliotheken

Mit *GLT* (letzte stabile Version von 2002)[L71], *GLOW* (letzte stabile Version von 2000)[L3] und *GLUI* (letzte stabile Version von 2007)[L62] gibt es einige weitere Bibliotheken, die direkt auf *OpenGL* aufsetzten, allerdings bereits seit Jahren nicht mehr aktualisiert wurden.

Natürlich gibt es noch viele weitere C++-GUI-Bibliotheken, die unter Linux verwendet werden können. Sie alle zu beschreiben, würde jedoch den Rahmen dieser Arbeit sprengen.

4.9.3. *Windowmanager*

Wird von der GUI-Bibliothek *X11* oder *Wayland* als Backend verwendet, muss, wie bereits erwähnt, ein *X11-Windowmanager* bzw. *Wayland-Compositor* verwendet werden. Diese Komponente fällt weg, wenn als Backend *OpenGL* bzw. der *Framebuffer* verwendet wird. Desktops wie *KDE* oder *GNOME* verwenden implizit einen *Windowmanager* (z.B. *KWin*[L84], bzw. *mutter*[L40]).

Viele Desktop-Oberflächen verwenden jedoch mittlerweile sehr viele visuelle Effekte und Animationen, die zwar die UX verbessern, aber auch sehr hohe Anforderungen an die Hardware stellen. Teilweise wird für die Animationen der Fenster sogar 3D-Grafikbeschleunigung vorausgesetzt.[L10] Zusätzlich werden von vielen Desktops im Hintergrund oft weitere Anwendungen gestartet. Auf vielen *SBCs*, wie dem *Raspberry PI 1*[L65], der teilweise nur mit 256MB Arbeitsspeicher ausgestattet ist, sind daher vollwertige Desktops wie *KDE* oder *GNOME* nach Meinung des Autors nur mit viel Geduld zu bedienen.

Daher werden gerade für diese *SBCs* schlanke Desktops, wie das *Lightweight X11 Desktop Environment* kurz *LXDE*, eingesetzt [vgl. 40, S. 45]. Wie der Name schon sagt, ist *LXDE* ein Desktop auf Basis des *X-Window-Systems X11* und erfordert zum Betrieb einen lauffähigen *X11*-Server.

Desktops eignen sich sehr gut, wenn Maus und Tastatur zur Verfügung stehen und unterschiedliche Anwendungen benutzt werden. Für den Betrieb von *MTPSW-Lynx* ist es allerdings nicht erforderlich, mehrere Anwendungen zu verwenden, da die Software so entwickelt wurde, dass sie nur mit dem eingebauten Touchscreen bedient wird. Daher bietet es sich an, diese Anwendung im Vollbildmodus zu betreiben.

Statt einem vollwertigen Desktop kann dafür beispielsweise auch der schlanke *Windowmanager Matchbox* verwendet werden:

> Matchbox is an Open Source base environment for the X Window System running on non-desktop embedded platforms such as handhelds, set-top boxes, kiosks and anything else for which screen space, input mechanisms or system resources are limited. [vgl. 50]

Gerade für Vollbildanwendungen eignet sich nach Meinung des Autors *Matchbox* hervorragend.

4.9.4. Fazit

In Tabelle 4.1 auf der nächsten Seite sind die in dieser Arbeit vorgestellten GUI-Bibliotheken für einen besseren Überblick gegenübergestellt.

	Plattformen	Verbreitung (unter Linux)	Backend (unter Linux)	Sprachanbindung	OS-Lizenz
GTK+ / gtkmm	Linux, Windows, Mac OS X	sehr verbreitet	*X-Server*, *Wayland*, HTML5	Python, C#, Java, Perl, Haskell, ...	*LGPLv2.1*
QT	Linux, Windows, Mac OS X, iOS, Android, Sailfish OS, ...	sehr verbreitet	*X-Server*, *Wayland*, *Framebuffer*, *EGL*, *KMS + DRM*	Python, C#, Java, Haskell, ...	*LGPLv2.1* *LGPLv3.0*, *GPLv3.0*
wxWidgets	Linux, Windows, Mac OS X	verbreitet	*GTK+*, *X-Server*, *DirectFB* (eingeschränkt)	Python, Perl, Haskell, ...	mod. *LGPL*
CEGUI	Linux, Windows, Mac OS X	wenig verbreitet	*OpenGL*	Python	*MIT*

Tabelle 4.1.: Vergleich der GUI-Bibliotheken für C++

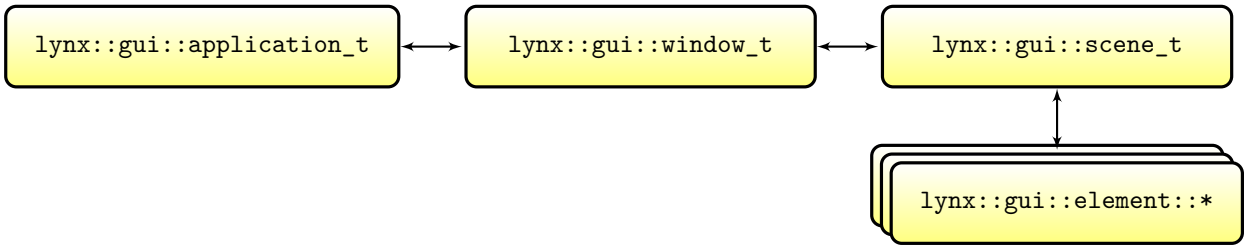


Abbildung 4.7.: Beziehung der GUI-Klassen

Für *MTPSW-Lynx* wurde *gtkmm*, also die C++-Schnittstelle von *GTK+* verwendet. Die Gründe für diese Entscheidung waren, dass durch die Verwendung des *X-Servers* die Eingabegeräteunterstützung bereits sehr gut ist. Außerdem ist durch die Unterstützung von *Wayland* ggf. auch ein Betrieb von *MTPSW-Lynx* unter *Wayland* möglich. Auch die Tatsache, dass *GTK+* sehr weit verbreitet und es dadurch einfacher ist, Hilfestellungen im Internet zu erhalten, hatte bei dieser Entscheidung einen Einfluss.

Die Entscheidung gegen *QT* trotz der zusätzlichen Backends *EGL* bzw. Linux-*Framebuffer* fiel zu Gunsten von *GTK+*, denn im Gegensatz zu *QT* ist es hier möglich, standardkonformen C++-Code zu programmieren. Außerdem ist die Performance über *X11* bzw. *Wayland* selbst mit dem bereits sehr betagten *Raspberry PI 1* ausreichend. Somit war es nicht notwendig, *EGL* bzw. Linux-*Framebuffer* zu verwenden.

Zusätzlich wurde der in Abschnitt 4.9.3 auf Seite 33 vorgestellte *Windowmanager Matchbox* eingesetzt. Grundsätzlich lässt sich aber jeder *Windowmanager* unter *X11* für *MTPSW-Lynx* betreiben.

In Abbildung 4.7 werden die in *MTPSW-Lynx* verwendeten GUI-Klassen näher beleuchtet.

Die meisten Klassen sind abgeleitet von *gtkmm*-Klassen. Alle *gtkmm*-Klassen sind leicht erkennbar durch den Namespace `Gtk`.

4.9.5. `lynx::gui::application_t`

Diese Klasse ist abgeleitet von `Gtk::Application` und wird erzeugt aus den Parametern `argc` und `argv` der `main()`-Funktion. Nach dem Start der Anwendung muss von `lynx::gui::application_t` eine `lynx::gui::window_t`-Instanz erzeugt werden. In dieser werden anschließend alle GUI-Elemente angezeigt.

Eine der Aufgaben von `lynx::gui::application_t` ist das Parsen und Validieren der *CLI*[22]-Optionen.

Nachdem die `lynx::gui::window_t`-Instanz übergeben wurde, wird mit dem Parsen der XML-Datei gestartet. Während die XML-Datei geparst wird, wird parallel dazu mit einem weiteren Thread versucht, eine Verbindung zum *LYNX-Server* herzustellen. Nach dem Parsen wird geprüft, ob der über die *CLI*-Optionen angegebene Benutzername auch in der XML-Datei als berechtigter Benutzername markiert wurde.

Treten beim Parsen der *CLI*-Optionen oder während der Verarbeitung der XML-Datei *Exceptions* auf, wird eine Fehlermeldung als Pop-Up angezeigt und anschließend das Programm beendet.

Meldet die Netzwerkklasse `net::net_t` (über *Exceptions*), dass keine Verbindung zum *LYNX-Server* hergestellt werden konnte oder dass die Verbindung zum *LYNX-Server* abgebrochen wurde, wird (ebenfalls in einem Pop-Up) abgefragt, ob die Verbindung erneut hergestellt werden soll. An-

[22] Command-line interface

schließend ist `lynx::gui::application_t` ebenfalls dafür verantwortlich, dass versucht wird, die Verbindung neu aufzubauen (wenn gewünscht).

Eine weitere Aufgabe von `lynx::gui::application_t` ist das Initialisieren des Standard-Logging. Dafür sind jedoch die *CLI*-Optionen notwendig, daher wird das Logging erst aktiv, nachdem diese geparst wurden [vgl. 66].

4.9.6. `lynx::gui::window_t`

`lynx::gui::window_t` ist abgeleitet von `Gtk::Window` und implementiert zusätzlich einen Synchronisationsmechanismus für die untergeordneten GUI-Elemente.

Das ist notwendig, denn *gtkmm* verwendet die Bibliothek *libsigc++*, die unter anderem das Signal-Slot-Konzept implementiert.[L55] Wenn allerdings mehrere Threads (wie in *MTPSW-Lynx*) auf die GUI-Objekte zugreifen, müssen zusätzliche Synchronisierungsmechanismen eingeführt werden, denn *libsigc++* ist nicht Threadsave [vgl. 3].

Daher wurde in `lynx::gui::window_t` der `Glib::Dispatcher` eingebaut, der dafür sorgt, dass alle GUI-Änderungen im Kontext des GUI-Threads ausgeführt werden.

Muss ein GUI-Element beispielsweise den angezeigten Wert ändern, weil der *LYNX-Server* die entsprechende Nachricht gesendet hat, dann synchronisiert dieses Element über die Schnittstelle seiner übergeordneten `lynx::gui::window_t`-Instanz die gewünschten Änderungen.

Zusätzlich zur Synchronisationsschnittstelle bietet `lynx::gui::window_t` auch eine Methode an, mit der die vom XML geparsten `lynx::gui::scene_t`-Instanzen der `lynx::gui::window_t`-Instanz zugeordnet werden können.

4.9.7. `lynx::gui::scene_t`

Eine Instanz der Klasse `lynx::gui::scene_t` repräsentiert ein `<scene>`-Tag der XML-Datei. Einer `lynx::gui::scene_t`-Instanz können beliebig viele `lynx::gui::element_t`-Instanzen zugeordnet werden, die in dieser `lynx::gui::scene_t`-Instanz anschließend verwaltet werden.

Wenn der im XML angegebene Benutzername nur Leserechte für alle Elemente im `<scene>`-Tag besitzt, ist `lynx::gui::scene_t` dafür verantwortlich, dass die remote Widgets von allen, die diesen Benutzernamen zum Betrieb der GUI verwenden, nicht verändert werden können.

4.9.8. `lynx::gui::element::*`

Eine Instanz einer Klasse im `lynx::gui::element`-Namespace repräsentiert ein `<element>`-Tag im XML. Welche Klasse konkret verwendet wird, hängt ab vom Typ, der im XML angegeben wurde.

In der GUI werden die Elemente als Widgets in einem Gitter angeordnet. Die Gitterabmessungen sind nicht genau festgelegt, d.h., die genaue Anzahl der Zeilen und Spalten wird nur durch die Elemente im Gitter (also durch das XML) bestimmt. Die einzige Einschränkung dabei ist, dass jede Gitterzelle dieselbe Größe besitzt, d.h., der verfügbare Platz wird so aufgeteilt, dass jedes Feld dieselbe Höhe und Breite besitzt.

In der Praxis stellt dies keine wirkliche Einschränkung dar, denn einem Element kann eine beliebige Anzahl an Gitterzellen in der Höhe und Breite zugeordnet werden. Dadurch ist es bei Bedarf möglich, gewisse Elemente breiter oder höher darzustellen. In Abbildung 4.8 auf der nächsten Seite ist ein mögliches Layout schematisch dargestellt.

Durch *Container*-Elemente, die weitere Elemente aufnehmen können, ist es möglich, Elemente zu gruppieren. In der GUI werden immer nur die Elemente angezeigt, die im selben *Container* liegen und sich gleichzeitig auf derselben Hierarchiestufe befinden.

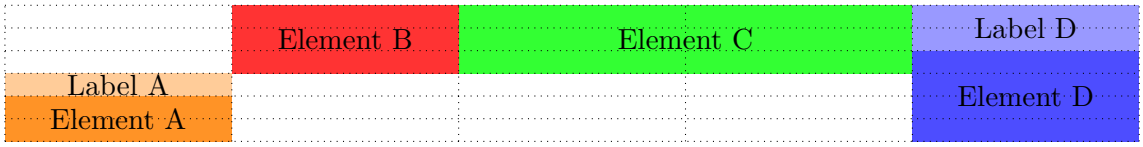


Abbildung 4.8.: Beispiel für ein gitterbasiertes Layout.

Unter den Elementen muss jedoch unterschieden werden, zwischen den sogenannten remote Widgets und den lokalen Widgets. Während die lokalen Widgets unabhängig vom *LYNX-Server* sind, können die remote Widgets Parameter am *LYNX-Server* anzeigen bzw. bearbeiten. In Abschnitt 4.9.8.2 auf der nächsten Seite bzw. Abschnitt 4.9.8.3 auf der nächsten Seite wird nochmals näher darauf eingegangen [vgl. 66].

4.9.8.1. Verfügbare Elementbasisklassen

`lynx::gui::element::base_t`
Grundsätzlich ist jedes Element (egal, ob dieses Element ein lokales- oder ein remote Widget enthält) entweder direkt oder indirekt von `lynx::gui::element::base_t` abgeleitet. Dadurch kann sichergestellt werden, dass jedes Element ein in der GUI visualisierbares *gtkmm*-Basis-Widget mit Label besitzt.

Allen Elementen wird, sofern sie in der GUI angezeigt werden sollen, im XML eine Position im Gitter zugewiesen. Beispiele dafür sind „Element B“ bzw. „Element C“ in Abbildung 4.8.

`lynx::gui::element::labeled_t`
Während manche *gtkmm*-Widgets das Label direkt anzeigen können (z.B. *Toggle*-Button), ist das bei anderen Widgets nicht möglich. Beispielsweise zeigt ein *Slider*-Widget immer nur die Slider-Position und den aktuellen Wert an. Ein zusätzliches Label hat in diesem Widget keinen Platz. Daher wird diesen Elementen, ein zusätzliches *gtkmm*-Label-Widget zugeordnet, das in der GUI anzeigt werden kann und dieses Label enthält. Beispiele dafür sind „Element A“ mit dem zugehörigen „Label A“ bzw. „Element C“ mit dem zugehörigen „Label C“ in Abbildung 4.8.

Alle Elemente, die nicht direkt das Label anzeigen, sondern ein separates *gtkmm*-Label-Widget besitzen, werden aus diesem Grund zusätzlich von `lynx::gui::element::labeled_t` abgeleitet.

Für diese Elemente kann im XML daher zusätzlich eine Label-Position angegeben werden. D.h., für Elemente, die sowohl von `lynx::gui::element::base_t`, als auch `lynx::gui::element::labeled_t` abstammen, muss, wenn beide Widgets angezeigt werden sollen, sowohl die Label-Widget-Position, als auch die Basis-Widget-Position angegeben werden [vgl. 66].

Somit erweitert die Klasse `lynx::gui::element::labeled_t` die Klasse `lynx::gui::element::base_t` um die Fähigkeit, neben dem *gtkmm*-Widget, noch ein weiteres *gtkmm*-Widget in die GUI einzubinden, das den Label-Text anzeigen kann.

`lynx::gui::element::param_t<T>`
Das Klassentemplate `lynx::gui::element::param_t<T>` erweitert `lynx::gui::element::base_t` um die Eigenschaften und Methoden, die remote Widgets besitzen müssen. Allen remote Widgets, muss ein Parameter am *LYNX-Server* zugeordnet sein. D.h., jedes Element, das ein remote Widget enthält, erbt entweder direkt, oder indirekt von `lynx::gui::element::param_t<T>`. Dadurch wird sichergestellt, dass jedem Element mit remote Widget ein Parameter zugeordnet werden kann.

Der Widget-Typ, der in der GUI für die Anzeige des Parameters zuständig ist, kann im XML definiert werden. Anhand dieser Definition wird an `lynx::gui::element::param_t<T>` der entsprechende Template-Parameter `<T>` übergeben. Sobald dieses Widget in der GUI angezeigt werden soll, wird die Klasse, die als `<T>` übergeben wurde, instanziiert und im Gitter angezeigt. Der Template-Parameter `<T>` ist dabei eine Klasse aus Abschnitt 4.9.9 auf der nächsten Seite.

Beispielsweise kann im XML angegeben werden, dass ein *ToggleButton* als Widget für einen binären Parameter angezeigt werden soll.

In *MTPSW-Lynx* ist jedes remote Widget für die Kommunikation mit dem *LYNX-Server* selbst verantwortlich. D.h., wenn Benutzereingaben den Wert eines remote Widgets ändern (z.B. indem der Slider verschoben wird, oder der Toggle-Button gedrückt wird), muss das Widget dafür sorgen, dass diese Parameteränderung dem Server kommuniziert wird. Auch umgekehrt gilt, dass das remote Widget die Parameter-Subscriptions selbst verwalten muss, um Benachrichtigt zu werden, wenn sich ein Parameter am *LYNX-Server* geändert hat. Ist dies der Fall, muss der im Widget angezeigte Wert an den *LYNX-Server*-Parameterwert angepasst werden.

Außerdem wird, sobald Benutzereingaben den Zustand eines Widgets ändern, das Elementlabel rot eingefärbt. Diese Einfärbung bleibt so lange erhalten, bis vom *LYNX-Server* die Antwort kommt, dass dieser Wert am Server auch angepasst wurde. Bleibt ein Label für längere Zeit auf der Farbe Rot, ist das ein Indiz, dass dieser Wert noch nicht vom Server bestätigt wurde.

`lynx::gui::element::labeled_param_t<T>`

Die Template-Klasse `lynx::gui::element::labeled_param_t<T>` wird abgeleitet von `lynx::gui::element::param_t<T>` und `lynx::gui::element::labeled_t`. Ebenso, wie die Klasse `lynx::gui::element::labeled_t` erlaubt die Klasse `lynx::gui::element::labeled_param_t<T>` den Elementen, die kein Label anzeigen können, ein weiteres *gtkmm*-Label-Widget im Gitter zu positionieren.

Zudem übernimmt diese Klasse das Einfärben des separaten *gtkmm*-Labels, sobald der Parameterwert über die GUI verändert wurde.

4.9.8.2. Lokale Widgets

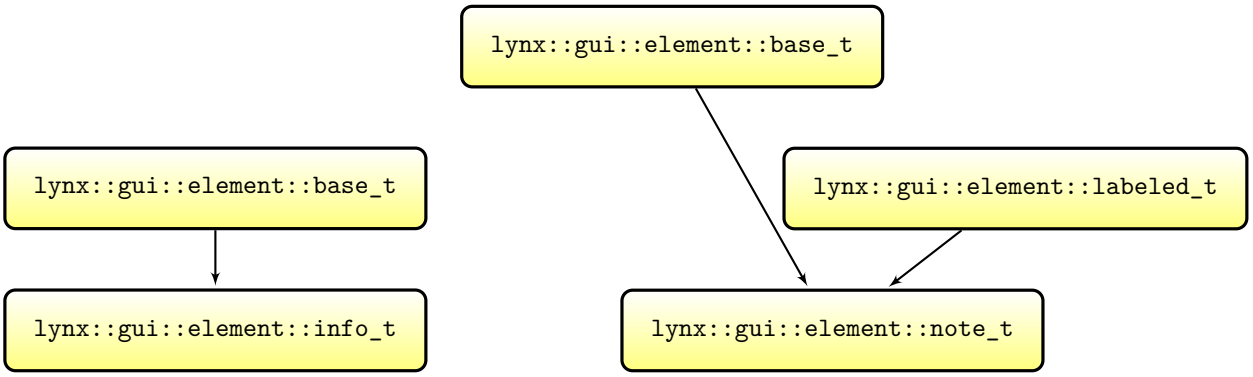
Lokale Widgets werden entweder nur von `lynx::gui::element::base_t` oder von `lynx::gui::element::base_t` und `lynx::gui::element::labeled_t` abgeleitet. Auf andere Weise können diese Widgets nicht erzeugt werden (siehe Abbildung 4.9 auf der nächsten Seite).

Zu den lokalen-Widgets zählen:

- *Info*-Widget (Definiert in Klasse `lynx::gui::element::info_t`, abgeleitet von `lynx::gui::element::base_t`)
- *Note*-Widget (Definiert in Klasse `lynx::gui::element::note_t`, abgeleitet von `lynx::gui::element::base_t` und `lynx::gui::element::labeled_t`)
- *Container*-Widget (Definiert in Klasse `lynx::gui::element::container_t`, abgeleitet von `lynx::gui::element::base_t`)

4.9.8.3. remote Widgets

remote Widgets werden entweder mit `lynx::gui::element::param_t<T>` oder mit `lynx::gui::element::labeled_param_t<T>` instanziiert. Wobei der Template-Parameter `<T>` angibt, welchen



(a) Vererbungsstruktur des *Info*-Widget (b) Vererbungsstruktur des *Note*-Widget

Abbildung 4.9.: Vererbungsstruktur der lokalen Widgets

Typ das remote Widget besitzt, d.h., welches Widget in der GUI zum Anzeigen/Verändern des Parameterwertes verwendet wird (siehe Abschnitt 4.9.9 für mögliche Template-Parameter). Auf andere Weise können diese Widgets nicht erzeugt werden (siehe Abbildung 4.10 auf der nächsten Seite).

4.9.9. `lynx::gui::widget::*`

Alle Klassen im Namespace `lynx::gui::widget` können als Template-Parameter `T` in den Klassen `lynx::gui::element::param_t<T>` bzw. `lynx::gui::element::labeled_param_t<T>` verwendet werden.

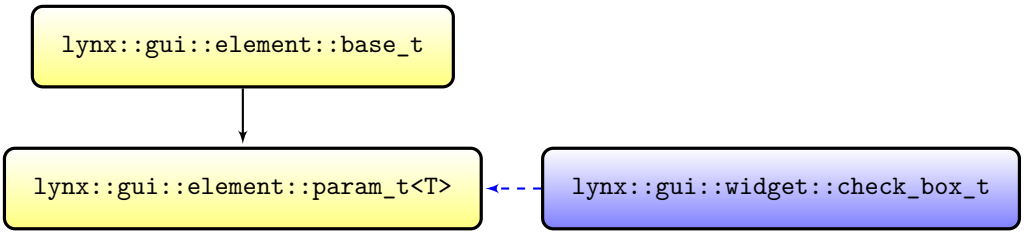
Ein *LYNX-Server* ist für unterschiedliche Parametertypen zuständig: Einfachen Text, binäre Werte, Auswahlen und Fließkommawerte.

Die Klassen `lynx::gui::widget::label_t` (für einfachen Text), `lynx::gui::widget::toggle_t` (für binäre Werte), `lynx::gui::widget::select_t` (für Auswahlen) und `lynx::gui::widget::float_t` (für Fließkommawerte) bündeln die Funktionalität, die innerhalb der Parametertypen gleich ist und definieren einheitliche Interfaces für die einzelnen Parametertypen, die von den instanziierbaren Klassen implementiert werden.

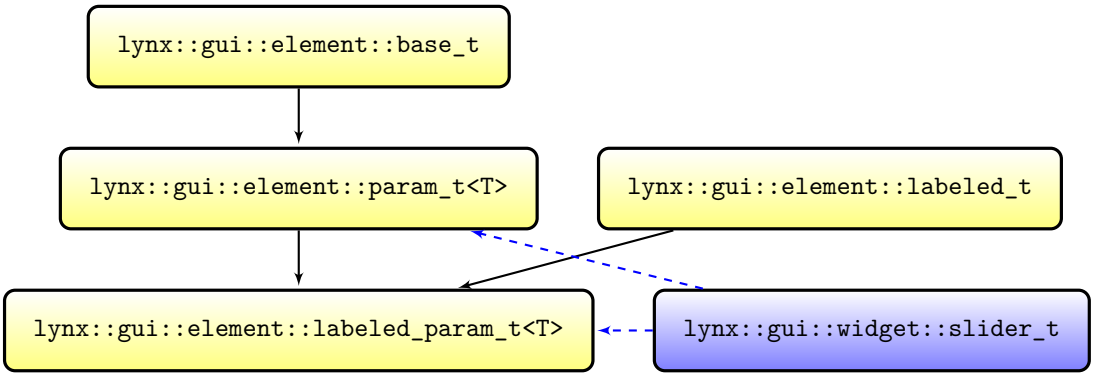
Bei allen instanziierbaren Klassen im Namespace `lynx::gui::widget` handelt es sich um Wrapperklassen über *gtkmm*-Widgets.

Diese Wrapperklassen sind notwendig, um allen Widgets, desselben Parametertyps ein einheitliches Interface zu ermöglichen. Beispielsweise besitzen die beiden *gtkmm*-Klassen `Gtk::RadioButton` und `Gtk::ComboBoxText` eine völlig unterschiedliche und zueinander inkompatible API. Dennoch eignen sie sich beide zur Verwendung von Auswahl-Parametern. Die Wrapperklassen `lynx::gui::widget::radio_buttons_t` bzw. `lynx::gui::widget::select_drop_down_t` erzeugen um diese *gtkmm*-Widgets eine einheitliche API. Dadurch sind beide Klassen gegenseitig austauschbar.

Im Folgenden wird noch näher auf die einzelnen Klassen im Namespace `lynx::gui::widget` eingegangen.



(a) Vererbungsstruktur des *Parameter*-Widgets `lynx::gui::element::param_t<T>` mit dem Template-Parameter `T=lynx::gui::check_box_t`.



(b) Vererbungsstruktur des *Labeled-Parameter*-Widgets `lynx::gui::element::labeled_param_t<T>` mit dem Template-Parameter `T=lynx::gui::slider_t`.

Abbildung 4.10.: Vererbungsstruktur der remote Widgets

4.9.9.1. Widgets für einfachen Text

`lynx::gui::widget::label_t`

Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::label_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden.

Die Klasse `lynx::gui::widget::label_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::Label`. Die Vererbungsstruktur von `lynx::gui::widget::label_t` findet sich in Abbildung 4.11.

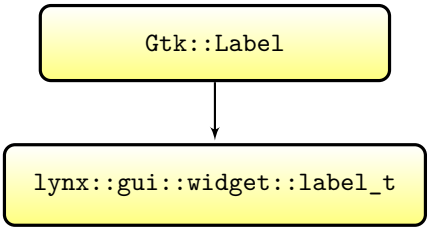


Abbildung 4.11.: Vererbungsstruktur des Widgets für einfachen Text (`lynx::gui::widget::label_t`).

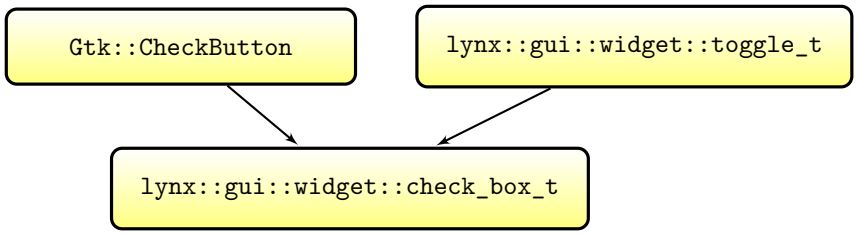


Abbildung 4.12.: Vererbungsstruktur des Widgets für binäre Werte (`lynx::gui::widget::label_t`).

4.9.9.2. Widgets für binäre Werte

`lynx::gui::widget::check_box_t`

Diese Klasse kann das Element-Label anzeigen, daher sollte `lynx::gui::widget::check_box_t` als Template-Parameter der Elementklasse `lynx::gui::element::param_t<T>` verwendet werden. Die Klasse `lynx::gui::widget::check_box_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::CheckButton` und von der abstrakten Klasse `lynx::gui::widget::toggle_t`. Die Vererbungsstruktur von `lynx::gui::widget::check_box_t` findet sich in Abbildung 4.12.

`lynx::gui::widget::button_t`

Diese Klasse kann das Element-Label anzeigen, daher sollte `lynx::gui::widget::button_t` als Template-Parameter der Elementklasse `lynx::gui::element::param_t<T>` verwendet werden. Die Klasse `lynx::gui::widget::button_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::ToggleButton` und von der abstrakten Klasse `lynx::gui::widget::toggle_t`.

`lynx::gui::widget::check_button_t`

Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::check_button_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden. Die Klasse `lynx::gui::widget::check_button_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::ToggleButton` und von der abstrakten Klasse `lynx::gui::widget::toggle_t`.

`lynx::gui::widget::status_label_t`

Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::status_label_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden. Die Klasse `lynx::gui::widget::status_label_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::ToggleButton` und von der abstrakten Klasse `lynx::gui::widget::toggle_t`.

`lynx::gui::widget::led_label_t`

Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::led_label_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden. Die Klasse `lynx::gui::widget::led_label_t` wird abgeleitet von der Klasse `lynx::gui::widget::status_label_t`.

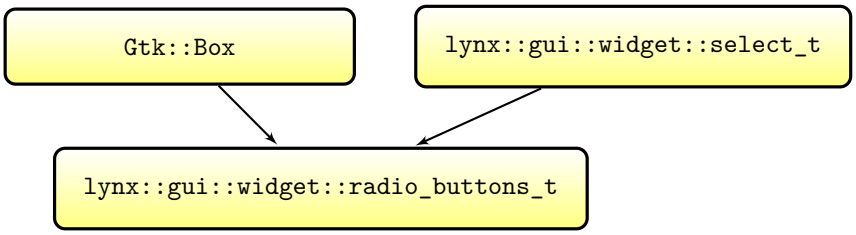


Abbildung 4.13.: Vererbungsstruktur des Widgets für Fließkomma-Werte (am Beispiel `lynx::gui::widget::radio_buttons_t`).

4.9.9.3. Widgets für Auswahlen

`lynx::gui::widget::radio_buttons_t`
Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::radio_buttons_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden.

Die Klasse `lynx::gui::widget::radio_buttons_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::Box` (die wiederum aus einer Liste von `Gtk::RadioButton`-Buttons besteht) und von der abstrakten Klasse `lynx::gui::widget::select_t`. Die Vererbungsstruktur von `lynx::gui::widget::radio_buttons_t` findet sich in Abbildung 4.13.

`lynx::gui::widget::select_drop_down_t`
Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::select_drop_down_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden. Die Klasse `lynx::gui::widget::select_drop_down_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::ComboBoxText` und von der abstrakten Klasse `lynx::gui::widget::select_t`.

4.9.9.4. Widgets für Fließkommawerte

`lynx::gui::widget::slider_t`
Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::slider_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden.

Die Klasse `lynx::gui::widget::slider_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::HScale` und von der abstrakten Klasse `lynx::gui::widget::float_t`. Die Vererbungsstruktur von `lynx::gui::widget::slider_t` findet sich in Abbildung 4.14 auf der nächsten Seite

`lynx::gui::widget::numeric_select_t`
Dadurch, dass diese Klasse nur den Parameterwert und nicht das Element-Label anzeigen kann, sollte `lynx::gui::widget::numeric_select_t` als Template-Parameter der Elementklasse `lynx::gui::element::labeled_param_t<T>` verwendet werden.

Die Klasse `lynx::gui::widget::numeric_select_t` wird abgeleitet vom *gtkmm*-Widget `Gtk::SpinButton` und von der abstrakten Klasse `lynx::gui::widget::float_t`.

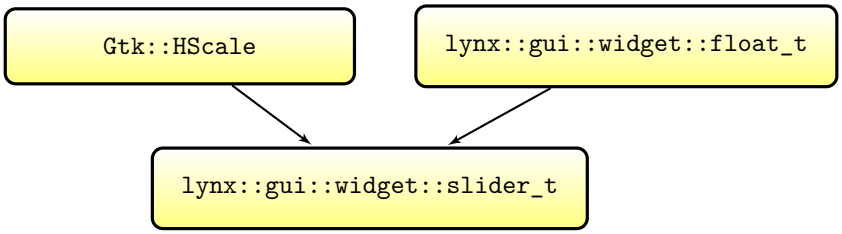


Abbildung 4.14.: Vererbungsstruktur des Widgets für Fließkomma-Werte (am Beispiel `lynx::gui::widget::slider_t`).

5. Zusammenfassung und Ausblick

Das im Rahmen der LVA *Bachelorarbeit für Informatik und Wirtschaftsinformatik* entwickelte Touchpanel ist bereits vollständig einsatzfähig. Allerdings sollte auch dieses Produkt ständig weiterentwickelt werden. In diesem Kapitel wird abschließend darauf eingegangen, welche weiteren Schritte dafür notwendig sind.

Zukünftige C++ Versionen

Durch die Verwendung von C++14 war es möglich, viele standardisierte Komponenten im Code zu verwenden und die Abhängigkeit zu externen Programmbibliotheken auf ein Minimum zu reduzieren.

Für die nächste C++-Version C++1z bzw. C++17 ist nach aktuellem Stand bereits geplant, die Standardbibliothek durch weitere Bibliotheken zu erweitern. Unter anderem soll eine Netzwerkbibliothek (basierend auf `boost::asio`[L47]) und eine Filesystembibliothek (basierend auf `boost::filesystem`[L22]) hinzugefügt werden.

In Zukunft kann daher die Abhängigkeit von nicht standardisierten Bibliotheken weiter reduziert werden. Da die geplante Netzwerkbibliothek für den zukünftigen C++-Standard auf `boost::asio` basiert, sollte dafür nicht viel Portierungsaufwand notwendig sein.

Grafikschnittstelle bzw. GUI-Bibliothek

Wie bereits in Abschnitt 4.9 auf Seite 26 erwähnt, wird das neue *Wayland*-Protokoll bereits von den *GTK+*-Bibliotheken (und der dazugehörigen Sprachanbindung *gtkmm*) unterstützt.

Da *Wayland* bereits von sehr vielen Distributionen als Nachfolger des sehr verbreiteten *X-Servers* gehandelt wird, sind in Zukunft immer mehr *Wayland*-Installationen zu erwarten.

Durch den Wechsel von *X-Server* auf *Wayland* sind geringere Hardwareanforderungen bei gleichbleibender Stabilität zu erwarten. Daher sollte dieser Wechsel nur Vorteile bringen.

Zukünftige Hardware

Das in *MTP-Lynx* verwendete *Cubieboard 2* wird in Zukunft durch einen anderen *SBC* ersetzt werden, denn die Hardwareunterstützung des *Cubieboard 2* in aktuellen Kerneln ist für *MTP-Lynx* leider unzureichend und die Verwendung des mehrere Jahre alten *sunxi-3.4*-Kernel ist keine Option.

Eine mögliche Alternative zum *Cubieboard 2* ist der bereits in Abschnitt 3.2 auf Seite 5 erwähnte *Raspberry PI 2*. Um das bereits verwendete Display *TFT1280120-1-E* weiterverwenden zu können, ist es notwendig, einen weiteren Adapter zu entwickeln, der das HDMI-Signal vom *Raspberry PI 2* in ein *LVDS*-Signal umwandelt. Alternativ kann das in Abschnitt 3.2 auf Seite 5 vorgestellte Display *EarthLCD-10.4-1024100* verwendet werden. Dieses kann direkt am *Raspberry PI 2* betrieben werden.

Einfachere Erstellung der XML-Datei

Um die XML-Datei einfacher erstellen zu können, ist außerdem eine *gtkmm*-basierte Desktopanwendung geplant, welche die von der *APPolo | Control*-GUI exportierte XML-Datei auswertet und eine intuitive Möglichkeit bieten soll, die darin enthaltenen Widgets im gitterbasierten Layout zu positionieren. Dadurch ist es für den Benutzer nicht mehr notwendig, den Inhalt der XML-Datei aus der *APPolo | Control*-GUI über einen Texteditor anzupassen.

A. Adapterplatine

A.1. Leiterplatten-Layout

Das Leiterplattenlayout für das Adapterboard finden Sie in Abbildung A.1 auf der nächsten Seite.

A.2. Schaltungen

A.2.1. PWM-Schaltung

Die PWM-Schaltung für die Hintergrundbeleuchtung des Displays ist in Abbildung A.2 auf Seite 47 skizziert.

Dort ist ersichtlich, dass das *PWM*-Signal über einen $1k\Omega$-Widerstand an das *Gate* eines selbstsperrenden *n-Kanal-MOSFET* angelegt wird. Der $1k\Omega$-Widerstand soll dabei den Strom zum *Gate* des *MOSFET* begrenzen.

Zwischen dem *Drain* des *MOSFET* und der Versorgungsspannung wird die Hintergrundbeleuchtung versorgt. Außerdem ist die *Source* mit dem *Drain* eines weiteren *MOSFET* verbunden.

An dessen *Gate* liegt (ebenfalls über einen $1k\Omega$-Widerstand) das vom *Cubieboard 2* erzeugte Signal *BL_EN* an. Somit lässt sich unabhängig vom *PWM*-Signal die Displaybeleuchtung abschalten.

Erst an der *Source* des zweiten *MOSFET* liegt Masse (*GND*).

Die Versorgungsspannung wird erzeugt aus einem 10Ω-Widerstand, der mit der Hintergrundbeleuchtung einen Spannungsteiler bildet. Dadurch fallen an diesem Widerstand etwa $2V$ ab und es wird die benötigte Versorgungsspannung von $9V - 10V5$ erzeugt.

Grundsätzlich wäre es aber auch möglich, einen *DC/DC-Konverter* wie *TSR 1-2490* zu verwenden, der aus $12V$ Eingangsspannung die benötigten $9V$ liefern kann [vgl. 14, S. 1].

A.2.2. Spannungsversorgung

Die Schaltung für die Spannungsversorgung des Adapterboards (inkl. Jumpersteckplatz) ist in Abbildung A.3 auf Seite 47 dargestellt.

Während die Versorgung der $3V3$ vom *Cubieboard 2* direkt verwendet werden kann, muss die $12V$-Spannung vom *Cubie-Baseboard* erst auf die benötigten $3V3$ reduziert werden.

Dazu wurde der *DC/DC-Konverter Tracopower TSR 1-2433* verwendet, der aus einer Eingangsspannung zwischen $4V75$ und $36V$ eine Spannung von $3V3$ erzeugen kann [vgl. 14, S. 1].

Wie in Abschnitt 3.3.2 auf Seite 8 beschrieben, kann zwischen den beiden Versorgungsspannungen mittels Jumper umgeschaltet werden.

A.2.3. *LVDS* in *parallel-RGB*

Die Schaltung für den *LVDS* in *parallel-RGB*-Konverter, der bereits in Abschnitt 3.3.1 auf Seite 7 erklärt wurde, ist in Abbildung A.4a auf Seite 49 kurz skizziert.

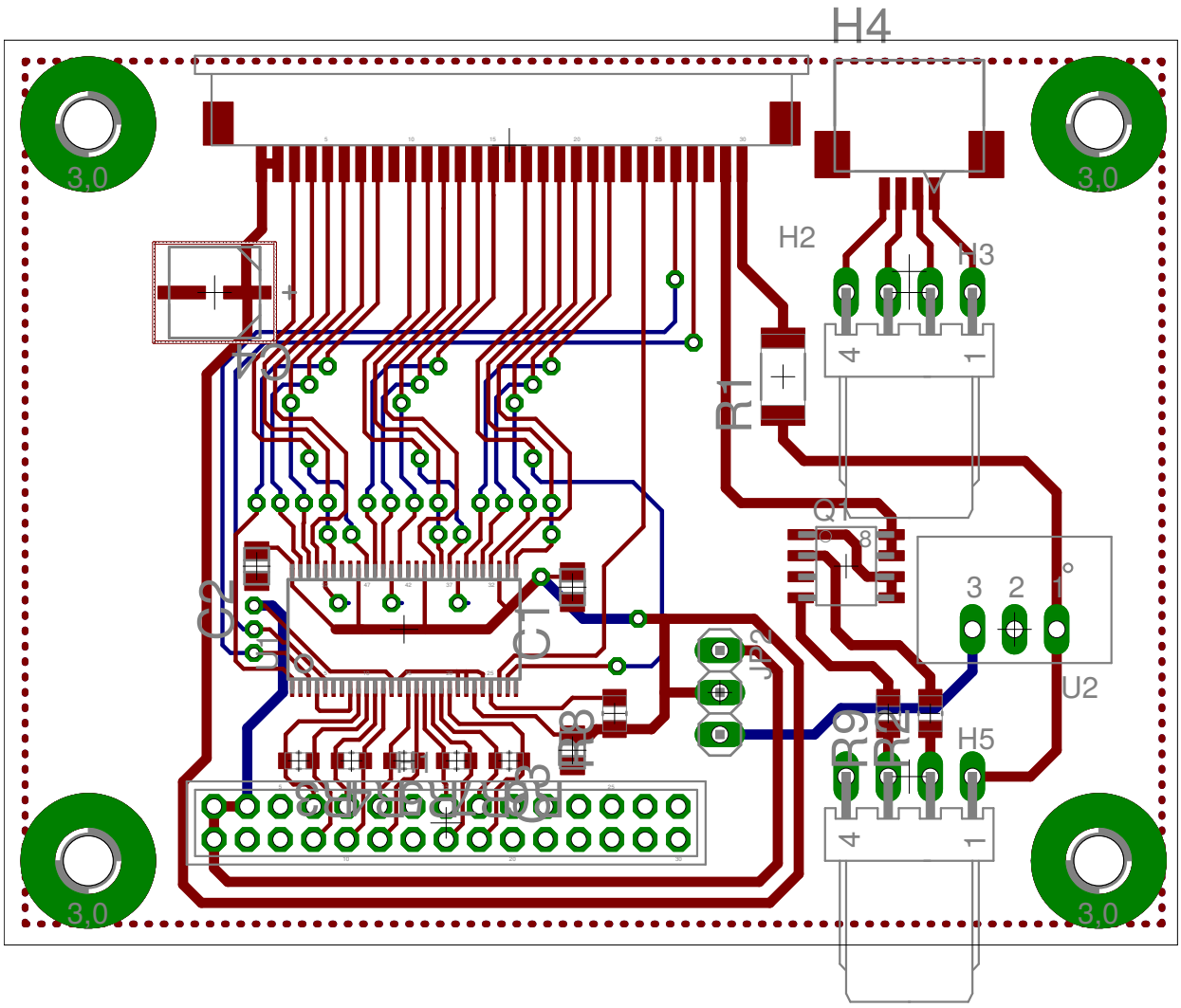


Abbildung A.1.: Leiterplattenlayout der Adapterplatine.

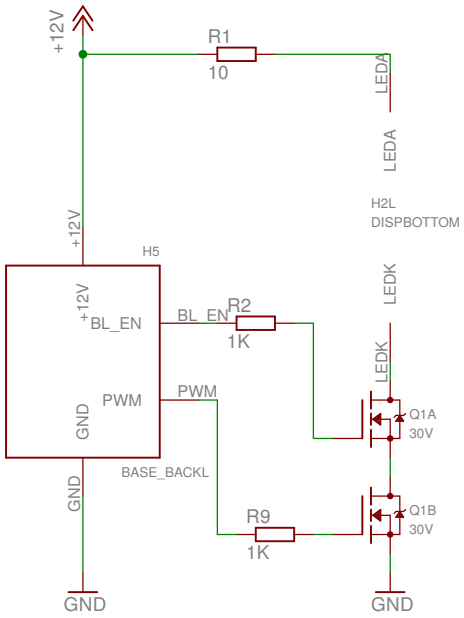


Abbildung A.2.: PWM-Schaltung für die Hintergrundbeleuchtung des Displays.

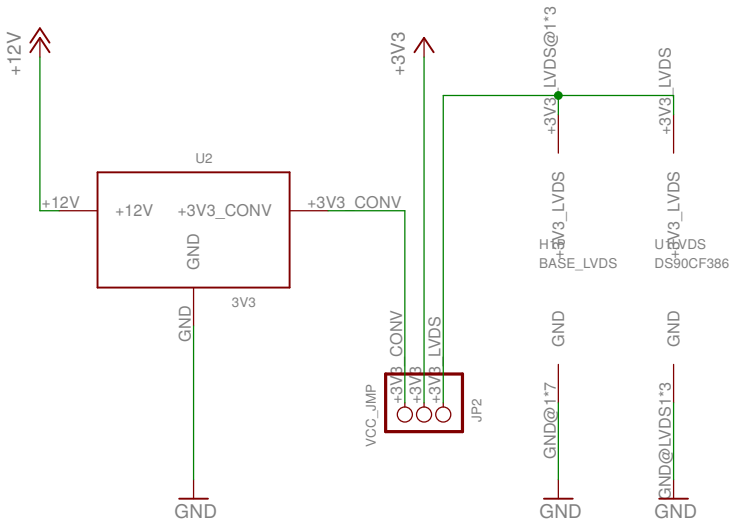


Abbildung A.3.: Spannungsversorgung des Adapterboards.

Wellenwiderstände

In Abbildung A.1 auf Seite 46, bzw. Abbildung A.4a auf der nächsten Seite sind die 100Ω-SMD-Widerstände vor dem *DS90CF86* zu sehen. Diese sind nötig, weil die Flanken der *LVDS*-Signale relativ steil sind. Typischerweise beträgt CHLT[1] für diesen Chip $1.8ns$ [vgl. 27, S. 3]. Daher sind bereits nach wenigen Zentimetern Abschlusswiderstände (Wellenwiderstände) nötig, um störende Reflexionen zu minimieren [vgl. 26, S. 6, bzw. 67].

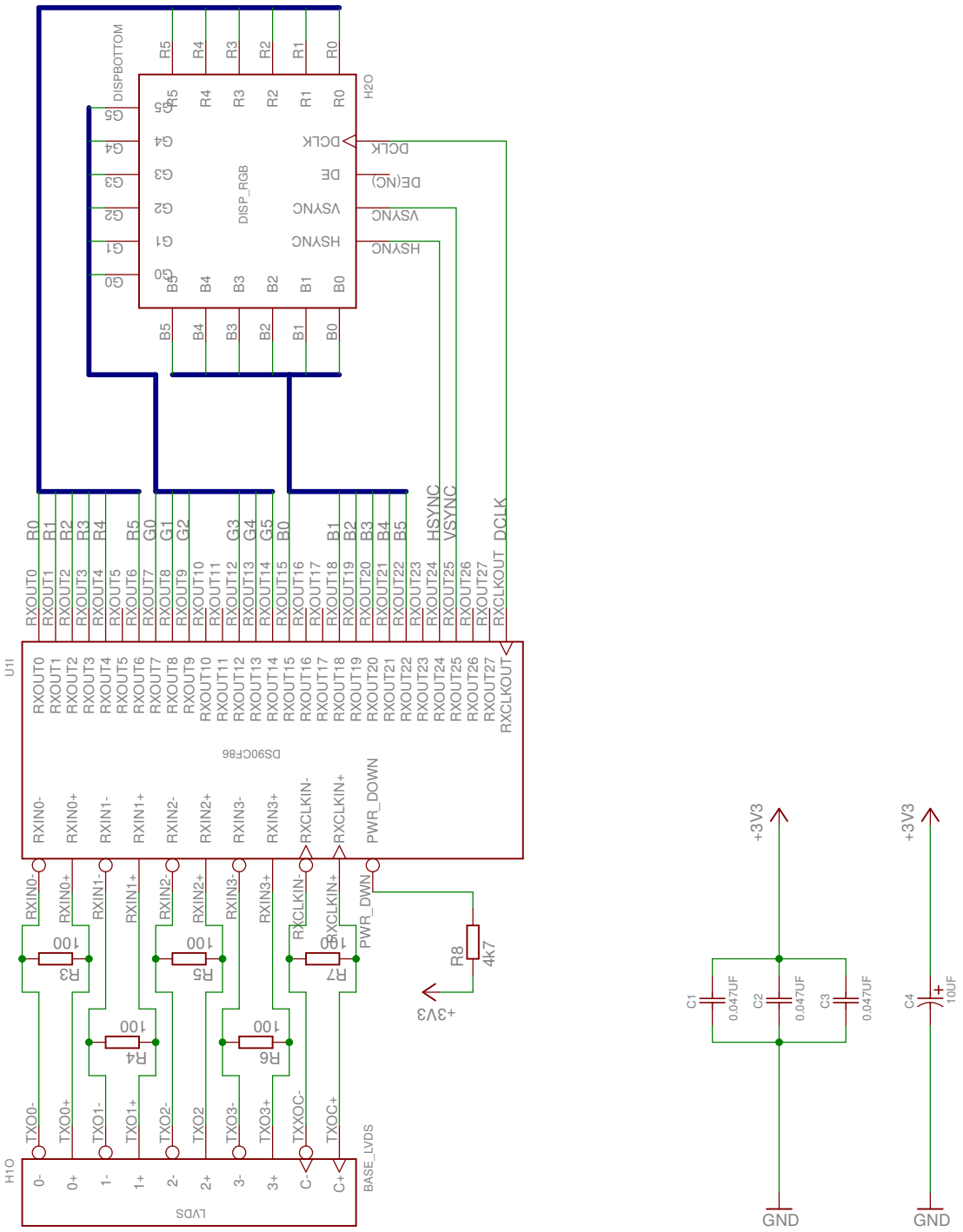
Sobald *LVDS*-Leitungen verwendet werden, muss ebenfalls sehr stark auf die Länge der Leitungen geachtet werden. Für die Entwicklung der Adapterplatine war es daher wichtig, dass sich die Längen der Signalpaare insgesamt kaum unterscheiden, vor allem aber müssen die Längen beider Leitungen im Signalpaar gleich sein [vgl. 26, S. 5].

Stützkondensatoren

Wie in Abbildung A.4b auf der nächsten Seite zu sehen, wurde außerdem für das Display ein $10\mu F$-Stützkondensator und für den *DS90CF86*-Chip drei $47nF$-Stützkondensatoren verbaut, um Störungen durch schwankende Versorgungsspannungen, die durch das permanente und hochfrequente Schalten des ICs entstehen, zu minimieren [vgl. 52].

Idealerweise befinden sich diese Stützkondensatoren, direkt neben den Chips, deren Spannungsversorgung sie „stützen“ sollen (siehe Abbildung A.1 auf Seite 46).

[1] CMOS/TTL High-to-Low Transition Time



(a) Schaltung zum Umwandeln von *LVDS* in *parallel-RGB*.

(b) Stützkondensatoren für den *DS90CF86* (links) und das Display *TFT1280120-1-E* (rechts).

Abbildung A.4.: *LVDS*-Schaltung inkl. Stützkondensatoren.

B. Log

B.1. Debug-Log mit NDEBUG

In Listing B.1 ist eine vereinfachte Version der Debug-Log-Bibliothek zu sehen, die für den Fall, dass das Makro NDEBUG nicht gesetzt wurde, den Logeintrag "Debug-Log only visible if NDEBUG is not set." auf STDOUT ausgibt.

Listing B.1: Vereinfachte Form der Debug-Log-Bibliothek.

```
0  #include <boost/log/trivial.hpp>
1
2  #ifdef NDEBUG
3  #define DBG_LOG(sev) \
4      while (false) ::dummy_log_t()
5  #else
6  #define DBG_LOG(sev) BOOST_LOG_TRIVIAL(sev)
7  #endif
8
9  class dummy_log_t {
10 public:
11     template <typename T>
12     inline dummy_log_t &operator<<(const T &) noexcept {
13         return *this;
14     }
15
16     template <typename T>
17     inline dummy_log_t &operator<<(T &&) noexcept {
18         return *this;
19     }
20 };
21
22 int main(void) {
23     DBG_LOG(warning) << "Debug-Log␣only␣visible␣if␣NDEBUG␣is␣not␣set.";
24     // BOOST_LOG_TRIVIAL(trace) << "Always visible, also if NDEBUG is set.";
25     return 0;
26 }
```

Ist jedoch NDEBUG beim Kompilieren gesetzt, vereinfacht sich das Programm mit eingeschalteter Optimierung so, als würde der Inhalt von Listing B.2 kompiliert werden.

Listing B.2: Ein zu Listing B.1 äquivalentes Programm (mit NDEBUG und Compileroptimierung)

```
0  #include <boost/log/trivial.hpp>
1  int main(void) { return 0; }
```

Dass beide Varianten zum selben Programm übersetzt werden, kann beispielsweise durch folgendes Script in Listing B.3 nachgeprüft werden.

Listing B.3: Script zum Vergleich von Listing B.1 auf der vorherigen Seite und Listing B.2 auf der vorherigen Seite.

```
0 #!/bin/bash
1 g++ -DBOOST_LOG_DYN_LINK -DNDEBUG -Wall -pedantic -Wextra -Werror -Wconversion
        -fdiagnostics-color=auto -O2 -std=c++14 -lboost_log -lpthread -o example
        dbg_log.cpp
2 objdump -d example > out1
3 g++ -DBOOST_LOG_DYN_LINK -DNDEBUG -Wall -pedantic -Wextra -Werror -Wconversion
        -fdiagnostics-color=auto -O2 -std=c++14 -lboost_log -lpthread -o example
        empty.cpp
4 objdump -d example > out2
5 diff out1 out2
```

C. RegEx-Evaluierung

Zur Evaluierung, welche RegEx-Bibliotheken verwendet werden sollten, wurde das in Listing C.1 auf Seite 56 zu findende Evaluierungsprogramm verwendet, das anhand des in *MTPSW-Lynx* verwendeten regulären Ausdrucks die Programmlaufzeit der einzelnen RegEx-Bibliotheken überprüft.

Für einen Testdurchlauf wurde der reguläre Ausdruck 100.000-Mal angewandt. Um die Auswirkung von statistischen Ausreißern zu minimieren, wurde der Testdurchlauf 1.000-Mal wiederholt und davon der Median gebildet.

Wie bereits in Abschnitt 4.7 auf Seite 20 erwähnt, wurden vier Bibliotheken im Rahmen dieser Evaluierung getestet. Diese waren:

- *PCRE*, in den Tabellen als „pcre“ bezeichnet
- *Posix Regex*, in den Tabellen als „posix“ bezeichnet
- `std::regex`. in den Tabellen als „std“ bezeichnet
- *Boost.Regex*, in den Tabellen als „boost“ bezeichnet.

In den folgenden Tabellen (Abschnitt C.1 bis Abschnitt C.3 auf Seite 56) sind die Ergebnisse dieser Evaluierung zu sehen.

C.1. Ergebnisse mit Intel i7-3770k

C.1.1. Unter *Ubuntu 15.10 (Wily Werewolf)*

GCC

lib	Median[ms]	Min[ms]	Max[ms]
pcre	259, 46	258, 18	412, 59
posix	1.162, 06	1.157, 60	1.644, 97
std	956, 60	917, 95	1.653, 12

Tabelle C.1.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `g++-4.9`

lib	Median[ms]	Min[ms]	Max[ms]
pcre	243, 77	242, 92	334, 66
boost	355, 82	354, 86	446, 63
posix	1.145, 78	1.141, 46	1.604, 84
std	331, 76	330, 47	422, 42

Tabelle C.2.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `g++-5.0`

Clang - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	246,00	241,60	335,66
boost	359,43	357,27	453,20
posix	1.141,65	1.137,44	1.488,37
std	281,34	279,83	408,72

Tabelle C.3.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.4` - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	244,77	243,78	383,75
boost	365,78	365,12	474,78
posix	1.141,90	1.138,17	1.894,52
std	259,11	257,68	392,83

Tabelle C.4.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.5` - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	246,49	244,67	371,11
boost	362,00	360,96	452,57
posix	1.140,96	1.137,92	1.415,27
std	267,71	266,53	358,95

Tabelle C.5.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.6` - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	247,40	246,26	363,88
boost	361,73	360,90	452,45
posix	1.143,55	1.138,64	1.603,74
std	267,98	266,81	369,16

Tabelle C.6.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.7` - libstdc++

Clang - libc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	237,11	234,85	327,53
boost	375,58	374,13	465,78
posix	1.140,32	1.136,02	1.649,15
std	2.620,53	2.597,56	3.192,30

Tabelle C.7.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.4` - libc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	237,00	235,37	327,63
boost	362,79	361,14	465,36
posix	1.145,03	1.138,64	1.935,70
std	2.668,58	2.648,06	3.816,35

Tabelle C.8.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.5` - libc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	238,24	236,64	381,97
boost	388,90	387,84	479,43
posix	1.140,01	1.136,50	1.414,84
std	2.621,20	2.595,34	3.134,64

Tabelle C.9.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.6` - libc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	237,79	236,68	374,48
boost	357,98	357,12	449,58
posix	1.141,54	1.136,15	1.952,26
std	2.585,56	2.561,25	3.373,67

Tabelle C.10.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `clang++-3.7` - libc++

C.1.2. Unter *Debian 8 (Jessie)*

GCC

lib	Median[ms]	Min[ms]	Max[ms]
pcre	267,62	264,86	357,97
boost	367,70	365,88	461,65
posix	1.049,88	1.015,32	1.166,46
std	865,67	848,77	984,64

Tabelle C.11.: Ergebnisse am Intel i7-3770k mit *Ubuntu 15.10 (Wily Werewolf)*, übersetzt mit `g++-4.9`

Clang - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	279,90	277,31	371,27
boost	378,49	376,36	470,53
posix	1.056,38	1.018,33	1.211,64
std	884,66	869,81	1.215,41

Tabelle C.12.: Ergebnisse am Intel i7-3770k mit *Debian 8 (Jessie)*, übersetzt mit `clang++-3.4` - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	310,85	307,95	405,18
boost	380,16	377,19	472,50
posix	1.046,23	1.017,93	1.163,22
std	875,29	852,71	976,69

Tabelle C.13.: Ergebnisse am Intel i7-3770k mit *Debian 8 (Jessie)*, übersetzt mit `clang++-3.5` - libstdc++

Clang - libc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	282,02	278,52	374,11
boost	376,90	374,16	470,51
posix	1.009,22	998,87	1.141,31
std	2.526,50	2.502,41	2.722,99

Tabelle C.14.: Ergebnisse am Intel i7-3770k mit *Debian 8 (Jessie)*, übersetzt mit `clang++-3.4` - libc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	252,33	251,31	343,75
boost	359,15	357,14	450,34
posix	1.003,78	995,09	1.145,34
std	2.504,85	2.478,93	2.717,29

Tabelle C.15.: Ergebnisse am Intel i7-3770k mit *Debian 8 (Jessie)*, übersetzt mit `clang++-3.5` - libc++

C.2. Ergebnisse mit *Raspberry PI 2* unter *Raspbian Jessie*

GCC

lib	Median[ms]	Min[ms]	Max[ms]
pcre	3.267,99	3.239,14	3.289,66
boost	2.829,48	2.829,04	2.845,55
posix	15.415,80	15.347,70	15.551,10
std	12.873,70	12.859,60	14.098,80

Tabelle C.16.: Ergebnisse am *Raspberry PI 2* mit *Raspbian Jessie*, übersetzt mit `g++-4.9`

Clang

lib	Median[ms]	Min[ms]	Max[ms]
pcre	3.242,68	3.235,53	3.260,62
boost	3.002,96	3.002,23	3.025,04
posix	15.409,10	15.290,80	15.608,80
std	13.392,50	13.387,50	15.285,80

Tabelle C.17.: Ergebnisse am *Raspberry PI 2* mit *Raspbian Jessie*, übersetzt mit `clang++-3.4 -libstdc++`

lib	Median[ms]	Min[ms]	Max[ms]
pcre	3.239, 18	3.223, 11	3.258, 68
boost	2.973, 10	2.972, 50	2.974, 55
posix	15.373, 50	15.264, 90	15.569, 30
std	12.905, 00	12.903, 60	14.752, 30

Tabelle C.18.: Ergebnisse am *Raspberry PI 2* mit *Raspbian Jessie*, übersetzt mit `clang++-3.5 -libstdc++`

C.3. Ergebnisse mit *Cubieboard 2* unter *armbian (Debian Jessie)*

GCC

lib	Median[ms]	Min[ms]	Max[ms]
pcre	3.357, 96	3.310, 65	3.559, 25
boost	2.800, 63	2.799, 40	2.904, 88
posix	15.027, 50	14.903, 90	15.454, 40
std	12.157, 00	12.120, 20	14.022, 70

Tabelle C.19.: Ergebnisse am *Cubieboard 2* mit *armbian (Debian Jessie)*, übersetzt mit `g++-4.9`

Clang

lib	Median[ms]	Min[ms]	Max[ms]
pcre	3.290, 76	3.290, 00	3.468, 72
boost	2.934, 34	2.933, 43	2.937, 65
posix	14.941, 10	14.706, 60	15.224, 80
std	14.357, 10	14.338, 40	16.273, 30

Tabelle C.20.: Ergebnisse am *Cubieboard 2* mit *armbian (Debian Jessie)*, übersetzt mit `clang++-3.4` - libstdc++

lib	Median[ms]	Min[ms]	Max[ms]
pcre	3.300, 28	3.299, 18	3.358, 47
boost	2.938, 90	2.938, 25	2.996, 67
posix	14.960, 00	14.732, 50	15.599, 50
std	13.291, 30	13.288, 70	15.143, 30

Tabelle C.21.: Ergebnisse am *Cubieboard 2* mit *armbian (Debian Jessie)*, übersetzt mit `clang++-3.5` - libstdc++

C.4. Programmcode des Tests

Listing C.1: Programm zur Evaluierung der RegEx-Parser-Bibliotheken.

```
0 #include <regex.h>
```

```
1  #include <pcre.h>
2  #include <boost/regex.hpp>
3
4  #include <string.h>
5  #include <iostream>
6  #include <stdexcept>
7  #include <chrono>
8  #include <regex>
9  #include <set>
10 #include <iomanip>
11 #include <numeric>
12 #include <algorithm>
13
14 #define MAX_MATCH 100
15 #define OVECCOUNT 90u // multiple of 3
16
17 #ifndef COMPILER
18 static_assert(false, "you␣need␣to␣give␣compiler␣ID.");
19 #endif
20
21 #define DEFAULT_REGEX \
22     R"(^((?:[^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*);)" \
23     R"((L|P|D|DN|R|W|LS|PS|RS|LU|PU|RU|LK|PK|RK|F);)" \
24     R"(((?:[^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*);)" \
25     R"(((?:[^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*);)" \
26     R"(((?:[^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;|\\:)*))" \
27     R"((?:;((?:[^\x00-\x1F\x80-\xFF\\;])" \
28     R"(|\\\\|\\;|\';\')*))?$)"
29
30 using measured_t = std::vector<std::size_t>;
31
32 struct results_t {
33     std::string address, command, client, param, value, error;
34 };
35
36 bool check_results(const results_t &res) {
37     return res.address == "add\\\\ress" && res.command == "L" &&
38           res.client == "client␣id" && res.param == "param\\;cc" &&
39           res.value == "value" && res.error == "error␣failed";
40 }
41
42 void pcre_regex(std::size_t num, const std::string &str, results_t &res) {
43     int ovector[OVECCOUNT];
44     static const std::string pattern{DEFAULT_REGEX};
45     const char *error;
46     int erroffset;
47     pcre *re;
```

```
48
49      // code based on /usr/share/doc/libpcre3-dev/examples/pcredemo.c.gz in
50      // package libpcre3-dev.
51      if ((re = pcre_compile(pattern.c_str(), 0, &error, &erroffset, NULL)) ==
52          NULL) {
53          std::string err = std::string("PCRE␣compilation␣failed␣at␣offset␣") +
54                            std::to_string(erroffset) + ":␣" + error;
55          throw std::runtime_error(err);
56      }
57
58      for (auto i = 0u; i < num; i++) {
59          auto rc = pcre_exec(re, NULL, str.c_str(), static_cast<int>(str.size()),
60                              0, 0, ovector, OVECCOUNT);
61
62          if (rc < 0) {
63              pcre_free(re);
64              if (rc == PCRE_ERROR_NOMATCH) {
65                  throw std::runtime_error("NO␣MATCH");
66              }
67              throw std::runtime_error("ERROR");
68          }
69
70          res = {
71              std::string{
72                  str.c_str() + ovector[2 * 1],
73                  static_cast<std::size_t>(ovector[2 * 1 + 1] - ovector[2 * 1])},
74              std::string{
75                  str.c_str() + ovector[2 * 2],
76                  static_cast<std::size_t>(ovector[2 * 2 + 1] - ovector[2 * 2])},
77              std::string{
78                  str.c_str() + ovector[2 * 3],
79                  static_cast<std::size_t>(ovector[2 * 3 + 1] - ovector[2 * 3])},
80              std::string{
81                  str.c_str() + ovector[2 * 4],
82                  static_cast<std::size_t>(ovector[2 * 4 + 1] - ovector[2 * 4])},
83              std::string{
84                  str.c_str() + ovector[2 * 5],
85                  static_cast<std::size_t>(ovector[2 * 5 + 1] - ovector[2 * 5])},
86              {}};
87          if (rc > 6) {
88              res.error.replace(0, std::string::npos, str, ovector[2 * 6],
89                                ovector[2 * 6 + 1] - ovector[2 * 6]);
90          }
91      }
92
93      pcre_free(re);
94  }
```

```
95
96  void posix_regex(std::size_t num, const std::string &str, results_t &res) {
97      char err_buf[BUFSIZ];
98      regmatch_t pmatch[MAX_MATCH];
99
100     static const std::string pattern{
101         R"(^(([^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*);)"
102         R"((L|P|D|DN|R|W|LS|PS|RS|LU|PU|RU|LK|PK|RK|F);)"
103         R"((([^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*);)"
104         R"((([^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*);)"
105         R"((([^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;)*))"
106         R"((;(([^\x00-\x1F\x80-\xFF\\;]|\\\\|\\;|\';\')*))?$)"};
107
108     static int tmp_res;
109
110     regex_t preg;
111
112     // code based on https://tinyurl.com/ogdtmsl
113     if ((tmp_res = regcomp(&preg, pattern.c_str(), REG_EXTENDED)) != 0) {
114         regerror(tmp_res, &preg, err_buf, BUFSIZ);
115         std::string err = std::string("regcomp: ") + err_buf;
116         throw std::runtime_error(err);
117     }
118
119     for (auto i = 0u; i < num; i++) {
120         tmp_res = regexec(&preg, str.c_str(), MAX_MATCH, pmatch, 0);
121
122         if (tmp_res == REG_NOMATCH) {
123             regfree(&preg);
124             throw std::runtime_error("ERROR");
125         }
126
127         res = {std::string{
128                     str.c_str() + pmatch[1].rm_so,
129                     static_cast<std::size_t>(pmatch[1].rm_eo - pmatch[1].rm_so)},
130                std::string{
131                     str.c_str() + pmatch[3].rm_so,
132                     static_cast<std::size_t>(pmatch[3].rm_eo - pmatch[3].rm_so)},
133                std::string{
134                     str.c_str() + pmatch[4].rm_so,
135                     static_cast<std::size_t>(pmatch[4].rm_eo - pmatch[4].rm_so)},
136                std::string{
137                     str.c_str() + pmatch[6].rm_so,
138                     static_cast<std::size_t>(pmatch[6].rm_eo - pmatch[6].rm_so)},
139                std::string{
140                     str.c_str() + pmatch[8].rm_so,
141                     static_cast<std::size_t>(pmatch[8].rm_eo - pmatch[8].rm_so)},
```

```
142             {}};
143         if (pmatch[11].rm_so != -1) {
144             res.error.replace(0, std::string::npos, str, pmatch[11].rm_so,
145                               pmatch[11].rm_eo - pmatch[11].rm_so);
146         }
147     }
148     regfree(&preg);
149 }
150
151 void boost_regex(std::size_t num, const std::string &str, results_t &res) {
152     boost::smatch pieces;
153     static const boost::regex pattern{DEFAULT_REGEX};
154     for (auto i = 0u; i < num; i++) {
155         bool matched = boost::regex_match(str, pieces, pattern);
156         if (!(matched && pieces.size() == 7)) {
157             throw std::runtime_error("ERROR");
158         }
159         res = {pieces[1], pieces[2], pieces[3],
160               pieces[4], pieces[5], pieces[6]};
161     }
162 }
163
164 void std_regex(std::size_t num, const std::string &str, results_t &res) {
165     std::smatch pieces;
166     static const std::regex pattern{DEFAULT_REGEX};
167     for (auto i = 0u; i < num; i++) {
168         bool matched = std::regex_match(str, pieces, pattern);
169         if (!(matched && pieces.size() == 7)) {
170             throw std::runtime_error("ERROR");
171         }
172         res = {pieces[1], pieces[2], pieces[3],
173               pieces[4], pieces[5], pieces[6]};
174     }
175 }
176
177 std::size_t get_min(const measured_t &measured_values) {
178     return *measured_values.cbegin();
179 }
180
181 std::size_t get_max(const measured_t &measured_values) {
182     return *measured_values.crbegin();
183 }
184
185 std::size_t get_median(const measured_t &measured_values) {
186     std::size_t median = measured_values[measured_values.size() / 2u];
187     if (measured_values.size() % 2u == 0u) {
188         median =
```

```
189             (median + measured_values[measured_values.size() / 2u - 1u]) / 2u;
190     }
191     return median;
192 }
193
194 std::size_t get_avg(const measured_t &measured_values) {
195     return static_cast<std::size_t>(
196         0.5 +
197         std::accumulate(measured_values.cbegin(), measured_values.cend(), 0u) /
198             static_cast<double>(measured_values.size()));
199 }
200
201 void do_test(const std::string &name,
202              void (*test_func)(std::size_t, const std::string &, results_t &)) {
203     constexpr std::size_t num = 100000;
204     constexpr std::size_t measure_num = 1000;
205     std::string str = "add\\\\ress;L;client␣id;param\\;cc;value;error␣failed";
206
207     measured_t measured_values;
208     results_t res;
209
210     for (std::size_t i = 0; i < measure_num; i++) {
211         auto start = std::chrono::system_clock::now();
212         test_func(num, str, res);
213         auto end = std::chrono::system_clock::now();
214
215         if (check_results(res) == false) {
216             throw std::runtime_error("No␣match");
217         }
218         measured_values.push_back(
219             std::chrono::duration_cast<std::chrono::microseconds>(end - start)
220                 .count());
221     }
222
223     std::sort(measured_values.begin(), measured_values.end());
224     std::cout << name << "␣-␣" << COMPILER << ";" << num << ";" << measure_num
225               << ";" << *measured_values.cbegin() << ";"
226               << *measured_values.crbegin() << ";" << get_avg(measured_values)
227               << ";" << get_median(measured_values) << std::endl;
228 }
229
230 int main(void) {
231     do_test("pcre", pcre_regex);
232     do_test("boost", boost_regex);
233     do_test("posix", posix_regex);
234     do_test("std", std_regex);
235 }
```

D. Abbildungsverzeichnis

E. Literaturverzeichnis

Literatur

[1] LYNX Technik AG. *LYNX REMOTE CONTROL PROTOCOL 1.9*. Englisch. 2014.

[2] LYNX Technik AG. *LYNX Technik Launches APPolo Version 8.0 Which Includes the New Custom Control Feature*. Englisch. URL: https://www.youtube.com/watch?v=iuweXVCVvzQ (besucht am 24. 11. 2015).

[3] Murray Cumming et al. *Programming with gtkmm 3 - Multi-threaded programs - The constraints*. Englisch. 2010. URL: https://developer.gnome.org/gtkmm-tutorial/3.12/sec-the-constraints.html.en (besucht am 17. 11. 2015).

[4] Ltd. Allwinner Technology Co. *A20 - User Manual*. Englisch. 2014. URL: http://dl.linux-sunxi.org/A20/A20%20user%20manual%20v1.3%2020141010.pdf (besucht am 01. 12. 2015).

[5] E. Amberg. *Linux-Server mit Debian 7 GNU/Linux: Das umfassende Praxis-Handbuch*. mitp Professional. mitp/bhv, 2014. ISBN: 9783826682025. URL: https://books.google.at/books?id=YeE9AwAAQBAJ (besucht am 13. 10. 2015).

[6] N. Barkakati. *Red Hat Fedora Linux Secrets*. Secrets. Wiley, 2005. ISBN: 9780471774921. URL: https://books.google.at/books?id=xBANKuFJWtUC (besucht am 13. 11. 2015).

[7] U. Breymann. „Neuerungen in C++14“. In: (2015). URL: http://www.informatik-aktuell.de/entwicklung/programmiersprachen/neuerungen-in-cpp-14.html (besucht am 15. 10. 2015).

[8] TIOBE Software BV. *TIOBE Index for October 2015*. Englisch. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (besucht am 13. 10. 2015).

[9] The Qt Company. *All Classes*. Englisch. 2015. URL: http://doc.qt.io/qt-5/classes.html (besucht am 17. 11. 2015).

[10] The Qt Company. *Qt for Embedded Linux*. Englisch. 2015. URL: http://doc.qt.io/qt-5/embedded-linux.html (besucht am 17. 11. 2015).

[11] The Qt Company. *Qt Licensing*. Englisch. 2015. URL: http://doc.qt.io/qt-5/licensing.html (besucht am 17. 11. 2015).

[12] The Qt Company. *Signals & Slots*. Englisch. 2015. URL: http://doc.qt.io/qt-5/signalsandslots.html (besucht am 17. 11. 2015).

[13] P. Cozzi und C. Riccio. *OpenGL Insights*. OpenGL, OpenGL Es, and Webgl Community Experiences. Taylor & Francis, 2012. ISBN: 9781439893760. URL: https://books.google.at/books?id=CCVenzOGjpcC (besucht am 16. 11. 2015).

[14] *DC/DC-Konverter*. TRACO ELECTRONIC GmbH. 2015. URL: http://www.tracopower.com/products/tsr1.pdf (besucht am 01. 12. 2015).

[15] EarthLCD. *Specification of LCD Module EarthLCD-10.4-1024100*. Englisch. 2014. URL: https://system.netsuite.com/core/media/media.nl?id=87162&c=318770&h=6e40c7fd3cafcb53ac50&_xt=.pdf&ck=UGyofrgCAumrBcfV&vid=UGyofgEDAm_MUJ5k&cktime=131769&addrcountry=US (besucht am 29. 11. 2015).

[16] ezLCD. *2015 Pi-Raq - (Raspberry Pi based 1U Appliance)*. Englisch. 2015. URL: https://www.youtube.com/watch?v=Ol8Y_Bkscv4 (besucht am 28. 09. 2015).

[17] *Fex Guide*. Englisch. linux-sunxi.org. 2015. URL: https://linux-sunxi.org/index.php?title=Fex_Guide&oldid=15391 (besucht am 01. 12. 2015).

[18] FreeGLUT. *About*. Englisch. URL: http://freeglut.sourceforge.net/ (besucht am 24. 11. 2015).

[19] Martin Gräßlin. *Porting Qt applications to Wayland*. Englisch. 2015. URL: http://blog.martin-graesslin.com/blog/2015/07/porting-qt-applications-to-wayland/ (besucht am 18. 11. 2015).

[20] R. Grimm. *C++11 für Programmierer: Den neuen Standard effektiv einsetzen*. O'Reilly Verlag, 2013. ISBN: 9783955613921. URL: https://books.google.at/books?id=TzSvAgAAQBAJ (besucht am 13. 10. 2015).

[21] Technical Steering Group. *Tizen 3.0 Milestones*. Englisch. 2015. URL: https://source.tizen.org/release/tizen-3.0-milestones (besucht am 12. 11. 2015).

[22] S. Guha. *Computer Graphics Through OpenGL: From Theory to Experiments, Second Edition*. Taylor & Francis, 2014. ISBN: 9781482258394. URL: https://books.google.at/books?id=je3MAwAAQBAJ (besucht am 16. 11. 2015).

[23] Miguel de Icaza. *The GNOME Desktop project*. Englisch. 1997. URL: http://blog.martin-graesslin.com/blog/2015/07/porting-qt-applications-to-wayland/ (besucht am 18. 11. 2015).

[24] Earth Computer Technologies Inc. *EarthLCD Presents The Pi-Raq*. Englisch. 2015. URL: https://system.netsuite.com/core/media/media.nl?id=89880&c=318770&h=65b11be9211cfcf1b4c3&_xt=.pdf&ck=UGyofrgCAumrBcfV&vid=UGyofgEDAm_MUJ5k&cktime=131768&addrcountry=US (besucht am 29. 11. 2015).

[25] Novell Inc. *wxWidgets*. Englisch. 2011. URL: https://de.opensuse.org/WxWidgets (besucht am 18. 11. 2015).

[26] Texas Instruments. *Application Note 1085 FPD-Link PCB and Interconnect Design-In Guidelines*. Englisch. 2011. URL: http://www.ti.com/lit/an/snla002/snla002.pdf (besucht am 01. 12. 2015).

[27] Texas Instruments. *DS90CF386/DS90CF366 +3.3V LVDS Receiver 24-Bit Flat Panel Display (FPD) Link - 85MHz, +3.3V LVDS Receiver 18-Bit Flat Panel Display (FPD) Link - 85 MHz*. Englisch. 2013. URL: http://www.ti.com/lit/ds/symlink/ds90cf386.pdf (besucht am 01. 12. 2015).

[28] Lars Knoll. *Adding LGPL v3 to Qt*. Englisch. 2014. URL: https://blog.qt.io/blog/2014/08/20/adding-lgpl-v3-to-qt/ (besucht am 17. 11. 2015).

[29] J. Langr. *Testgetriebene Entwicklung mit C++: Sauberer Code. Bessere Produkte*. dpunkt.verlag GmbH, 2014. ISBN: 9783864901898. URL: https://books.google.at/books?id=TQn-nwEACAAJ (besucht am 16. 10. 2015).

[30] Michael Larabel. „A Common Input Device Library For Wayland“. In: (2013). URL: http://www.phoronix.com/scan.php?page=news_item&px=MTUxMjE (besucht am 12. 11. 2015).

[31] Thorsten Leemhuis. „Wayland vereinfacht den Grafikstack“. In: *c't Linux 2015: Server-Sicherheit, Distributionen mit Langzeit-Support* (2015), S. 86–87. URL: https://books.google.at/books?id=yDFaCgAAQBAJ (besucht am 11. 11. 2015).

[32] *Linux Kernel*. Englisch. linux-sunxi.org. 2015. URL: https://www.mikrocontroller.net/wikisoftware/index.php?title=Wellenwiderstand&oldid=89884 (besucht am 01. 12. 2015).

[33] *Linux mainlining effort*. Englisch. linux-sunxi.org. 2015. URL: https://linux-sunxi.org/index.php?title=Linux_mainlining_effort&oldid=15574 (besucht am 01. 12. 2015).

[34] Digia Finland Ltd. *LanguageBindings*. Englisch. 2015. URL: https://wiki.qt.io/index.php?title=Category:LanguageBindings&oldid=19183 (besucht am 17. 11. 2015).

[35] Digia Finland Ltd. *What is Qt?* Englisch. 2015. URL: https://wiki.qt.io/index.php?title=About_Qt&oldid=19812 (besucht am 17. 11. 2015).

[36] LYNX. *Home*. Englisch. URL: http://www.lynx-technik.com/en/home/ (besucht am 24. 11. 2015).

[37] R. MALL. *FUNDAMENTALS OF SOFTWARE ENGINEERING*. PHI Learning, 2009. ISBN: 9788120338197. URL: https://books.google.at/books?id=ppIjvQdEDhIC (besucht am 13. 11. 2015).

[38] R. MALL. *FUNDAMENTALS OF SOFTWARE ENGINEERING*. PHI Learning, 2009. ISBN: 9788120338197. URL: https://books.google.at/books?id=ppIjvQdEDhIC (besucht am 13. 11. 2015).

[39] Stéphane Marchesin. *Linux Graphics Drivers: an Introduction*. 2012. URL: http://people.freedesktop.org/~marcheu/linuxgraphicsdrivers.pdf (besucht am 15. 11. 2015).

[40] S. McManus und M. Cook. *Raspberry Pi For Dummies*. For dummies. Wiley, 2014. ISBN: 9781118905005. URL: https://books.google.at/books?id=XbMwBQAAQBAJ (besucht am 11. 11. 2015).

[41] S. Meyers. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. Englisch. O'Reilly Media, 2014. ISBN: 9781491908433. URL: https://books.google.at/books?id=rjhIBQAAQBAJ (besucht am 14. 10. 2015).

[42] S. Meyers. *Effektiv C++ programmieren: 55 Möglichkeiten, Ihre Programme und Entwürfe zu verbessern*. Programmer's Choice. Pearson Deutschland, 2011. ISBN: 9783827330789. URL: https://books.google.at/books?id=jPLzshqOJ9gC (besucht am 13. 10. 2015).

[43] Jonas Ådahl Peter Hutterer. *libinput*. Englisch. 2015. URL: http://www.freedesktop.org/wiki/Software/libinput/ (besucht am 12. 11. 2015).

[44] Jonas Ådahl Peter Hutterer. *libinput Documentation*. Englisch. 2015. URL: http://wayland.freedesktop.org/libinput/doc/1.1/ (besucht am 13. 11. 2015).

[45] The GNOME Project. *GNOME 3.10 Released!* Englisch. 2013. URL: https://www.gnome.org/news/2013/09/gnome-3-10-released/ (besucht am 12. 11. 2015).

[46] The GNOME Project. *gtkmm - Introduction*. Englisch. 2015. URL: https://developer.gnome.org/gtkmm-tutorial/3.12/sec-gtkmm.html.en (besucht am 17. 11. 2015).

[47] The GNOME Project. *libxml++: libxml++ Reference Manual*. Englisch. 2014. URL: https://developer.gnome.org/libxml++/2.36/ (besucht am 28. 09. 2015).

[48] The GNOME Project. *Porting GTK+ applications to Wayland*. Englisch. 2015. URL: https://wiki.gnome.org/Initiatives/Wayland/Applications/Porting (besucht am 17. 11. 2015).

[49] The GNOME Project. *Using GTK+ with Broadway*. Englisch. 2015. URL: https://developer.gnome.org/gtk3/stable/gtk-broadway.html (besucht am 17. 11. 2015).

[50] Yocto Project. *Matchbox*. 2013. URL: https://www.yoctoproject.org/tools-resources/projects/matchbox (besucht am 19. 11. 2015).

[51] K. Riedling. *Technisches Programmieren in C++*. 2014. URL: http://www2.isas.tuwien.ac.at/riedling/cpp/C++-Skriptum_Web.pdf (besucht am 14. 10. 2015).

[52] Patrick Schnabel. *Stützkondensator*. 2009. URL: http://www.elektronik-tipps.de/archiv/stuetzkondensator/ (besucht am 01. 12. 2015).

[53] A. Semashev. *Sink backends*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_55_0/libs/log/doc/html/index.html#log.moti (besucht am 13. 10. 2015).

[54] Simos. „Baseboard for the Cubieboard". Englisch. In: (2013). URL: http://cubieboard.org/2013/01/14/daughterboard-for-the-cubieboard/ (besucht am 01. 12. 2015).

[55] *SPECIFICATION TFT1280120-1-E Rev.1.1*. Englisch. Truly Semiconductors Ltd. 2010. URL: http://www.naxtechnologies.com/datasheets/TFT/TFT1280120-1-E(11D)v1.1.pdf (besucht am 01. 12. 2015).

[56] B. Stroustrup. „An Overview of the C++ Programming Language". Englisch. In: CRC Press, 1999. ISBN: 0-8493-3135-8. URL: http://www.stroustrup.com/crc.pdf (besucht am 13. 10. 2015).

[57] B. Stroustrup. *The C++ Programming Language*. Englisch. Pearson Education, 2013. ISBN: 9780133522853. URL: https://books.google.at/books?id=PSUNAAAAQBAJ (besucht am 13. 10. 2015).

[58] Bjarne Stroustrup. *C++11 - the new ISO C++ standard*. Englisch. 2015. URL: http://www.stroustrup.com/C++11FAQ.html#what-libraries (besucht am 28. 09. 2015).

[59] H. Sutter. „Build 2011 - Writing modern C++ code: how C++ has evolved over the years." In: Microsoft. 2011. URL: http://video.ch9.ms/build/2011/slides/TOOL-835T_Sutter.pptx (besucht am 12. 10. 2015).

[60] Debian Wiki team. *WindowManager*. 2012. URL: https://wiki.debian.org/WindowManager (besucht am 19. 11. 2015).

[61] The GTK+ Team. *Language Bindings*. Englisch. 2015. URL: http://www.gtk.org/language-bindings.php (besucht am 17. 11. 2015).

[62] The GTK+ Team. *What is GTK+, and how can I use it?* Englisch. 2015. URL: http://www.gtk.org/ (besucht am 17. 11. 2015).

[63] Ian Main Tony Gale. *GTK v1.2 Tutorial - Introduction*. Englisch. 2000. URL: http://www.gtk.org/tutorial1.2/gtk_tut-1.html (besucht am 17. 11. 2015).

[64] Daniel Veillard. *The XML C parser and toolkit of Gnome - libxml*. Englisch. 2015. URL: http://www.xmlsoft.org/ (besucht am 01. 11. 2015).

[65] J. Walter. *Mikrocomputertechnik mit der 8051-Controller-Familie: Hardware, Assembler, C*. Englisch. Springer Berlin Heidelberg, 2008. ISBN: 9783540694656. URL: https://books.google.at/books?id=bAYjBAAAQBAJ (besucht am 13. 10. 2015).

[66] Martin Wührer. *MTPSW-Lynx - Bedienungsanleitung*. 2015.

[67] www.mikrocontroller.net. *Wellenwiderstand*. 2015. URL: https://www.mikrocontroller.net/wikisoftware/index.php?title=Wellenwiderstand&oldid=89884 (besucht am 01. 12. 2015).

[68] wxWidgets. *Overview*. Englisch. 2015. URL: https://www.wxwidgets.org/about/ (besucht am 19. 11. 2015).

[69] wxWidgets. *wxWidgets*. Englisch. 2015. URL: https://www.wxwidgets.org/ (besucht am 19. 11. 2015).

[70] wxWidgets. *wxWidgets-3.0.2*. Englisch. 2014. URL: ftp://ftp.wxwidgets.org/pub/3.0.0/wxWidgets-3.0.0.zip (besucht am 18. 11. 2015).

Weiterführende Links

[L1] Ltd Allwinner Technology Co. Englisch. 2013. URL: http://dl.cubieboard.org/model/cubieboard2/Hardware/A20%20Datasheet%20V1.1%2020130321.pdf (besucht am 13. 10. 2015).

[L2] SD Association. *Bus Speed (Default Speed/ High Speed/ UHS)*. Englisch. URL: https://www.sdcard.org/developers/overview/bus_speed/ (besucht am 12. 10. 2015).

[L3] Daniel Azuma. *GLOW*. Englisch. 2000. URL: http://glow.sourceforge.net/ (besucht am 17. 11. 2015).

[L4] BananaPi.com. *What's the BPi-M2?* Englisch. 2014. URL: http://www.bananapi.com/index.php/component/content/article?layout=edit&id=73 (besucht am 01. 12. 2015).

[L5] BananaPi.org. *Banana Pi*. Englisch. 2014. URL: http://www.bananapi.org/p/product.html (besucht am 01. 12. 2015).

[L6] bob. *Define: EIA-310*. Englisch. 2007. URL: https://www.server-racks.com/eia-310.html (besucht am 29. 11. 2015).

[L7] M. Champlon. *Turtle*. Englisch. 2015. URL: http://turtle.sourceforge.net/ (besucht am 14. 10. 2015).

[L8] Matthias Clasen. *3.18 Release Notes Items*. Englisch. URL: https://wiki.gnome.org/ThreePointSeventeen/ReleaseNotes (besucht am 12. 11. 2015).

[L9] cmake.org. *About CMake*. Englisch. 2008. URL: https://cmake.org/overview/ (besucht am 24. 11. 2015).

[L10] Compiz.org. *Setup*. Englisch. 2010. URL: http://wiki.compiz.org/Setup (besucht am 19. 11. 2015).

[L11] Compiz.org. *What is Compiz ?* Englisch. URL: http://www.compiz.org/ (besucht am 19. 11. 2015).

[L12] cppreference.com. *Algorithms library*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/algorithm&oldid=70996 (besucht am 14. 10. 2015).

[L13] cppreference.com. *C++ compiler support*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/compiler_support&oldid=80748 (besucht am 15. 10. 2015).

[L14] cppreference.com. *Containers library*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/container&oldid=78132 (besucht am 14. 10. 2015).

[L15] cppreference.com. *Date and time utilities*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/thread&oldid=78644 (besucht am 14. 10. 2015).

[L16] cppreference.com. *Dynamic memory management*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/memory&oldid=71013 (besucht am 14. 10. 2015).

[L17] cppreference.com. *Input/output library*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/io&oldid=77036 (besucht am 14. 10. 2015).

[L18] cppreference.com. *Regular expressions library*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/regex&oldid=73404 (besucht am 14. 10. 2015).

[L19] cppreference.com. *Strings library*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/string&oldid=70998 (besucht am 14. 10. 2015).

[L20] cppreference.com. *Thread support library*. Englisch. 2015. URL: http://en.cppreference.com/mwiki/index.php?title=cpp/thread&oldid=78644 (besucht am 14. 10. 2015).

[L21] CubieBoard. 2013. URL: http://cubieboard.org/2013/06/19/cubieboard2-is-here/ (besucht am 16. 11. 2015).

[L22] Beman Dawes. *Filesystem Library Proposal (Revision 4)*. Englisch. 2013. URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3505.html (besucht am 12. 12. 2015).

[L23] docs.cubieboard.org. *Cubieboard Open-Source Main-Boards*. Englisch. 2015. URL: http://docs.cubieboard.org/products/start?rev=1426128902 (besucht am 01. 12. 2015).

[L24] Raspberry PI Foundation. *Downloads*. Englisch. 2015. URL: https://www.raspberrypi.org/downloads/ (besucht am 24. 11. 2015).

[L25] Raspberry PI Foundation. *Raspberry Pi Model Specifications*. Englisch. 2015. URL: https://www.raspberrypi.org/documentation/hardware/raspberrypi/models/specs.md (besucht am 01. 12. 2015).

[L26] X.Org Foundation. *X11 - Release 7.7*. Englisch. URL: http://www.x.org/wiki/Releases/7.7/ (besucht am 12. 11. 2015).

[L27] Inc. Free Software Foundation. *GNU Autoconf*. Englisch. 2008. URL: https://www.gnu.org/software/autoconf/manual/autoconf-2.63/autoconf.txt (besucht am 24. 11. 2015).

[L28] Freedesktop.org. *Wayland Architecture*. Englisch. URL: http://wayland.freedesktop.org/architecture.html (besucht am 12. 11. 2015).

[L29] Khronos Group. 2015. URL: https://www.khronos.org/assets/uploads/developers/library/2015-gdc/Khronos-Vulkan-GDC_Mar15.pdf (besucht am 17. 11. 2015).

[L30] Sebastian Grüner. „KWin-Wayland wird Libinput und Logind verwenden“. In: (2014). URL: http://www.golem.de/news/kde-plasma-kwin-wayland-wird-libinput-und-logind-verwenden-1410-109978.html (besucht am 12. 11. 2015).

[L31] Sebastian Grüner. „Patch für 20 Jahre alte X-Server-Lücke“. In: (2013). URL: http://www.golem.de/news/x-org-patch-fuer-20-jahre-alte-x-server-luecke-1310-102069.html (besucht am 11. 11. 2015).

[L32] gtkmm.org. *License*. Englisch. 2015. URL: http://www.gtkmm.org/en/license.html (besucht am 17. 11. 2015).

[L33] Ltd. Hardkernel co. *ODROID-C1*. 2015. URL: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141578608433&tab_idx=1 (besucht am 12. 10. 2015).

[L34] Philip Hazel. *README file for PCRE2 (Perl-compatible regular expression library)*. Englisch. 2013. URL: http://www.pcre.org/readme.txt (besucht am 01. 11. 2015).

[L35] F. Holzbauer. *Seagate Enterprise Capacity 3.5 HDD v4 6000GB: Riesenspeicher mit High-speed*. 2014. URL: http://www.chip.de/artikel/Seagate-Enterprise-Capacity-3.5-HDD-v4-6000GB-SATA-Festplatte-3-5-Zoll-Test_69927228.html (besucht am 12. 10. 2015).

[L36] Free Software Foundation Inc. *GNU General Public License, version 3*. Englisch. 2007. URL: https://www.gnu.org/licenses/gpl-3.0.en.html (besucht am 17. 11. 2015).

[L37] Free Software Foundation Inc. *GNU Lesser General Public License, version 2.1*. Englisch. 1999. URL: https://www.gnu.org/licenses/lgpl-2.1.en.html (besucht am 17. 11. 2015).

[L38] Free Software Foundation Inc. *GNU Lesser General Public License, version 3*. Englisch. 2007. URL: https://www.gnu.org/licenses/lgpl-3.0.en.html (besucht am 17. 11. 2015).

[L39] MIPI Alliance Inc. *Display Interface Specifications*. Englisch. 2015. URL: http://mipi.org/specifications/display-interface (besucht am 29. 11. 2015).

[L40] Kristian Kißling. „Mutter: Windowmanager für Gnomes Zukunft“. In: (2009). URL: http://www.linux-magazin.de/NEWS/Mutter-Windowmanager-fuer-Gnomes-Zukunft (besucht am 19. 11. 2015).

[L41] C. Kohlhoff. *Daytime.1 - A synchronous TCP daytime client*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio/tutorial/tutdaytime1.html (besucht am 14. 10. 2015).

[L42] C. Kohlhoff. *Future Example*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio/example/cpp11/futures/daytime_client.cpp (besucht am 14. 10. 2015).

[L43] C. Kohlhoff. *Signal Handling*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio/overview/signals.html (besucht am 14. 10. 2015).

[L44] C. Kohlhoff. *Socket Iostreams*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio/overview/networking/iostreams.html (besucht am 14. 10. 2015).

[L45] C. Kohlhoff. *SSL*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio/overview/ssl.html (besucht am 14. 10. 2015).

[L46] C. Kohlhoff. *steady_timer*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio/reference/steady_timer.html (besucht am 14. 10. 2015).

[L47] Christopher Kohlhoff. *Networking Library Proposal (Revision 4)*. Englisch. 2015. URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4370.html (besucht am 12. 12. 2015).

[L48] V. Lextrait. *The Programming Languages Beacon*. 2015. URL: http://www.lextrait.com/vincent/implementations.html (besucht am 12. 10. 2015).

[L49] ARM Ltd. Englisch. 2015. URL: http://www.arm.com/products/processors/classic/arm11/arm1176.php (besucht am 13. 10. 2015).

[L50] ARM Ltd. Englisch. 2015. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0464f/index.html (besucht am 13. 10. 2015).

[L51] Jolla Ltd. *Sailfish OS*. 2015. URL: https://sailfishos.org/ (besucht am 12. 11. 2015).

[L52] John Maddock. *Boost.Regex*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_54_0/libs/regex/doc/html/ (besucht am 01. 11. 2015).

[L53] Mesa3D.org. 2012. URL: http://www.mesa3d.org/faq.html (besucht am 16. 11. 2015).

[L54] Igor Pečovnik. *armbian - linux for ARM development boards*. Englisch. 2015. URL: http://www.armbian.com/ (besucht am 24. 11. 2015).

[L55] Ainsley Pereira. *libsigc++ - Chapter 1. Introduction*. Englisch. 2014. URL: https://developer.gnome.org/libsigc++-tutorial/stable/ch01.html (besucht am 21. 11. 2015).

[L56] Steve Perkins. *wxJava*. Englisch. 2015. URL: http://sourceforge.net/projects/wxjava/ (besucht am 13. 12. 2015).

[L57] Thomas Petazzoni. *Device Tree for Dummies!* Englisch. 2013. URL: https://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf (besucht am 01. 12. 2015).

[L58] *Powering the Cubie with the Baseboard*. Englisch. 2013. URL: http://www.iotllc.com/joomla/index.php/forum/baseboard/2-powering-the-cubie-with-the-baseboard (besucht am 01. 12. 2015).

[L59] The GNOME Project. *Full Wayland support in GTK+*. Englisch. 2015. URL: https://wiki.gnome.org/Initiatives/Wayland/GTK+ (besucht am 17. 11. 2015).

[L60] The GNOME Project. *GNOME Applications under Wayland*. Englisch. 2015. URL: https://wiki.gnome.org/Initiatives/Wayland/Applications (besucht am 17. 11. 2015).

[L61] Andreas Proschofsky. „Fedora wagt den Wechsel auf Wayland“. In: (2015). URL: https://derstandard.at/userprofil/527290 (besucht am 11. 11. 2015).

[L62] Paul Rademacher. *GLUI*. Englisch. 2000. URL: http://glui.sourceforge.net/ (besucht am 17. 11. 2015).

[L63] International Rectifier. *IRF7313PbF*. Englisch. 2004. URL: http://www.irf.com/product-info/datasheets/data/irf7313pbf.pdf (besucht am 01. 12. 2015).

[L64] G. Rozental. *Boost Test Library*. Englisch. 2007. URL: http://www.boost.org/doc/libs/1_54_0/libs/test/doc/html/index.html (besucht am 14. 10. 2015).

[L65] Patrick Schnabel. 2015. URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2002051.htm (besucht am 16. 11. 2015).

[L66] Patrick Schnabel. 2015. URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2002071.htm (besucht am 16. 11. 2015).

[L67] A. Sebayang. „Samsung 950 Pro Schnelle NVMe-SSD im M.2-Format kommt für den Massenmarkt“. In: (2015). URL: http://www.golem.de/news/samsung-950-pro-schnelle-nvme-ssd-im-m-2-format-kommt-fuer-den-massenmarkt-1509-116425.html (besucht am 12. 10. 2015).

[L68] A. Semashev. *Sink backends*. Englisch. 2013. URL: http://www.boost.org/doc/libs/1_55_0/libs/log/doc/html/log/detailed/sink_backends.html#log.detailed.sink_backends.text_file (besucht am 13. 10. 2015).

[L69] Uwe Sperling. „HDMI So funktionierts !“ In: (2010). URL: http://www.spatz-tech.com/shop_pdfs/HDMI_funktioniert.pdf (besucht am 01. 12. 2015).

[L70] International Organization for Standardization. *ISO/IEC 14882:2014*. Englisch. 2014. URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64029 (besucht am 14. 10. 2015).

[L71] N.T. Stewart. *GLT*. Englisch. 2010. URL: http://www.nigels.com/glt/download.html (besucht am 17. 11. 2015).

[L72] Debian Wiki team. *Debian Jessie*. Englisch. 2015. URL: https://wiki.debian.org/DebianJessie (besucht am 24. 11. 2015).

[L73] Debian Wiki team. *Debian Wheezy*. Englisch. 2015. URL: https://wiki.debian.org/DebianWheezy (besucht am 24. 11. 2015).

[L74] Paul D Turner & The CEGUI Development Team. *CEGUI*. Englisch. 2013. URL: http://static.cegui.org.uk/docs/0.8.4/index.html (besucht am 17. 11. 2015).

[L75] Paul D Turner & The CEGUI Development Team. *CEGUI*. Englisch. 2013. URL: http://static.cegui.org.uk/docs/0.8.4/rendering_tutorial.html (besucht am 17. 11. 2015).

[L76] Paul D Turner & The CEGUI Development Team. *CEGUI*. Englisch. 2013. URL: http://static.cegui.org.uk/docs/0.8.4/licensing.html (besucht am 17. 11. 2015).

[L77] The GTK+ Team. *GTK+ Download*. Englisch. 2015. URL: http://www.gtk.org/download/ (besucht am 17. 11. 2015).

[L78] Jörg Thoma. „Debian kehrt zurück zu Gnome“. In: (2014). URL: http://www.golem.de/news/linux-desktops-debian-kehrt-zurueck-zu-gnome-1409-109468.html (besucht am 12. 11. 2015).

[L79] Jörg Thoma. „Samsung 950 Pro Schnelle NVMe-SSD im M.2-Format kommt für den Massenmarkt“. In: (2012). URL: http://www.golem.de/news/freie-grafiksysteme-directfb-1-6-wird-stabil-1206-92771.html (besucht am 15. 11. 2015).

[L80] Linus Torvalds. *Linux 3.4 released*. Englisch. 2012. URL: http://lkml.iu.edu/hypermail/linux/kernel/1205.2/02545.html (besucht am 01. 12. 2015).

[L81] ubuntuusers.de. *XServer*. 2015. URL: https://wiki.ubuntuusers.de/XServer?rev=835228 (besucht am 11. 11. 2015).

[L82] CEGUI Wiki. *PyCEGUI*. Englisch. 2011. URL: http://cegui.org.uk/wiki/index.php?title=PyCEGUI&oldid=4450 (besucht am 13. 12. 2015).

[L83] Embedded Linux Wiki. 2010. URL: http://elinux.org/index.php?title=DirectFB&oldid=21752 (besucht am 15. 11. 2015).

[L84] KDE UserBase Wiki. *KWin*. 2015. URL: https://userbase.kde.org/index.php?title=KWin/de&oldid=350133 (besucht am 19. 11. 2015).

[L85] Wikipedia. *Concepts (C++)*. Englisch. 2015. URL: https://en.wikipedia.org/w/index.php?title=Concepts_(C++)&oldid=685987816 (besucht am 20. 10. 2015).

[L86] Graeme W. Wilford. (Wilf.) *REGEX(3)*. Englisch. 2013. URL: http://man7.org/linux/man-pages/man3/regcomp.3.html (besucht am 01. 11. 2015).

[L87] wxWiki. *Bindings*. Englisch. 2013. URL: https://wiki.wxwidgets.org/index.php?title=Bindings&oldid=9952 (besucht am 13. 12. 2015).