MPD-Client

Teil der Software Engineering II Studienarbeit WS 2011/2012, Inf 3

Christopher Pahl,
Christoph Piechula,
Eduard Schneider,
und Marc Tigges

2. Januar 2012

Inhaltsverzeichnis

1 Einleitung

Ziel dieser Studienarbeit ist die vollständige Bearbeitung einer vorgegebenen Aufgabenstellung nach einem selbst gewählten Vorgehensmodell. Die Aufgabenstellung schreibt vor, sich in einer Gruppe zusammen zu finden und gemeinsam ein Software-Projekt zu bearbeiten und dabei strukturiert und professionell vorzugehen.

1.1 Rahmenbedingungen

- Persistente Datenspeicherung
  - Datei oder Datenbank (wenn schon bekannt)
- Netzwerk-Programmierung
  - Eine verteilte Architektur (z.B.: Client/Server)
- GUI
  - Swing
  - Web-basiert
  - Gtk+ (durch Nachfrage)

1.2 Prozess-Anforderungen

- Dokumentation aller Phasen(Analyse bis Testen)
- Auswahl eines konkreten Prozessmodells
  - Z.B. sd&m, M3, RUP, Agile Methoden ...
  - Begründung (warum dieser Prozess passt zu Ihrem System)
- Erstellung der Dokumente und UML-Diagramme
  - Visio

  - UML Werkzeuge (freie Wahl)
- Fertige Implementierung
  - Es kann mehr spezifiziert sein als implementiert
- Spezifikation von Testszenarien
  - und der Beleg der erfolgreichen Ausführung
- Lauffähiges System

1.3 Mögliche Themen

- CRM Systeme
  - Bibliothek
  - Musikshop
  - ...
- Kommunikationssysteme
- Chat-Variationen (Skype, etc.)
- File-Verwaltungs-Systeme (eigener Cloud-Dienst)
- ...

Portale

- Mitfahrgelegenheit
- Dating-Agentur ;)
- ...

[1]

Diese Arbeit ist wichtig, um den Studenten zu zeigen, wie man in einem Team zusammenarbeitet und nach Software-Engineering-Methoden qualitativ hochwertige Software erstellt. Es geht im Folgenden um einen Music-Player-Daemon-Client (Näheres bitte der Definition entnehmen). Dieses Thema wird behandelt, da es alle Rahmenbedingungen abdeckt und im Interesse der Autoren liegt. Die Besonderheit liegt darin, dass sich diese Software nach Fertigstellung auch wirklich anwenden lässt. Ziel ist die Erweiterung der Fähigkeitn im Bereich der Software Engineering sowie das Erlernen von Methoden für wissenschaftliches Arbeiten.

[1] Folie Anforderungen, Autor Prof. Dr. Philipp Schaible, WS 2011/2012, Inf 3

2 Wasserfallmodell mit Rücksprung

2.1 Definition

Das Wasserfallmodell ist ein lineares (nicht iteratives) vorgehensmodell in der Softwareentwicklung, bei dem der Sotwareentwicklungsprozess in Phasen organisiert wird. Dabei gehen die Phasenergebnisse wie bei einem Wasserfall immmer als bindende Vorgaben für die nächsttiefere Phase ein.

Im Wasserfallmodell hat jede Phase vordefinierte Start- und Endpunkte mit eindeutig definierten Ergebnissen. In Meilensteinsitzungen am jeweiligen Phasenende werden die Ergenisdokumente verabschiedet. Zu den wichtigsten Dokumenten zählen dabei das Lastenheft sowie das Pflichenheft. In der betrieblichen Praxis gibt es viele Varianten des reinen Modells. Es ist aber das traditionell am weitesten verbreitete Vorgehensmodell.

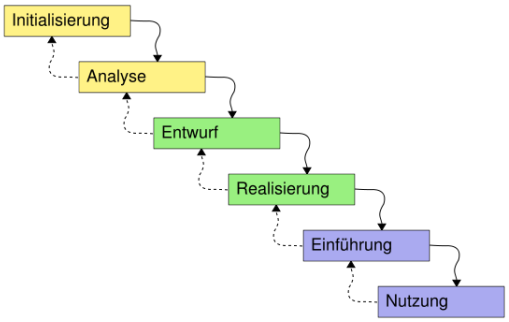
Der Name Ẅasserfall̈kommt von der häufig gewählten grafischen Darstellung der fünf bis sechs als Kaskade angeordneten Phasen. Ein erweitertes Wasserfallmodell mit Rücksprungmöglichkeiten (gestrichelt).

Erweiterungen des einfachen Modells (Wasserfallmodell mit Rücksprung) führen iterative Aspekte ein und erlauben ein schrittweises Äufwärtslaufen̈der Kaskade, sofern in der aktuellen Phase etwas schieflaufen sollte, um den Fehler auf der nächsthöheren Stufe beheben zu können. [1]

[2]

[1] Zitat aus: http://de.wikipedia.org/wiki/Wasserfallmodell

[2] Wasserfallmodell mit Rücksprung,
Bild-Quelle: http://upload.wikimedia.org/wikipedia/commons/thumb/e/e5/Wasserfallmodell.svg/567px-Wasserfallmodell.svg.png



2.2 Warum dieses Modell?

Wir haben uns für das Wasserfallmodell mit Rücksprung entschieden, weil dieses Modell alle Phasen der Entwicklung klar abgrenzt und sich optimal auf einen professionellen Softwareentwicklungsvorgang abbilden lässt. Dieses Modell ermöglicht eine klare Planung und Kontrolle unseres Softwareprojekts, da die Anforderungen stets die gleichen bleiben und der Umfang einigermaßen gut abschätzbar ist. Für die erweiterte Version dieses Modells, nämlich mit Rücksprung, haben wir uns entschieden, um ein paar Nachteile dieses Modells auszuhebeln. Beispielsweise sind die klar voneinander abgegrenzten Phasen in der Realität oft nicht umsetzbar. Des weiteren sind wir somit flexibler gegenüber Änderungen.

2.3 Tatsächliche Umsetzung

Im Laufe der Entwicklung mussten wir feststellen, dass das Wasserfallmodell als Vorgehensweise für unser Projekt doch nicht so gut geeignet war wie wir erwarteten. Da das Projekt eine viel zu komplexe Infrastruktur besitzt, die wir vorher unmöglich abschätzen konnten, war es unmöglich zu planen, welcher Arbeitsaufwand nötig ist. Somit konnten keine brauchbaren Entwürfe der Software gemacht werden, da man sich manche Funktionen des MPD völlig anders vorgestellt hatte, als sie in Wirklichkeit funktionierten. Daher mussten wir, nachdem das Lasten- und Pflichtenheft fertiggestellt waren (welche unabhängig von der Infrastruktur sind)), auf agile Entwicklungsmethoden umschwänken. Aufgrund der hohen Komplexität der MPD Libraries war es uns nicht möglich sich einen Überblick über interne Abläufe zu verschaffen so mussten wir parallel zu den eigentlichen Planungen erste Testanwendungen schrieben um sich mit der Materie vertraut zu machen. Als Beispiel sei hier der Testclient genannt, der letzlich auch die Grundlage für die heutige Architektur darstellt, bzw. die Grundlage aus der sie entstanden ist.

(TODO: Spiralmodell?)

3 Richtlinien

3.1 Programmierrichtlinien

- Tabstop = 4 Leerzeichen.
- Keine nichtkonstanten globalen Variablen.
- Sinnvolle Variablenbenennung, “lowercase“.
- Klassenmethoden (static) nur in Ausnahmefällen bzw. nur mit guten Gründen.
- Valgrind darf keine Laufzeitfehler bringen, die nicht von Gtk oder anderen Bibliotheken stammen.
- ,,camelcase” bei Objektnamen, C-Style (function_name) bei Funktionsnamen - Präzise Namen.
- Modulare Gestaltung.
- Durchgehendes Model-View-Controller Pattern.
- Code-Sauberkeit ist wichtiger als Code Performance.
- “make“ sollte keine Warnungen ausgeben, die man leicht umgehen könnte. (-Wall und -Wextra -std=c++0x wird genutzt)
- “make test“ soll vollständig durchlaufen.
- Allman-Stil.

3.1.1 Begründung

Einhaltung dieser Programmierrichtlinen sorgen für ein einfaches, übersichtliches und einheitliches Arbeiten. Jeder sollte sich ohne größere Umstände in den Code eines anderen einlesen können. Dies gewährleistet eine hohe Wartbarkeit der Programm-Codes

und beugt außerdem Fehlern vor. Das Programm ist leicht erweiterbar ohne große Anpassungen vornehmen zu müssen.

3.2 Toolauswahl

- CMake (Buildsystem)
- g++ (C++ Compiler)
- Valgrind (Memorydebugger)
- git (Hosting auf Github) [1]
- Glade (GUI-Designer)
- doxygen (Interne Dokumentationsgenerierung)
- Devhelp (Dokumentationsbrowser)

3.2.1 Begründung

CMake wurde ausgewählt da es eine solide und vor allem einfache Syntax bietet, und zudem leicht anpasspar ist. Git dient zur Versionsverwaltung. Github wurde dabei als Hostingplattform ausgewählt, da das Hosting dort für Opensourceprojekte frei ist. Glade bietet eine solide Trennung von der grafischen Oberfläche zum Kontrollkern des Programms außerdem kann mit Glade sehr einfach eine grafische Oberfläche erstellt werden. Da man sich in C++ im Gegensatz zu Java um sehr viele Sache selbst kümmern muss wurde Valgrind als Memorydebugger ausgewählt, um das Aufspüren von Fehlern zu vereinfachen.

3.3 Bibliotheken

- gtkmm3 (C++ Wrapper für Gtk+) [2]
- libmpdclient (Lowlevel MPD-Bibliothek für C) [3]
- libxml2 (XML Parser Library für C) [4]

[1] https://github.com/studentkittens/Freya
[2] http://www.gtkmm.org/de/index.html
[3] http://www.musicpd.org/doc/libmpdclient/files.html
[4] http://xmlsoft.org/index.html

- libnotify (Anzeige von Desktopnachrichten) [5]
- Avahi-glib (Interface zum Avahidaemon) [6]

3.3.1 Begründung

C++ wurde aufgrund persönlicher Interessen der Autoren gewählt. Außerdem gibt es für Java nur wenige oder sehr alte Bibliotheken für dieses Projekt. Gtkmm3 bietet ein dynamisches Layout und ist leichtgewichtiger als Qt, außerdem ist es einfacher in der Handhabung und lässt sich auf den meisten Desktopumgebungen besser integrieren. Swing ist aus Sicht der Authoren nicht geeignet. Libmpdclient ist eine eigene lowlevel C-Bibliothek, sie ist unabhängig von eigentlichen MPD-Server. Libxml2 liefert einen standardisierten Xml Parser und ist sehr leichtgewichtig, sowie auf einer großen Zahl von Systemen vorhanden. Libnotify liefert Benachrichtigungen über interne Events und ist auf den meisten Linux Distributionen verbreitet. Avahi-glib ist ein Interface für den Avahidaemon, der optional ist. Avahi dient als Server-Borwser, kann allerdings nur MPD Server finden, die sich per Zeroconf am Avahidaemon registriert haben.
Primäre Entwicklerplattform ist ein GNU/Linux-System nach Wahl.

[5] http://developer.gnome.org/libnotify/
[6] http://avahi.org/wiki/WikiStart#WhatisAvahi

4 Definition

Der MPD ist eine Client/Server-Architektur, in der die Clients und Server (MPD ist der Server) über ein Netzwerk interagieren. MPD ist also nur die Hälfte der Gleichung. Zur Nutzung von MPD, muss ein MPD-Client (auch bekannt als MPD-Schnittstelle) installiert werden. Als Netzwerkprotokoll wird das MPD eigene Protokoll verwendet. [1]

4.1 Definition des MPD

Der Music Player Daemon (kurz MPD) ist ein Unix-Systemdienst, der das Abspielen von Musik auf einem Computer ermöglicht. Er unterscheidet sich von gewöhnlichen Musik-Abspielprogrammen dadurch, dass eine strikte Trennung von Benutzeroberfläche und Programmkern vorliegt. Dadurch ist die grafische Benutzeroberfläche auswechselbar und auch eine Fernsteuerung des Programms über das Netzwerk möglich. Die Schnittstelle zwischen Client und Server ist dabei offen dokumentiert und der Music Player Daemon selbst freie und quelloffene Software.

Der MPD kann wegen seines geringen Ressourcenverbrauchs nicht nur auf Standartrechnern sondern auch auf einem abgespeckten Netzwerkgerät mit Audioausgang betrieben werden und von allen Computern oder auch Mobiltelefonen / PDAs im Netzwerk ferngesteuert werden.

Es ist auch möglich den Daemon und den Client zur Fernsteuerung lokal auf dem gleichen Rechner zu betreiben, er fungiert dann als normaler Medienspieler, der jedoch von einer Vielzahl unterschiedlicher Clients angesteuert werden kann, die sich in Oberflächengestaltung und Zusatzfunktionen unterscheiden. Mittlerweile existierten auch zahlreiche Clients, die eine Webschnittstelle bereitstellen.

[1] http://www.musicpd.org/doc/protocol/index.html

Der MPD spielt die Audioformate Ogg Vorbis, FLAC, OggFLAC, MP2, MP3, MP4/AAC, MOD, Musepack und wave ab. Zudem können FLAC-, OggFLAC-, MP3- und OggVorbis-HTTP-Streams abgespielt werden. Die Schnittstelle kann auch ohne manuelle Konfigruation mit der Zeroconf-Technik angesteuert werden. Des Weiteren wird Replay Gain, Gapless Playback, Crossfading und das Einlesen von Metadaten aus ID3-Tags, Vorbis comments oder der MP4-Metadatenstruktur unterstützt. [2]

[2]Zitat aus: http://de.wikipedia.org/wiki/Music_Player_Daemon

4.1.1 Der MPD kann:

- Musik abspielen
- Musik kontrollieren und in Warteschlangen reihen
- Musik Dateien dekodieren
- HTTP-Streaming
  - Eine HTTP-URL kann zur Warteschlange hinzugefügt oder direkt abgespielt werden.

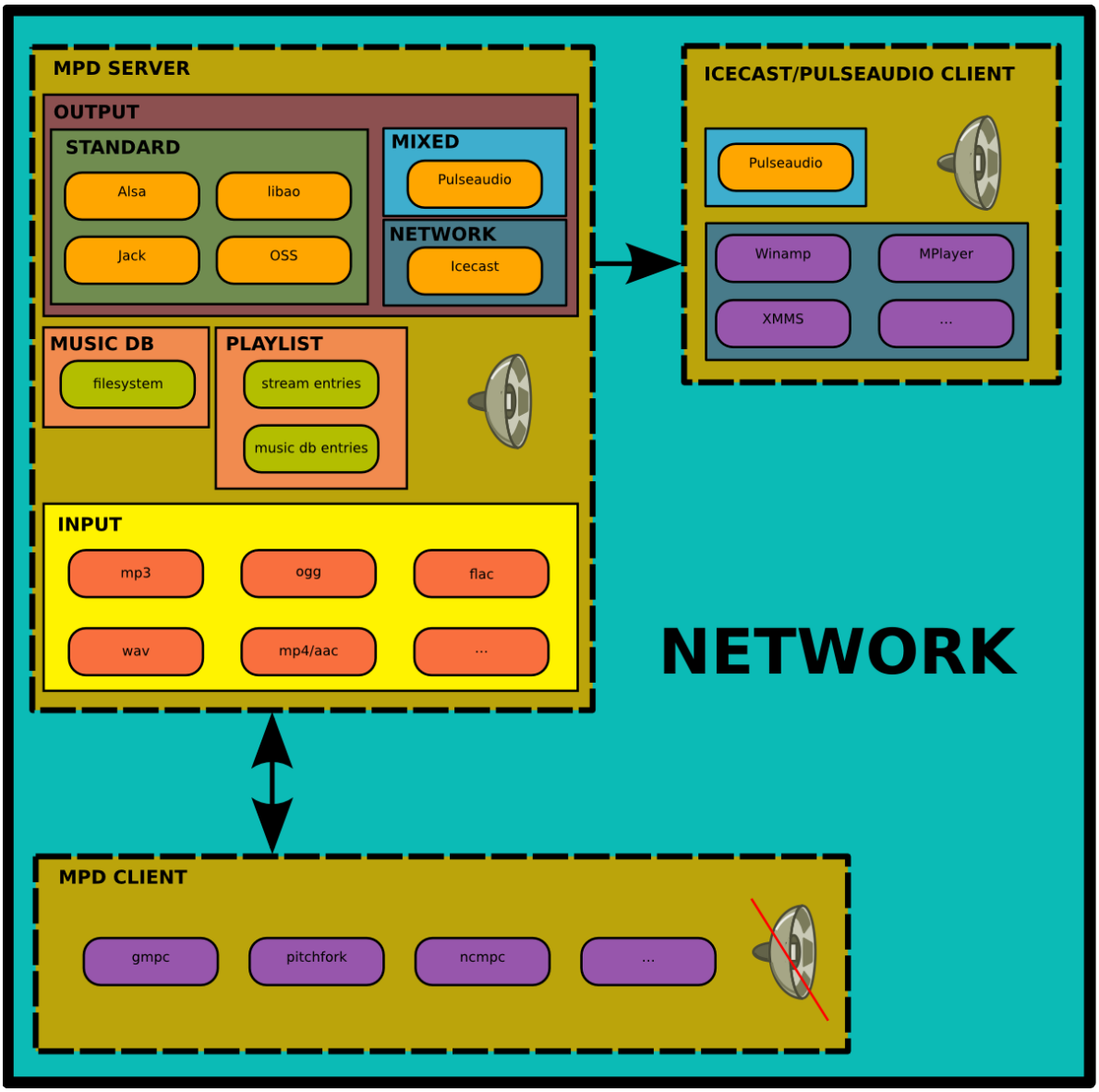
4.1.2 Der MPD kann nicht:

- Album-Cover speichern
- Funktionen eines Equalizers bereitstellen
- Musik Taggen (Informationen aus dem Web suchen)
- Text für Playlist-Dateien parsen
- Statistische Auswertungen machen
- Musik visualisieren
- Funktionen eines Remote-File-Servers bereitstellen
- Funktionen eines Video-Servers bereitstellen

4.2 Definiton des MPD-Client

Der Music Player Daemon Client ist nun die Schnittstelle zum MPD. Über diesen Client kann der MPD gesteuert werden. Es gibt viele verschiedene Clients mit unterschiedlichsten Funktionen, da der Client nicht auf den Funktionsumfang des MPD begrenzt ist. Das heißt im Klartext, dass der Client zwar nur die Funktionen über das Netzwerk steuern kann, die vom MPD implementiert sind aber nicht, dass er deshalb auch keine lokalen Dienste bzw. Funktionen anwenden kann. So kann ein Client beispielsweise alle Funktionen lokal implementieren, die unter dem Punkt ̈3.1.2 Der MPD kann nicht:ërwähnt wurden.

4.3 Grafische Übersicht



[3] Der MPD-Server bekommt als Input mp3, ogg, flac, wav, mp4/aac,... Musik-Dateien die entweder in einer Musik-Datenbank oder in Playlisten gespeichert sind. Der Standardoutput des MPD ist Alsa, libao, jack oder OSS, die Musik kann aber auch über einen Icecast oder Pulseaudio Clienten ausgegeben werden. Der MPD-Client steuert den MPD-Server und hat selbst keinen Audio-Output.

[3] Bild-Quelle: http://images.wikia.com/mpd/images/6/68/Mpd-overview.png

5 Lastenheft

Bei der Erstellung dieses Lastenheftes wurde sich an folgendem Beispiel orientiert:

http://www.stefan-baur.de/cs.se.lastenheft.beispiel.html?glstyle=2010

5.1 Zielbestimmungen

Welche Ziele sollen durch den Einsatz der Software erreicht werden?

Dem einzelnen Benutzer soll das abspielen von Musik über eine Netzwerkverbindung ermöglicht werden, dabei soll die Steuerung von einem lokalen Client übernommen werden. Die Musik wird dabei vom MPD Server in einer Datenbank gespeichert, der Rechner auf dem dieser Dienst läuft ist auch primär für die Audioausgabe zuständig, desweiteren sind andere Audioausgabequellen wie z.B. PulseAudio möglich. Die Client-Rechner sollen die Ausgabe steuern und Abspiellisten auf dem Server verwalten können. Die Bedienung soll für alle Benutzer sehr einfach und komfortabel über einen lokalen Client realisiert werden. Bei jedem Start des Clients wird der Zustand des MPD-Servers ,,gespiegelt".

Standardmäßig sollen den Benutzern folgende Funktionen zuf Verfügung stehen:

- Abspielen von Musik
- Steuerung von Musik (Play, Stop, Skip, ...)
- Input-Stream via HTTP
- Erstellung und Verwaltung von Playlisten
- Verwaltung der der Audioausgabegeräte

Weitere Funktionen müssen modular integrierbar sein, allerdings müssen sie noch nicht implementiert werden. Einige Beispiele für weitere Funktionen wären:

- Finden von Album-Informationen [1]
- Speichern von Serveradressen
- Visualisierung der Musik
- Dynamische Playlisten

Die Systemsprache soll auf Englisch festgelegt werden.

5.1.1 Projektbeteiligte

Wer soll an dem Projekt teilnehmen?

- Christopher Pahl
- Christoph Piechula
- Eduard Schneider
- Marc Tigges

5.2 Produkteinsatz

Für welche Anwendungsbereiche und Zielgruppe ist die Software vorgesehen?

Der MPD-Client ist nicht auf bestimmte Gewerbe beschränkt, ein jeder soll diesen Client verwenden können.
Die Software soll unter folgender Lizenz stehen: Public License (GPL) Version 3 vom 29 Juni 2007.

Definition der GPL v3:

http://www.gnu.org/licenses/gpl.html

5.2.1 Anwendungsbereiche

Einzelpersonen verwenden dieses System überall da, wo mit einem Unix-artigen Betriebssystem Musik abgespielt werden soll. Das wären z.B. Personal Computer, Musikanlagen, Laptops und evtl. sogar diverse Smartphones möglich.

[1] Unter Verwendung von: https://github.com/sahib/glyr

5.2.2 Zielgruppen

Personengruppen die komfortabel von überall aus auf ihre Musik und Playlist zugreifen wollen ohne diese jedes mal aufwändig synchronisieren zu müssen (z.B. durch Abgleich von Datenträgern).

Aufgrund der für das System vorgesehenen Betriebsumgebung sind ebenso Kenntnisse im Umgang mit Unix nötig.

Der Benutzer muss die Systemsprache Englisch beherrschen.

5.2.3 Betriebsbedingungen

Das System soll sich bezüglich der Betriebsbedingungen nicht sonderlich von vergleichbaren Systemen bzw. Anwendungen unterscheiden und dementsprechend folgende Punkte erfüllen:

- Betriebsdauer: Täglich, 24 Stunden
- Keinerlei Wartung soll nötig sein
- Sicherungen der Konfiguration müssen vom Benutzer vorgenommen werden

5.3 Produktumgebung

5.3.1 Software

Softwareabhängigkeiten sollen durch den Entwickler bestimmt werden. Dies gewährleistet, dass der Entwickler diesbezüglich nicht eingeschränkt wird und somit mehr Möglichkeiten hat.

5.3.2 Hardware

Das Produkt soll möglichst wenig Anforderungen and die Hardware stellen, da die Software eventuell auch auf sehr Hardwarearmen Geräten (wie z.B. Smartphones) verwendet werden soll.

5.3.3 Orgware

Es soll nach Möglichkeit keine Orgware von nöten sein. Der Nutzer der Software soll sich um möglichst wenig Nebenläufiges zu kümmern haben.

5.4 Produktfunktionen

Welche sind die Hauptfunktionen aus Sicht des Auftraggebers?

5.4.1 Allgemein

Beim ersten Start des Systems soll eine Standard-Konfiguration geladen werden und die Verbindungseinstellungen zu einem MPD-Server müssen vorgenommen werden. Bei jedem weiteren Start soll die Konfiguration geladen werden, die vom Benutzer erstellt wurde, falls keine Konfiguration gefunden wurde, oder diese korrumpiert ist, so wird auf eine vom Client bereitgestellte Standardkonfiguration zurückgefallen. Der Benutzer soll sämtliche Einstellungen selbstverständlich zu jeder Zeit über die Oberfläche des Clients ändern können. Natürlich sollen alle üblichen Musik Abspielfunktionen vorhanden sein, dazu gehört Play, Stop, Previous und Next. Aber auch erweiterte Funktionen wie Repeat, Consume und Random sollen einstellbar sein. Der Benutzer soll über die Software direkten Zugriff auf das virtuelle Dateisystem des Servers haben, um nach Musik zu suchen und diese abspielen zu können. Aus dem Dateisystem heraus soll der Nutzer ebenfalls die Möglichkeit haben, Musik-Dateien direkt zu Playlisten und zur Warteschlange hinzuzufügen. Verbindungseinstellungen müssen auf möglichst einfache Art und Weise vorgenommen werden können, wenn möglich sollte dem Nutzer eine Liste von verfügbaren Servern angezeigt werden. Dem Nutzer soll ermöglicht werden, dass er nach bestimmten Titeln, Alben oder Interpreten suchen kann, da es mit dieser Software möglich ist, auch sehr große Musik-Datenbanken zu steuern. Administratorfunktionen müssen nicht implementiert werden, das sie vom Unix-System übernommen werden.

5.5 Produktdaten

Welche Daten sollen persistent gespeichert werden?

Die vom Benutzer vorgenommenen Verbindungseinstellungen und Client spezifischen Einstellungen, sollen auf dem Rechner lokal und persistent gespeichert werden. Nur so

kann ermöglicht werden, dass nach jedem Start des Systems diese Einstellungen geladen und übernommen werden können.
Außerdem soll eine Log-Datei auf den einzelnen Rechnern angelegt werden, die dieses System verwenden. Die Speicherung des Logs und der Konfiguration soll dabei dem XDG-Standard[2] entsprechen. In dieser Log-Datei werden Nachrichten des Systems gespeichert, um eventuelle Fehler leicht finden und beheben zu können. Es soll stets der aktuelle Zustand des MPD Servers widergespiegelt werden.
Dem Nutzer sollen viele verschiedene Informationen angezeigt werden, nicht nur Standardinformationen wie Titel, Album und Interpret, sondern auch Musik-Qualität, -Länge und Lautstärke. Es soll außerdem eine primitive Statistik implementiert werden die anzeigt, wie viele Lieder, Alben und Interpreten in der Datenbank vorhanden sind, wie lange man schon mit dem Server verbunden ist und wie lange die gesamte Abspielzeit aller Lieder in der Datenbank dauert.
Eine Profilverwaltung muss nicht implementiert werden, dies wird bereits über die Unix Benutzerverwaltung geregelt.
Eine lokale Datenbank muss ebenfalls nicht vorhanden sein, dies wird durch den MPD-Server ermöglicht.

5.6 Qualitätsanforderungen

Die Software soll natürlich von hoher Qualität sein. Hierfür sollen folgende Anforderungen erfüllt werden:

Die Software soll korrekt sein, d. h. möglichst wenige Fehler enthalten. Sie soll aber auch, für den Fall das dennoch Fehler auftreten, robust und tolerant auf diese reagieren. Außerdem spielt die Wartbarkeit eine wichtige Rolle, falls sich die Softwareumgebung des MPD-Clients ändert, muss dieser leicht angepasst werden können. Der Client soll intuitiv und schnell bedienbar sein. Der Hauptaugenmerk liegt dabei aber auf Ressourceneffizienz, vor allem soll dabei dabei das Netzwerk nicht stark beansprucht werden um auch bei langsamen Verbindungen den Client bedienen zu können. Speicher und Recheneffizienz ist hierbei zwar von Belang, aber dennoch zweitrangig. Sollte es Funktionen geben, die nicht unter den Begriff SStandardffallen, sollte eine knappe und präzise Beschreibung der Funktion vorhanden sein. Das Design der Software muss zwar ansprechend sein, ist

[2] http://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html#variables

im Endeffekt allerdings drittrangig.

5.7 Ergänzungen

5.7.1 Realisierung

Das System muss mit den Programmiersprachen C und/oder C++ realisiert werden. Dabei ist auf Objektorientierung zu achten, um Modularität und Wartbarkeit gewährleisten zu können. Es können beliebige Entwicklungsumgebungen verwendet werden, wobei allerdings auch ein Texteditor mit Syntaxhighlighting vollkommen ausreichend ist. Um einfaches und sicheres Arbeiten ermöglichen zu können, soll die Versionsverwaltungssoftware *git* benutzt werden, um die Entwicklungsdateien zu speichern und zu bearbeiten. Zu dem Projekt soll eine ausführliche Dokumentation erstellt werden, um dauerhafte Wartbarkeit und Anpassung des MPD-Clients gewährleisten zu können, dazu gehören auch entsprechende Software-Diagramme (wie z.B. UML).

5.7.2 Die nächste Version

Aufgrund des modularen Aufbaus kann das System beliebig oft und in verschiedene Richtungen weiterentwickelt werden. Weiter oben sind bereits Möglichkeiten für konkrete Erweiterungen aufgelistet.

6 Pflichtenheft

Bei der Erstellung dieses Pflichtenheftes wurde sich an folgendem Beispiel von Stefan Baur orientiert.[1]

6.1 Zielbestimmungen

6.1.1 Projektbeteiligte

Wer soll an dem Projekt teilnehmen?

- Christopher Pahl
- Christoph Piechula
- Eduard Schneider
- Marc Tigges

6.1.2 Muss-Kriterien

- Verbindungsaufbau
- Durchführung benutzerspezifischer Client-Einstellungen
- Musik-Steuerung
- Warteschlangenverwaltung
- Playlistverwaltung
- Datenbankverwaltung
- Frontend für die Settings
- Statistikanzeige
- Verwaltung der Ausgabegeräte

[1] http://www.stefan-baur.de/cs.se.pflichtenheft.beispiel.html?glstyle=2010

6.1.3 Wunsch-Kriterien

- Anzeige von Onlinecontent (Albencover, Lyrics etc.) unter Verwendung von libglyr[2]

6.2 Produkteinsatz

Welche Anwendungsbereiche (Zweck), Zielgruppen (Wer mit welchen Qualifikationen), Betriebsbedingungen (Betriebszeit, Aufsicht)?

Der MPD-Client ist nicht auf bestimmte Gewerbe beschränkt, ein jeder soll diesen Client verwenden können. Die Software soll unter folgender Lizenz stehen: Version 3 vom 29 Juni 2007.

Definition der GPL:

http://www.gnu.org/licenses/gpl.html

6.2.1 Anwendungsbereiche

Einzelpersonen verwenden dieses System überall da, wo mit einem Unix-artigen Betriebssystem Musik abgespielt werden soll. Das wären z.B. Personal Computer, Musikanlagen, Laptops und evtl. sogar diverse Smartphones möglich.

6.2.2 Zielgruppen

Personengruppen die komfortabel von überall aus auf ihre Musik und Playlist zugreifen wollen ohne diese jedes mal aufwändig synchronisieren zu müssen (z.B. durch Abgleich von Datenträgern). Aufgrund der für das System vorgesehenen Betriebsumgebung sind ebenso Kenntnisse im Umgang mit Unix nötig. Der Benutzer muss die Systemsprache Englisch beherrschen.

6.2.3 Betriebsbedingungen

Das System soll sich bezüglich der Betriebsbedingungen nicht sonderlich von vergleichbaren Systemen bzw. Anwendungen unterscheiden und dementsprechend folgende Punkte erfüllen:

- Betriebsdauer: Täglich, 24 Stunden

[2] https://github.com/sahib/glyr

- Keinerlei Wartung soll nötig sein
- Sicherungen der Konfiguration müssen vom Benutzer vorgenommen werden

6.3 Produktumgebung

6.3.1 Software

- Unixodes Betriebssystem
- MPD-Server
- Avahi Daemon
- Benötigte Bibliotheken können statisch einkompiliert werden

Ein MPD-Server muss nicht unbedingt lokal installiert sein, dann muss allerdings über das Netzwerk (Internet) ein Server erreichbar sein. Ohne MPD-Server soll der Client in einen definierten Zustand hochgefahren werden der das Verbinden ermöglicht. Avahi Daemon ist optional. Er ist nicht von nöten aber durchaus praktisch wenn man sich nicht ständig den Host (bzw. die IP) vom Administrator geben lassen will.

6.3.2 Orgware

- CMake (Buildsystem)
- g++ (C++ Compiler)
- Valgrind (Memorydebugger)
- git (Hosting auf Github)
- Glade (GUI-Designer)
- doxygen (Interne Dokumentationsgenerierung)
- Devhelp (Dokumentationsbrowser)

6.4 Produktfunktionen

Funktionen des MPD-Clients.
Beim ersten Start des Systems soll eine einkompilierte Standard-Konfiguration geladen werden und die Verbindungseinstellungen zu einem MPD-Server müssen vorgenommen werden. Bei jedem weiteren Start soll die Konfiguration geladen werden, die vom Benutzer erstellt hat, falls keine Konfiguration gefunden wurde, oder diese korrumpiert ist, so wird auf eine vom Client bereitgestellte Standardkonfiguration zurückgefallen. Der Benutzer soll sämtliche Einstellungen selbstverständlich zu jeder Zeit ändern können.

Um ein Gefühl für die Funktionalität zu bekommen die der Client haben soll, wurde mit Glade ein nichtfunktionales Mockup erstellt. Daraus sollen die finalen Features abgeleitet werden.

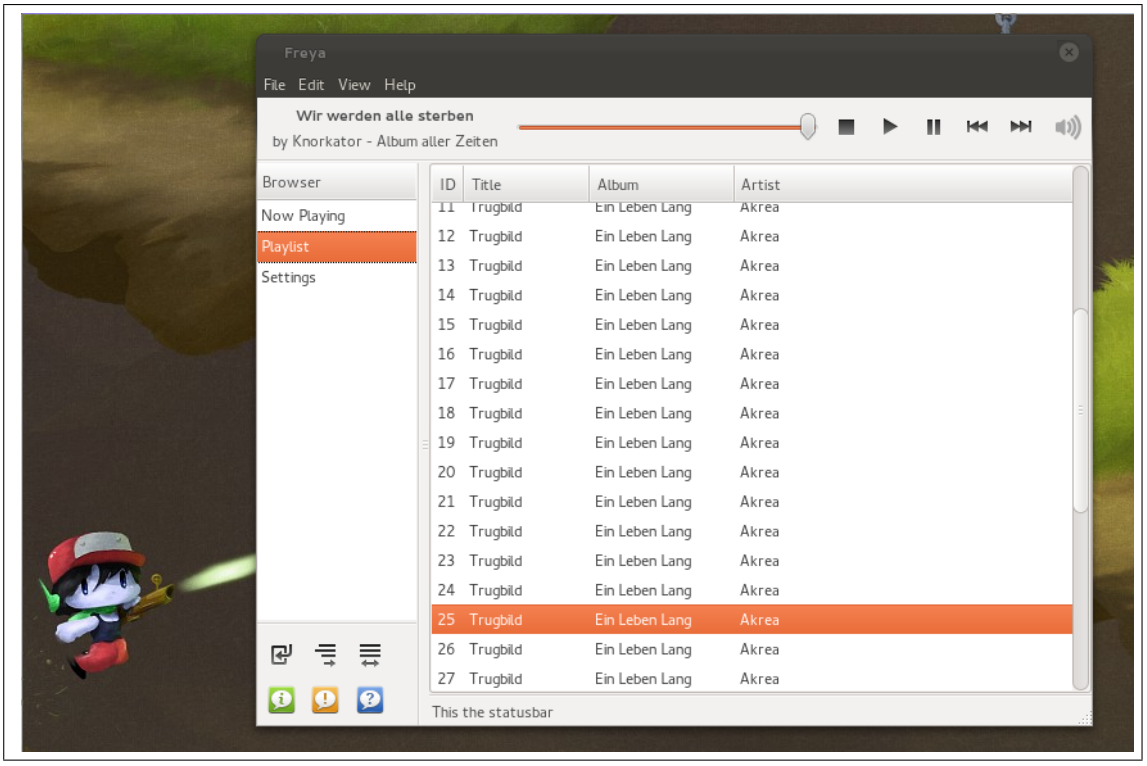


Abbildung 6.1: Mockup des MPD Clients

Die Auflistung der Features erfolgt von links oben nach rechts unten.

6.4.1 Menü

Überall Anzeige der Keyshortcuts.

- File
  - ,,Connect"; Insensitiv wenn bereits verbunden
  - ,,Disconnect", Insensitiv wenn nicht verbunden
  - ,,Quit"
- ,,Playback" Anzeige mit Haken falls aktiviert
  - Next
  - Previous
  - Play
  - Stop
  - Consume, Single, Random, Repeat -
- Misc
  - ,,Increase Volume"
  - ,,Decrease Volume"
- Help
  - About Dialog

6.4.2 Titelbar

- Anzeige des Musiktitels
  - Benutzerhinweis wenn Client nicht verbunden ist
- ,,Timeslider"
  - Zeigt die aktuelle Liedposition an
  - Durch klicken Sprung an bestimmte Musikposition möglich
  - Bei Stop wird Slider auf 0 gesetzt, bei Pause bleibt dieser stehen
- Steuerbuttons
  - Stopbutton - Stoppt Lied, setzt Timeslide zurück)

  - Pausebutton - zeigt ,,play icon" wenn pausiert und ,,stop icon" bei Wiedergabe
  - Pervious - vorheriges Lied
  - Next - nächtes Lied
  - Volumebutton - Popupslider zum regeln der Lautstärke
- Untere Zeile
  - Anzeige von ,,by (Artist) on (Album) ((Erscheinungsjahr))"
  - Falls nicht connected oder nicht spielend soll ,,Not Playing angezeigt"

6.4.3 Sidebar

Die Sidebar befindet sich links und zeigt eine Liste verfügbarer Browser

- Bei Klick auf einem Browser wird er ausgewählt
- der Inhalt wird im Pane daneben angezeigt
- Sollte die Browserliste überfüllt sein wird ein Scrollbalken angezeigt

Nach einem Separator findet sich ein inaktives widget das den Song anzeigt der als nächstes spielen wird. Falls kein nächster Song wird eine entsprechende Nachricht angezeigt. Darunter findet sich 4 eindrückbare Buttons die eindrückbar sind:

- Repeat
- Consume
- Repeat
- Single

Falls diese Buttons beispielsweise in einem anderen Client aktiviert werden, so sollen die Änderungen automatisch durch die Eindrückung angezeigt werden.

6.4.4 Playlistmanager

Zeigt alle vorhandenen auf dem Server gespeicherten Playlisten in einer Liste. Dabei wird in der Liste der Playlistname und das letzte Änderungsdatum angezeigt. Die Namenszellen sind editierbar, sodass der Playlistname einfach geändert werden kann. Ein Rechtsklickmenü bietet soll die folgenden Operationen bieten:

- Append: Fügt den Inhalt der Playliste der Queue am Ende hinzu
- Replace: Ersetzt den Inhalt der Queue mit dieser Playlist
- Delete: Entfernt die ausgewählten Playlisten unwiderruflich

6.4.5 Databasebrowser

Der Databasebrowser zeigt eine Visualisierung der Datenbank die an einen Filebrowser angelehnt ist. [3]

- Anzeige von Songs und Files durch unterschiedliche Icons
- Anzeige des "Dateinamenünter dem Icon
- Doppelklick oder Enter auf einen Ordner bewirkt ein Absteigen in diesem
- Doppelklick oder Enter auf ein Songfile bewirkt ein Hinzufügen dessen zur Ende der Queue
- 'Backspace' geht ebenso ein Verzeicniss nach oben.
- Am unteren Rand befindet sich eine Steuerleiste:
  - homebutton: Geht zum Wurzelverzeichniss zurück
  - Zurückbutton: Dasselbe wie backspace
  - Das Suchfeld filtert die momentane Anzeige
  - ganze rechts zeigt ein Label den aktuellen Pfad an
- Ein Kontextmenü bietet folgende Optionen:
  - Add: Fügt die ausgewählte Menge der Queue hinzu, rekursiv falls sich Verzeichnisse darunter befinden
  - Add All: Fügt alles, ungeachtet der auswahl der queue hinzu (performatere Version von add)
  - Replace: Wie add, löscht aber vorher die Queue
  - Update: Weist den Server an die Datenbank zu aktualisieren, und neue/-veränderte files zu aktualisieren

[3] Es handelt sich hierbei nicht wirklich um ein Dateisystem, nur um eine Darstellung der Musikfiles des Ordners, der in der Konfigurationsdatei des MPD-Servers also Musik-Ordner festgelegt wurde

  - Rescan: Weist den Server die Datenbank zu aktualisieren; untersucht alle files von neuen (teuer)

6.4.6 Queue

- Zeigt die aktuelle Warteschlange an
  - Künstler, Album und Musiktitel
  - Spalten der Queue sind frei anordenbar
- Auswahl erfolgt durch linksklick (kombiniert mit shift/strg) oder durch Auswählen mit der Maus („Rubberbanding")
- Ein Rechtsklickmenü bietet die folgenden Möglichkeiten:
  - Remove: Entfernen der Songs aus der Queue
  - Clear: Entfernen aller Songs aus der Queue
  - Save as playlist: Die gesamte Queue wird als playlist abgespeicher, der name der neuen playlist wird durch einen Dialog abgefragt.
- Bei Änderungen des Server wird die Queue im Hintergrund geupdatet

6.4.7 Settingsbrowser

Die Einstellungen sollen in ein Tabbasiertes Layout eingebettet sein. Werden Änderungen vorgenommen, so werden sie nicht gleich gespeichert und übernommen. Es sollte daher ein Speicherfunktion geben, und dazugehörig eine Undofunktion für die letzten Änderungen.
Falls dem Client die Verbindung verloren soll zum Settingsbrowser gesprungen werden, sodass der Benutzer entsprechende Änderungen machen kann. Aus diesem Grunde muss der Settingsbrowser auch ohne Verbindung voll funktionsfähig sein. Alle Settingstabs ändern die Werte nicht sofort:

- Sie werden erst persistent übernommen wenn der user die Konfiguration abspeichert
- Der Rückgängig button setzt die Präsentation auf die letzen persistent gesetzten Werte
- SZurücksetzenllädt überall die Defaultconfig

Innerhalb der Tabs sollen folgende Funktionen bereit gestellt werden:

- Der Benutzer kann Netzwerk-Einstellungen vornehmen
  - Server IP / Port
  - Avahi-Browser (Serverliste)
  - Autoconnect
- Der Benutzer kann Playback-Einstellungen vornehmen
  - Crossfade in Sekunden (Weicher Übergang zwischen jetzigem und nächstem Lied)
  - Musik beim verlassen stoppen
- Der Benutzer kann Allgemein-Einstellungen vornehmen
  - Notifications(libnotify) nutzen, falls ja auch wie lange diese angezeigt werden
  - Tray-Icon anzeigen
- Der Benutzer soll eine Audioausgabeliste haben:
  - Innerhalb der Liste soll es Checkboxes geben um den output entweder an oder auszuschalten
  - Dies soll nicht verfügbar sein wenn die Verbindung verloren geht

6.4.7.1 Fußleiste

Falls verbunden zeigt sie:

- Samplingrate in Khz
- Audiobitrate in Kbit
- Outputart (serverbedingt kann nur Stereo oder Mono angezeit werden, 5.1 Audiofiles werden auch als Stereo dargestellt)
- Zeit aktuell von insgesamt
- Anzahl an Songs in der Datenbank
- Komplette Abspielzeit der Datenbank
- Lautstärke 0-100%

6.4.7.2 Sonstiges

- D_0060 Nächster Song (Seitenleiste)

6.5 Produktdaten

6.5.1 Starten und Beenden

- Der Benutzer kann das System zu jedem Zeitpunkt starten.
- Der Benutzer kann das System zu jedem Zeitpunkt beenden.
- Beim ersten Start wird ein Standart-System-Zustand geladen.
- Beim Beenden wird der aktuelle System-Zustand gespeichert.
- Bei jedem weiteren Start wird der letzte System-Zustand geladen.

6.5.2 Abspielen von Musik (Buttons)

Der Benutzer kann

- Musik abspielen (Play)
- Musik stoppen (Stop)
- Musik pausieren (Pause)
- Musik vor und zurück schalten (Skip)
- Musik vor und zurück spuhlen (Seek)
- Musik zufällig abspielen (Random)
- Musik wiederholen (Repeat)
- Musik im Consume-Mode abspielen
- Musik im Single-Mode abspielen

6.5.3 Abspielen von Musik (Shortcuts)

Folgende Shortcut sollen in der finalen Version verfügbar sein:

- Play (ctrl + G)
- Stop (ctrl + S)
- Previous (ctrl + P)
- Next (ctrl + N)
- Random (ctrl + Z)
- Single (ctrl + Y)
- Repeate (ctrl + R)
- Consume (ctrl + T)
- Verbinden (ctrl + C)
- Trennen (ctrl + D)
- Beenden (ctrl + Q)

6.5.4 Administrator-Funktionen

Durch das Unix-artige System wird der Administrator-Zugriff geregelt. Sobald sich der Benutzer im Unix System als Administrator befindet, kann er auch den MPD-Client administrieren. Ein zusätzlicher Administrator-Modus wurde also nicht implementiert.

6.5.5 Suchen in der Queue

Eine einfache Textsuche zum finden von Titeln, Alben oder Interpreten innerhalb der Abspiellisten wurde implementiert. Dabei springt die Markierung des Textes beim eingeben von Zeichen in die Suche zu der ersten übereinstimmenden Stelle in der Plaxlist des Clients. Erst beim bestätigen der Eingabe im Suchfeld wird die Auswahl gefiltert.

- Der Benutzer kann seine Queue durchsuchen
- Der Benutzer kann sein Dateisystem durchsuchen

6.5.6 Statistik

- F_0070 Der Benutzer kann eine gesamt Statistik einsehen
  - Anzahl der Interpreten
  - Anzahl der Alben
  - Anzahl der Lieder
  - Musiklänge der Datenbank
  - Abspielzeit
  - Zeit Online bzw. mit MPD verbunden
  - Letztes Datenbank-Update

6.5.7 Persönliches Profil

Da die Software auf Unix-artige Systeme beschränkt ist, wurde keine Profil-Verwaltung implementiert. Die verschiedenen Profile werden durch die verschiedenen Profile des gesamten Betriebssystems definiert und differenziert. Für spätere Versionen könnte ergänzend auch ein Serveraddressbuch implementiert werden.

6.5.8 Mehrfachstart des Clients

Gegen mehrfaches Starten ist der Client nicht abgesichert. Die einzige Schnittmengen die mehrere Instanzen des Clients sich teilen liegt in der persistenten Datenspeicherung des Logs. Werden die Clients zeitversetzt gestartet sollte hier allerdings kaum etwas passieren.

6.5.9 Persönliche Datenbank

Eine persönliche Datenbank ist lokal nicht vorhanden. Die Datenbank des Benutzers befindet sich auf dem MPD-Server. Einzig und alleine modulare Erweiterungen des MPD-Clients können lokale Datenbank-Implementierungen erfordern.

6.5.10 Persönliche Einstellungen

Client Einstellungen werden lokal gespeichert, außerdem ist stets eine Defaultconfig vorhanden, falls die des Dateisystems defekt ist. Die Konfigurationsdatei wird nach dem XDG-Standard in /.config/freya/config.xml gespeichert. Den selben Speicherplatz wählt

auch die Logdatei (/.config/freya/log.txt). Sollten nur einzelne Werte in der Konfigurationsdatei nicht vorhanden sein so wird nachgeschaut ob die Defaultconfig diese Werte bereitstellt und es wird versucht sie von dort zu laden. So ist für valide Wert stets abgesichert dass mindestens ein Wert vorliegt.

6.6 Qualitätsanforderungen

Die Software soll natürlich von hoher Qualität sein. Hierfür sollen folgende Anforderungen erfüllt werden:

6.6.1 Korrektheit

Die Software muss möglichst fehlerfrei und korrekt sein. Es wurden Testszenarien und Testfälle erstellt, um Fehler zu finden und auszubessern. Aber auch wenn nach Veröffentlichung der Software ein Fehler gefunden werden sollte, wird dieser sofort ausgebessert. Bei schwerwiegenden Fehlern werden die Nutzer direkt auf den Fehler aufmerksam gemacht.

6.6.2 Wartbarkeit

Der Wartungsaufwand der Software ist gering bis gar nicht vorhanden. Ändert sich die Umgebungssoftware (z.B. der MPD-Server) dann sind die Änderungen so geringfügig bzw. trivial, dass sie den MPD-Client nicht beeinflussen werden. Fehler der Software (sollten Fehler auftreten) wären leicht analysier- bzw. prüfbar und natürlich auch leicht zu beheben. Zur Wartbarkeit gehört ebenso die Modularität, d. h. die Software ist technisch so realisiert, dass sie leicht erweitert werden kann, Stichwort Model View Controller (MVC). An kritischen Stellen versuchen Fehlerbehandlungroutinen alle Fehler möglichst vollständig abzufangen. Alle diese Routinen schreiben Meldungen in eine Log-Datei.

6.6.3 Zuverlässigkeit

Das System funktioniert und reagiert tolerant auf fehlerhafte Eingaben bzw. fehlerhafte Benutzung. Das Programm funktioniert sieben Tage die Woche und 24h am Tag und muss nicht abgeschaltet werden.

6.6.4 Effizienz

Der MPD-Client funktioniert möglichst effizient, d.h. das Programm ist schnell geladen und Eingaben des Benutzers werden praktisch sofort ausgeführt. Es gibt so gut wie keine Wartezeiten, jedenfalls sind diese so genannten Reaktionszeiten für den Benutzer nicht merkbar. Selbst bei sehr großen Musik-Datenbanken und Playlists benötigt das Programm kaum Rechenzeit und sonstige Hardwareressourcen.

6.6.5 Benutzbarkeit

Die Software ist leicht verständlich und intuitiv bedienbar. Nötige Kenntnisse zur Nutzung des MPD-Clients sind leicht zu erlernen. Hier wird allerdings davon ausgegangen dass der MPD Server bereits fertig eingerichtet ist. Sollte dies nicht der Falls sein, so sind Fähigkeiten in der Kommandozeile durchaus hilfreich.

6.6.6 Design

Das Design soll ansprechend und modern sein, allerdings wenn es Konflikte zwischen technischer Umsetzung und Design oder Effizienz und Design geben sollte, ist stets im Interesse der technischen Umsetzung bzw. der Effizienz zu entscheiden.

6.6.7 Hardware

Minimale Hardwareanforderungen: 500 Mhz, 512MB Ram, Festplattenspeicher ¡ 20MB
Empfohlene Hardwareanforderungen: 1 Ghz, 512MB Ram, Festplattenspeicher ¡ 20MB

6.6.8 Orgware und Entwicklungsumgebung

- CMake (Buildsystem)
- g++ (C++ Compiler)
- Valgrind (Memorydebugger)
- git (Hosting auf Github) [4]
- Glade (GUI-Designer)
- doxygen (Interne Dokumentationsgenerierung)

[4] https://github.com/studentkittens/Freya

- Devhelp (Dokumentationsbrowser)
- Unixodes Betriebssystem
- MPD-Server
- Avahi-Browser
- gcc (Compiler)
- libmpdclient
- gtkmm libraries

6.7 Globale Testszenarien und Testfälle

6.7.1 Cxxtest

Als Testframework wurde CxxTest ausgewählt. Die Gründe für diese Entscheidung werden gut von der offiziellen zusammengefasst:

> [5] CxxTest is a JUnit/CppUnit/xUnit-like framework for C/C++.
> It is focussed on being a lightweight framework that is well suited for integration into embedded systems development projects.
> CxxTest's advantages over existing alternatives are that it:
>
> - Doesn't require RTTI
> - Doesn't require member template functions
> - Doesn't require exception handling
> - Doesn't require any external libraries (including memory management, file/console I/O, graphics libraries)
> - Is distributed entirely as a set of header files (and a python script).
> - Doesn't require the user to manually register tests and test suites
>
> This makes it extremely portable and usable.

[5] http://cxxtest.tigris.org/

6.7.2 Testfälle

Für Teile des Programmes die nicht vom Testprotokoll erfasst werden, und automatisch gestestet werden sollen Testfälle mit Cxxtest geschrieben werden.

6.7.3 Testprotokoll

Um Fehler aufzuspüren, die die grafische Oberfläche betreffen, wurde ein Testprotokoll erstellt in dem zunächst alle möglichen Funktionen der grafischen Oberfläche aufgelistet werden. Außerdem müssen diese Funktionen mit anderen Funktionen kombiniert und mehrfach ausgeführt werden. Zu jedem dieser Fälle ist ein zu erwartendes Ergebnis festzulegen und anschließend zu überprüfen ob das erwartete Ergebnis eingetroffen ist. Das eingetroffene Ergebnis ist ebenfalls zu protokollieren. Es wurden jeweils die Buttons, sowie die Shortcuts geprüft.

6.7.3.1 Abspielfunktionen

Einfache Ausführung:

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Play	Musik spielt ab. Play wird zu Pause.	Ja
Pause	Musik pausiert. Pause wird zu Play.	Ja
Next	Nächstes Lied abspielen	Ja
Previous	Vorheriges Lied abspielen	Ja
Stop	Beende abspielen Pause wird zu Play.	Ja
Skipping	An Liedposition springen	Ja
Random	Musik der Queue zufällig abspielen	Ja
Repeat	Ein Lied wiederholen	Ja
Repeat all	Queue wiederholen	Ja
Consume Mode	Ein abgespieltes Lied entfernen	Ja
Single Mode	Ein Lied abspielen, dann Stoppen	Ja

Kombinierte Ausführung:

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Play, Stop, Play(1)	Musik spielt ab. Musik stoppt. Musik spielt ab.	Ja
Play, Pause, Play(2)	Musik spiel ab. Musik pausiert. Musik spiel ab.	Ja
Next, Next(3)	Skip weiter. Skip weiter.	Ja
Previous, Previous(4)	Skip zurück. Skip zurück	Ja
Next, Previous(5)	Skip weiter. Skip zurück	Ja
Previous, Next(6)	Skip zurück. Skip weiter	Ja
Random, Repeat all	Musik der Queue zufällig abspielen Queue wiederholen	Ja
Random, Consume Mode	Musik der queue zufällig abspielen Ein abgespieltes Lied entfernen	Ja
Random, Single Mode	Musik der Queue zufällig abspielen Ein Lied abspielen, dann stoppen	Ja
Consume Mode, Single Mode	Ein abgespieltes Lied entfernen Ein Lied abspielen, dann Stoppen	Ja
Consume Mode, Repeat all	Kann nur einmal durchlaufen	Ja
Random, 1	Musik der Queue zufällig abspielen 1	Ja
Random, 2	Musik der Queue zufällig abspielen 2	Ja
Random, 3	Musik der Queue zufällig abspielen 3	Ja
Random, 4	Musik der Queue zufällig abspielen 4	Ja

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Repeat all, 5	Queue wiederholen 5	Ja
Repeat all, 6	Queue wiederholen 6	Ja
Consume Mode, 1	Ein abgespieltes Lied entfernen 1	Ja
Consume Mode, 2	Ein abgespieltes Lied entfernen 2	Ja
Consume Mode, 3	Ein abgespieltes Lied entfernen 3	Ja
Consume Mode, 4	Ein abgespieltes Lied entfernen 4	Ja
Consume Mode, 5	Ein abgespieltes Lied entfernen 5	Ja
Consume Mode, 6	Ein abgespieltes Lied entfernen 6	Ja
Single Mode, 1	Ein Lied abspielen, dann Stoppen 1	Ja
Single Mode, 2	Ein Lied abspielen, dann Stoppen 2	Ja
Single Mode, 3	Ein Lied abspielen, dann Stoppen 3	Ja
Single Mode, 4	Ein Lied abspielen, dann Stoppen 4	Ja
Single Mode, 5	Ein Lied abspielen, dann Stoppen 5	Ja
Single Mode, 6	Ein Lied abspielen, dann Stoppen 6	Ja

Im folgenden wird auf das Protokoll der kombinierten Ausführung referenziert.

Mehrfache Ausführung:

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Fall 1 x 10	Fall 1 x 10	Ja
Fall 2 x 10	Fall 2 x 10	Ja
Fall 3 x 10	Fall 3 x 10	Ja
Fall 4 x 10	Fall 4 x 10	Ja
Fall 5 x 10	Fall 5 x 10	Ja
Fall 6 x 10	Fall 6 x 10	Ja
Fall 12 x 10	Fall 12 x 10	Ja
Fall 13 x 10	Fall 13 x 10	Ja
Fall 14 x 10	Fall 14 x 10	Ja
Fall 15 x 10	Fall 15 x 10	Ja
Fall 16 x 10	Fall 16 x 10	Ja
Fall 17 x 10	Fall 17 x 10	Ja
Fall 18 x 10	Fall 18 x 10	Ja
Fall 19 x 10	Fall 19 x 10	Ja
Fall 20 x 10	Fall 20 x 10	Ja
Fall 21 x 10	Fall 21 x 10	Ja
Fall 22 x 10	Fall 22 x 10	Ja
Fall 23 x 10	Fall 23 x 10	Ja
Fall 24 x 10	Fall 24 x 10	Ja
Fall 25 x 10	Fall 25 x 10	Ja
Fall 26 x 10	Fall 26 x 10	Ja
Fall 27 x 10	Fall 27 x 10	Ja
Fall 28 x 10	Fall 28 x 10	Ja
Fall 29 x 10	Fall 29 x 10	Ja
Fall 30 x 10	Fall 30 x 10	Ja
Fall 31 x 10	Fall 31 x 10	Ja
Fall 32 x 10	Fall 32 x 10	Ja
Fall 33 x 10	Fall 33 x 10	Ja
Fall 34 x 10	Fall 34 x 10	Ja
Fall 35 x 10	Fall 35 x 10	Ja

6.7.3.2 Queue-Funktionen

Einfache Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Remove	Ein Lied aus Queue entfernen	Ja
Clear	Alle Lieder aus Queue entfernen	Ja
Save as Playlist	Queue als Playlist speichern	Ja
Suchen	Nach eingegebenem Wort suchen	Ja

Kombinierte Ausführung

Kombinierte Ausführung der Funktionen der Queue machen nicht wirklich viel Sinn da z.B. die Funktion Clear die Queue löscht. Auch Save as Playlist wird wohl kaum öfter als einmal pro Queue angewandt. Die einzige Kombination die Sinn macht getestet zu werden ist die folgende:

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Suchen, Remove	Nach eingegebenem Wort suchen Ein Lied aus der Queue entfernen	Ja

Mehrfache Ausführung

Die mehrfache Ausführung ist ähnlich unsinnig wie die der kombinierten Ausführung. Mehrmals hintereinander die Queue löschen ist nicht möglich, genauso wie man wohl kaum mehrmals die gleiche Playlist erstellt. So bleibt wieder nur ein Testfall zu prüfen:

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Suchen, Remove x 10	Nach eingegebenem Wort suchen Ein Lied aus der Queue entfernen x 10	Ja

6.7.3.3 Playlist-Funktionen

Einfache Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Hinzufügen	Playlist hinzufügen	Ja
Ersetzen	Playlist ersetzen	Ja
Playlist entfernen	Playlist löschen	Ja

Kombinierte Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Hinzufügen, Hinzufügen	Playlist hinzufügen Playlist hinzufügen	Ja
Ersetzen, Ersetzen	Playlist ersetzen Playlist ersetzen	Ja
Entfernen, Entfernen	Playlist entfernen Playlist entfernen	Ja
Hinzufügen, Entfernen	Playlist hinzufügen Playlist löschen	Ja
Ersetzen, Entfernen	Playlist ersetzen Playlist entfernen	Ja

Im folgenden wird auf das Protokoll der kombinierten Ausführung referenziert.

Mehrfache Ausführung:

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Fall 1 x 10	Fall 1 x 10	Ja
Fall 2 x 10	Fall 2 x 10	Ja
Fall 3 x 10	Fall 3 x 10	Ja
Fall 4 x 10	Fall 4 x 10	Ja
Fall 5 x 10	Fall 5 x 10	Ja

6.7.3.4 Dateibrowser-Funktionen

Einfache Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Hinzufügen	Zur Queue hinzufügen	Ja
Alle Hinzufügen	Alle zur Queue hinzufügen	Ja
Ersetzen	Queue durch Auswahl ersetzen	Ja
Aktualisieren	Dateibrowser aktualisieren	Ja
Neu einlesen	Dateibrowser neu einlesen	Ja
Suchen	Nach eingegebenem Wort suchen	Ja

Kombinierte Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Hinzufügen Hinzufügen	Zur Queue hinzufügen Zur Queue hinzufügen	Ja
Alle Hinzufügen Alle hinzufügen	Alle zur Queue hinzufügen Alle zur Queue hinzufügen	Ja
Ersetzen Ersetzen	Queue durch Auswahl ersetzen Queue durch Auswahl ersetzen	Ja
Aktualisieren Aktualisieren	Dateibrowser aktualisieren Dateibrowser aktualisieren	Ja
Neu einlesen Neu einlesen	Dateibrowser neu einlesen Dateibrowser neu einlesen	Ja
Suchen Suchen	Nach eingegebenem Wort suchen Nach eingegebenem Wort suchen	Ja
Hinzufügen Alle Hinzufügen	Zur Queue hinzufügen Alle zur Queue hinzufügen	Ja
Hinzufügen Ersetzen	Zur Queue hinzufügen Queue durch Auswahl ersetzen	Ja
Hinzufügen Aktualisieren	Zur Queue hinzufügen Dateibrowser aktualisieren	Ja
Hinzufügen Neu einlesen	Zur Queue hinzufügen Dateibrowser neu einlesen	Ja

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Hinzufügen Suchen	Zur Queue hinzufügen Nach eingegebenem Wort suchen	Ja
Alle Hinzufügen Ersetzen	Alle zur Queue hinzufügen Queue durch Auswahl ersetzen	Ja
Alle Hinzufügen Aktualisieren	Alle zur Queue hinzufügen Dateibrowser aktualisieren	Ja
Alle Hinzufügen Neu einlesen	Alle zur Queue hinzufügen Dateibrowser neu einlesen	Ja
Alle Hinzufügen Suchen	Alle zur Queue hinzufügen Nach eingegebenem Wort suchen	Ja
Ersetzen Aktualisieren	Queue durch Auswahl ersetzen Dateibrowser aktualisieren	Ja
Ersetzen Neu einlesen	Queue durach Auswahl ersetzen Dateibrowser neu einlesen	Ja
Ersetzen Suchen	Queue durch Auswahl ersetzen Nach eingegebenem Wort suchen	Ja
Aktualisieren Neu einlesen	Dateibrowser aktualisieren Dateibrowser neu einlesen	Ja
Aktualisieren Suchen	Dateibrowser aktualisieren Nach eingegebenem Wort suchen	Ja
Neu einlesen Suchen	Dateibrowser neu einlesen Nach eingegebenem Wort suchen	Ja

Im folgenden wird auf das Protokoll der kombinierten Ausführung referenziert.

Mehrfache Ausführung

Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen?
Fall 1 x 10	Fall 1 x 10	Ja
Fall 2 x 10	Fall 2 x 10	Ja
Fall 3 x 10	Fall 3 x 10	Ja
Fall 4 x 10	Fall 4 x 10	Ja
Fall 5 x 10	Fall 5 x 10	Ja
Fall 6 x 10	Fall 6 x 10	Ja
Fall 7 x 10	Fall 7 x 10	Ja
Fall 8 x 10	Fall 8 x 10	Ja
Fall 9 x 10	Fall 9 x 10	Ja
Fall 10 x 10	Fall 10 x 10	Ja
Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Fall 11 x 10	Fall 11 x 10	Ja
Fall 12 x 10	Fall 12 x 10	Ja
Fall 13 x 10	Fall 13 x 10	Ja
Fall 14 x 10	Fall 14 x 10	Ja
Fall 15 x 10	Fall 15 x 10	Ja
Fall 16 x 10	Fall 16 x 10	Ja
Fall 17 x 10	Fall 17 x 10	Ja
Fall 18 x 10	Fall 18 x 10	Ja
Fall 19 x 10	Fall 19 x 10	Ja
Fall 20 x 10	Fall 20 x 10	Ja
Fall 21 x 10	Fall 21 x 10	Ja

6.7.3.5 Statistik

Für die Statistik kann kein Testprotokoll angewandt werden.

6.7.3.6 Einstellungen

Einfache Ausführung

Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen?
Zeige Liste	Zeige Avahi Liste	Ja

Kombinierte Ausführung

Es existieren keine Buttons oder Shortcuts die kombiniert werden könnten.

Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen?
Fall 1 x 10	Fall 1 x 10	Ja

6.7.3.7 Lautstärke

Einfache Ausführung

Lautstärke erhöhen	Lautstärke erhöhen	Ja
Lautstärke verringern	Lautstärke verringern	Ja
Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**

Kombinierte Ausführung

Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen?
Lautstärke erhöhen Lautstärke verringern	Lautstärke erhöhen Lautstärke verringern	Ja

Im folgenden wird auf das Protokoll der kombinierten Ausführung referenziert.

Mehrfache Ausführung

Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen?
Fall 1 x 10	Fall 1 x 10	Ja

6.7.3.8 Sonstiges

Einfache Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Verbinden	Verbindung zum MPD-Server	Ja
Trennen	Verbindung zum Server trennen	Ja
Beenden	MPD-Client beenden	Ja

Kombinierte Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Verbinden Verbinden	Verbindung zum MPD-Server Verbindung zum MPD-Server	Ja
Verbinden Trennen	Verbindung zum MPD-Server Verbindzung zum Server trennen	Ja
Verbinden Beenden	Verbindung zum MPD-Server MPD-Client beenden	Ja
Trennen Beenden	Verbindung zum Server trennen MPD-Client beenden	Ja

Im folgenden wird auf das Protokoll der kombinierten Ausführung referenziert.

Mehrfache Ausführung

Testfall	**Erwartetes Ergebnis**	**Ergebnis eingetroffen?**
Fall 1 x 10	Fall 1 x 10	Ja
Fall 2 x 10	Fall 2 x 10	Ja
Fall 3 x 10	Nur 1 x ausführbar	Ja
Fall 4 x 10	Nur 1 x ausführbar	Ja

7 Software Design

7.1 Einführung

7.1.0.9 Namespace-Übersicht

7.2 ,,Das Problem"

Da die grundlegende Funktionsweise des MPD Server auf einer Client Server Architektur beruht, muss der MPD Client verschiedene Kommandos wie zum Beispiel play, pause, listplaylists etc. an den Server schicken und zur gleichen Zeit aber auch auf Änderungen reagieren können, d.h. zum Beispiel wenn sich die Lautstärke ändert, da jederzeit auch andere Clients oder Server den MPD internen Zustand ändern können. Diese Änderungen müssen auch anderen Programmteilen bekannt gemacht werden. (Observer Pattern) Der Client sollte im ,,idle"-Mode möglichst keine Ressourcen verschwenden und auch beim disconnecten und conntecten die entprechenden Änderungen anderen Teilen des Programms mitteilen können(Observer Pattern)[1].

Das MPD Protokoll [2] bietet folgende Möglichkeiten das zu realisieren

Periodisch (zB. alle 500ms) das ,,status" command absetzen und nach Bedarf auch commands wie ,,currentsong" senden
Problem: Bei langsamen Netzwerkverbindungen erzeugt dies unnötige Netzwerklast Prinzipiell würde sich auf diese Art jedoch die z. B. Musik Bitrate anzeigen lassen, es ist jedoch ein wenig komfortabler Weg da hier wieder einmal das Rad neu erfunden werden müsste.

Nutzung der ,,idle" und ,,noidle" commands: ,,idle" versetzt die Verbindung zum Server in einen Schlafzustand, sobald ,,events" wie 'player' (also z. B. pause oder play)

[1] http://de.wikipedia.org/wiki/Observer_(Entwurfsmuster)
[2] http://www.musicpd.org/doc/protocol/index.html

eintreten, wacht die Verbindung aus diesem Zustand auf und sendet an den Client eine Liste der Events die aufgetreten sind:

```
changed: player
changed: mixer
...
OK
```

Einschränkung: Während die Verbindung im idle mode ist kann kein reguläres Kommando wie ,,play" gesendet werden! Sollte man es doch tun wird man vom Server augenblicklich mit einem Disconnect belohnt. Die einzige Möglichkeit aus dem idle mode aufzuwachen ist das 'noidle' command das gesendet werden kann während die verbindung schlafen gelegt wurde. Jedoch gibt es auch hier ein Problem, denn das ,,idle" command blockiert, sprich es sendet kein ,,OK" zurück zum Sender. Ein Warten auf dieses ,,OK" würde mit den Wunsch eine bedienbare Oberfläche zu haben kollidieren.

Prinzipiell gibt es 2 Möglichkeiten dieses Problem zu lösen:

- Man hält zwei Verbindungen zum Server, eine die Kommandos sendet, eine die stets im ,,idle" mode liegt, Für die Realisierung müssten Threads herangezogen werden. Ein Thread würde dann im Hintergrund auf events lauschen, der andere würde zum Abschicken der Kommandos benutzt werden. Problem: Es müssen 2 Verbindungen gehandelt werden, was wiederum ein Mehraufwand an Code bedeutet. Desweiteren werden Threads benötigt die auch in anderen Bereichen des Programms Lockingmechanismen bedeuten würden.

- Man hält eine asynchrone verbindung zu dem server. Diese kann das 'idle' command zum server schicken, returned aber sofort. Um nun eine Liste der events zu bekommen setzt man einen ,,Watchdog"auf die asynchrone verbindung an (Vergleiche dazu den Systemaufrug 'man 3 poll'). Da poll() ebenfalls den aufrufenden Prozess blockiert, wird Glib::signal_io() benutzt, das sich in den laufenden MainLoop (*) einhängt und eine Callbackfunktion aufruft sobald auf der verbindung etwas interessantes passiert. Da während des Wartens der MainLoop weiterarbeitet, bleibt die GUI (und andere Module) aktiv und benutzbar. Problem: Vor dem Senden eines Kommandos wie ,,play" muss der idle mode verlassen werden. Lösung: Man kann das ,,noidle" Kommando zum verlassen senden, und nach dem Absenden des eigentlichen Kommandos wieder den idle-mode betreten.

```
(master) $ telnet localhost 6600
Trying ::1...
Connected to localhost.
OK MPD 0.16.0       # Der Server antwortet bei verbindungsaufbau
                    # stets mit einem OK und der Versionsnummer
pause               # Wir senden das 'pause' kommando zum
                    #pausieren des aktuellen liedes
OK                  # Der Server fuehrt es aus und antwortet
                    # mit einem OK
play                # Wir tun dasselbe mit dem 'play' command.
OK
idle                # Wir sagen dem server dass wir die
                    # Verbindung schlafen legen wollen...
changed: player     # Er returned aber sofort da seit dem
                    # Verbindungsaufbau etwas geschehen ist.
changed: mixer      # Und zwar wurde der Player pausiert,
                    # und das volume geaendert.
OK                  # Das Ende des idlemodes wird wieder
                    # mit OK angezeigt.
idle                # Probieren wir es noch einmal..
                    # Er antwortet nicht mit OK sondern
                    # schlaeft jetzt. Wuerden wir in
                    # einem anderen client pausieren
                    # So wuerde er hier aufwachen.
noidle              # Um aus den idlemode vorher aufzuwachen
                    # senden wir das noidle command
OK                  # OK sagt uns dass alles okay ist.
idle                # Probieren wir mal ein command zu senden
                    # waehrend die verbindung idlet:
play                # zum Beispiel das play command... als
                    # antwort wird die verbindung geschlossen:
Connection closed by foreign host.
$ Freya git:(master) $ echo 'ende.'
```

Die Idee zu dieser Implementierung (speziell das Benutzen einer asynchronen Ver-

bindung), kommt von ,,ncmpc", der inoffiziellen offiziellen Referenzimplementierung des MPD Mit-Authors *Max Kellermann*. Vergleiche ncmpc quellcode: src/gidle.c und src/mpdclient.c

7.3 Aufbau des Clients

Aus den oben genannten Anforderungen kann eine grobe Architektur abgeleitet werden:

7.3.1 Hauptklassen

7.3.1.1 BaseClient

- Kann nicht selbst instanziert werden.
- Verwaltet connect / disconnect und reconnect vorgänge
- Bietet Funktionen zum einfachen verlassen und eintreten des idlemodes an
- Implementiert keine konkreten Kommandos die er an den server schicken kann
- Geht die verbindung verloren (ohne dass *disconnect()* explizit aufgerufen wurde), so versucht er periodisch sich zu reconnecten.

7.3.1.2 Listener

- Verwaltet das ein- und austreten aus dem Idlemode
- Parst die Responseliste (also changed: player)
- Verfügt über ein ,,EventNotifer" (ein sigc::signal) Module können sich über connect() registrieren, bemerkt der Listener events so ruft er emit() auf dem signal auf und teilt allen anderen Modulen so mit welche events geschehen sind.
- Zudem bietet er eine Möglichkeit ein update zur forcen, das heißt ,,künstlich" alle möglichen events auzulösen was nützlich zum Initialisieren ist (force_update())
- Bei einem connect vorgang wird eine Instanz des Listeners instanziert und sofort der idlemodus betreten
- Bei einem disconnect wird der Listener gelöscht.

7.3.1.3 Connection

- Ein Wrapper um die mpd_connection Struktur von libmpdclient
- Ruft letzendlich mpd_connection_new() auf
- Bietet eine Schnittstelle um sich über Fehler informieren zu lassen (signal_error())
- Bietet eine get_connection() methode die bei jedem aufruf prüft ob fehler passiert sind In diesem Falle versucht MPD::Connection den Fehler zu bereinigen (falls ein nicht fataler Fehler war). Anschließend benachrichtigt MPD::Connection alle module die sich vorher über signal_error() registriert haben (wie der BaseClient es beispielsweise mit handle_error() tut)

7.3.1.4 Client

- Der Client erbt von BaseClient und implementiert konkrete Commandos wie ,,play",,,random" etc.
- Er bietet zudem Schnittstellen zur Befüllung der Datenbank, der Queue und des Playlistmanagers
- Er bietet die Methoden connect() und disconnect()
- Ist in der config ,,settings.connection.autoconnect" gesetzt so connected er sich automatisch.
- Er bietet zudem eine schnittstelle um sich beim listener zu registrieren und im falle von änderungen des connection zustands benachrichtigt zu werden.

7.3.1.5 NotifyData

- Speichert den Status, den aktuellen Song und die aktuelle Datenbankstatistik
- Der Listener...
  - instanziert NotifyData im Konstruktor
  - sagt NotifyData wann er sich updaten soll (update_all())
  - gibt eine Referenz auf NotifyData an alle registrierten Module weiter, damit diese konkrete Informationen beziehen können.

7.3.2 Weitere Klassen

Desweiteren gibt es einige weitere Klassen die am Rande eine Rolle spielen, und meist Objektorientierte Wrapperklassen für die C-Strukturen von libmpdclient bereitstellen.

7.3.2.1 Song

Die Song Klasse für Wrapper für mpd_song Struktur und die dazugehörigen Klassen (libmpdclient). Soll alle Funktionen von libmpdclient [3] anbieten, diese werden hier nur aufgelistet aber nicht erklärt da sie genau wie ihre Vorbilder funktionieren:

```
const char * get\_path(void);
const char * get\_tag(enum mpd\_tag\_type type, unsigned idx);
unsigned get\_duration(void);
time\_t get\_last\_modified(void);
void set\_pos(unsigned pos);
unsigned get\_pos(void);
unsigned get\_id(void);
```

MPD::Song soll zudem eine Funktion bieten um die Metadaten des Songs in einer printf änhlichen Art als String zurückzuliefern:

```
Glib::ustring song\_format(const char* format, bool markup=true);
```

Ein beispielhafter Aufruf:

```
SomeSong.song\_format("Artist is by \${artist}")
```

Die folgenden Tagarten sollen dabei unterstützt werden (sie spiegeln in etwa die mpd_tag_type Enumeration von libmpdclient wieder)

- artist
- title
- album
- track
- name

[3] http://www.musicpd.org/doc/libmpdclient/song_8h.html

- data
- album_artist
- genre
- composer
- performer
- comment
- disc

Ist ein Escapestring nicht bekannt, so wird er nicht escaped. Ist der tag nicht vorhanden soll mit ünknownëscaped werden.

7.3.2.2 Directory

Die Directory Klasse ist Wrapper für mpd_directory C-Strukutr. Diese wird als Anzeige für ein Verzeichniss benutzt, jedoch nicht als Container für andere Elemente.

Entsprechend implementiert bietet MPD::Directory nur:

```
void get_path(void);
```

Dies ist von der AbstractComposite vorgegeben.

7.3.2.3 Statistics

Die Statistics Klasse ist Wrapper für mpd_stats, implementiert gemäß http://www.musicpd.org/doc/libmpdclient/stats_8h.html folgende Funktionen:

```
unsigned get\_number\_of\_artists(void);
unsigned get\_number\_of\_albums(void);
unsigned get\_number\_of\_songs(void);
unsigned long get\_uptime(void);
unsigned long get\_db\_update\_time(void);
unsigned long get\_play\_time(void);
unsigned long get\_db\_play\_time(void);
```

7.3.2.4 Playlist

Die Playlist Klasse ist Wrapper für die mpd_playlist Struktur, implementiert von http://www.musicpd.org/d folgende Funktionen:

```
const char * get\_path(void);
time\_t get\_last\_modified(void);
```

Bietet desweiteren funktionen zum: Entfernen der Playlist vom Server (Das Playlist-objekt ist danach invalid):

```
void remove(void);
```

Laden der Playlist in die Queue:

```
void load(void);
```

Umbennen der Playlist:

```
void rename(const char * new\_name);
```

Hinzufügen von Songs zur Playlist:

```
void add\_song(const char * uri);
void add\_song(MPD::Song& song);
```

Die genannten Funktionen benötigen müssen den idlemode verlassen können, daher leitet MPD::Playlist von AbstractClientExtension ab.

7.3.2.5 AudioOutput

Die AudioOutput Klasse ist ein Wrapper für mpd_output, implementiert von http://www.musicpd.org/doc/
folgende Funktionen:

```
unsigned get\_id(void);
const char * get\_name(void);
bool get\_enabled(void);
```

Bietet desweiteren funktionen zum:

- Enablen des Ausgabegerätes:

```
bool enable(void);
```

- Disablen des Ausgabegerätes:

```
bool disable(void);
```

Die genannten Funktionen benötigen müssen den idlemode verlassen können, daher leitet MPD::AudioOutput von AbstractClientExtension ab.

7.3.3 Abstrakte Klassen

7.3.3.1 AbstractClientExtension

Diese abstrakte Klasse erlaubt abgeleiteten Klasse ähnlich zum BaseClient eigene Kommandos zu implementieren. Wird von MPD::Playlist und MPD::AudioOutput benutzt

7.3.3.2 AbstractClientUser

- Verwaltet einen Pointer auf die MPD::Client Klasse, so dass der Anwender der Klasse dies nicht selbst tun muss.
- Leitet man ab so müssen folgenden Methoden implementiert werden:

```
void on\_client\_update(enum mpd\_idle event, MPD::NotifyData
```

Wird aufgerufen sobald der Listener eine Änderunge feststellt, siehe weiter unten Ïnteraktion des Clients mit anderen Modulenffür eine genauere Erklärung.

```
void on\_connection\_change(bool server\_changed, bool is\_co
```

Wird aufgerufen sobald sich der verbunden/getrennt hat. Im ersten Fall ist is_connected true, im anderen false. Sollte sich der Client verbunden haben, und der neue Server entspricht nicht mehr dem neuen so ist auch server_changed true. Dies ist automatisch wahr beim ersten Start. Diese werden automatisch durch Ableiten von AbstractClientUser registriert. Weiterhin können alle Klassen über den mp_Client Pointer auf den Client zugreifen.

7.3.3.3 AbstractItemlist

Für bestimmte Client funktionen muss eine Nutzerklasse von AbstractItemlist ableiten. Leitet man ab so muss die Methode add_item(AbstractComposite * data) implementiert werden. Je nach Bedarf kann über dynamic_cast¡Zieltyp*¿(data) der entsprechende Datentyp rausgecasted werden. Beim Aufruf von MPD::Client::fill_queue ruft der Client die add_item methode für jeden song den er vom server bekommt auf. Die ableitende Klasse kann diese dann verarbeiten.

Dadurch werden alle Methoden von AbstractItemGenerator (bzw. die Klassen die davon ableiten) benutzbar:

- fill_queue
- fill_queue_changes
- fill_playlists
- fill_ouputs
- fill_filelist

7.3.3.4 AbstractItemGenerator

Lässt ableitende Klasse folgende Methoden implementieren: Jede dieser Methoden ruft MPD::Playlist add_item() von AbstractItemlist auf um ihre Resultate weiterzugeben.

Holt alle Songs der aktuellen Queue.

```
void fill\_queue(AbstractItemlist& data\_model);
```

Holt alle geänderten Songs in der Queue seit der Version last_version. Die Position des ersten geänderten Songs wird in first_pos gespeichert.

```
void fill\_queue\_changes(AbstractItemlist& data\_model, unsigned last\_ve
```

Holt alle gespeicherten Playlisten vom Server.

```
void fill\_playlists(AbstractItemlist& data\_model);
```

Holt alle Audio Outputs vom Server.

```
void fill\_outputs(AbstractItemlist& data\_model);
```

Holt alle Songs und Directories aus der Datenbank im Pfad path (nicht rekursiv!)

```
void fill\_filelist(AbstractItemlist& data\_model, const char * path);
```

7.3.3.5 AbstractComposite

Vereinheitlicht Zugriff auf Komponenten verschiedenen Types. Die abstrakte Klasse zwingt seine Kinder dazu eine *get_path()* zu implementieren die die Lage im virtuellen Filesystem des Servers angibt. Der Hauptnutznieser dieser Klasse ist der Databasebrowser, bzw. den dahinter gelagerten Cache Songs und Verzeichnisse gleich zu behandeln.

Die erbende Klasse muss im Konstruktor angeben ob es sich bei der Klasse um ein ,,File" (*true* für MPD::Song) oder um einen ,,Container" (*false* für MPD::Directory) handelt. Diese ,,is_leaf" Eigenschaft kann später mit der Funktion *is_leaf()* abgefragt werden.

7.4 Interaktion des Clients mit anderen Modulen

- Die meisten GUI Klassen leiten von AbstractClientUser ab und speichern daher eine Referenz auf eine Instanz von MPD::Client Sie können daher Funktionen wie queue_add() direkt aufrufen.
- AbstractClientUser zwingt die abgeleitenden Klassen folgende Funktionen zu implementieren:

```
void on_client_update(mpd_idle event, MPD::NotifyData& data)
void on_connection_change(bool server_changed, bool is_connec
```

1) wird aufgerufen sobald der Listener ein Event festgestellt hat. Für jedes eingetretene Event wird 1) einmal aufgerufen. 'event' ist dabei eine Enumeration aller möglichen Events, die von libmpdclient vorgegeben werden. 'data' ist eine Referenz auf eine Instanz von MPD::NotifyData. Die benutzenden Klassen können mit

  - get_status() gibt den aktuellen MPD::Status
  - get_song() gibt den aktuellen MPD::Song
  - get_statistics() gibt die aktuellen MPD::Statistics

so bei Änderungen sofort die aktuellen Änderungen auslesen.

2) on_connection_change wird vom Client aufgerufen sobald die Verbindung verloren geht. Dabei zeigt der übergebene boolean Wert ,,is_connected" an ob man connected wurde, oder disconnected wurde. ,,server_changed" soll dann anzeigen ob der Server derselbe ist beim zuvor geschehenen Connectvorgang. Dies ist beim ersten Start stets wahr. ,,server+_changed" kann nicht wahr sein wenn ,,is_connected" falsch ist.

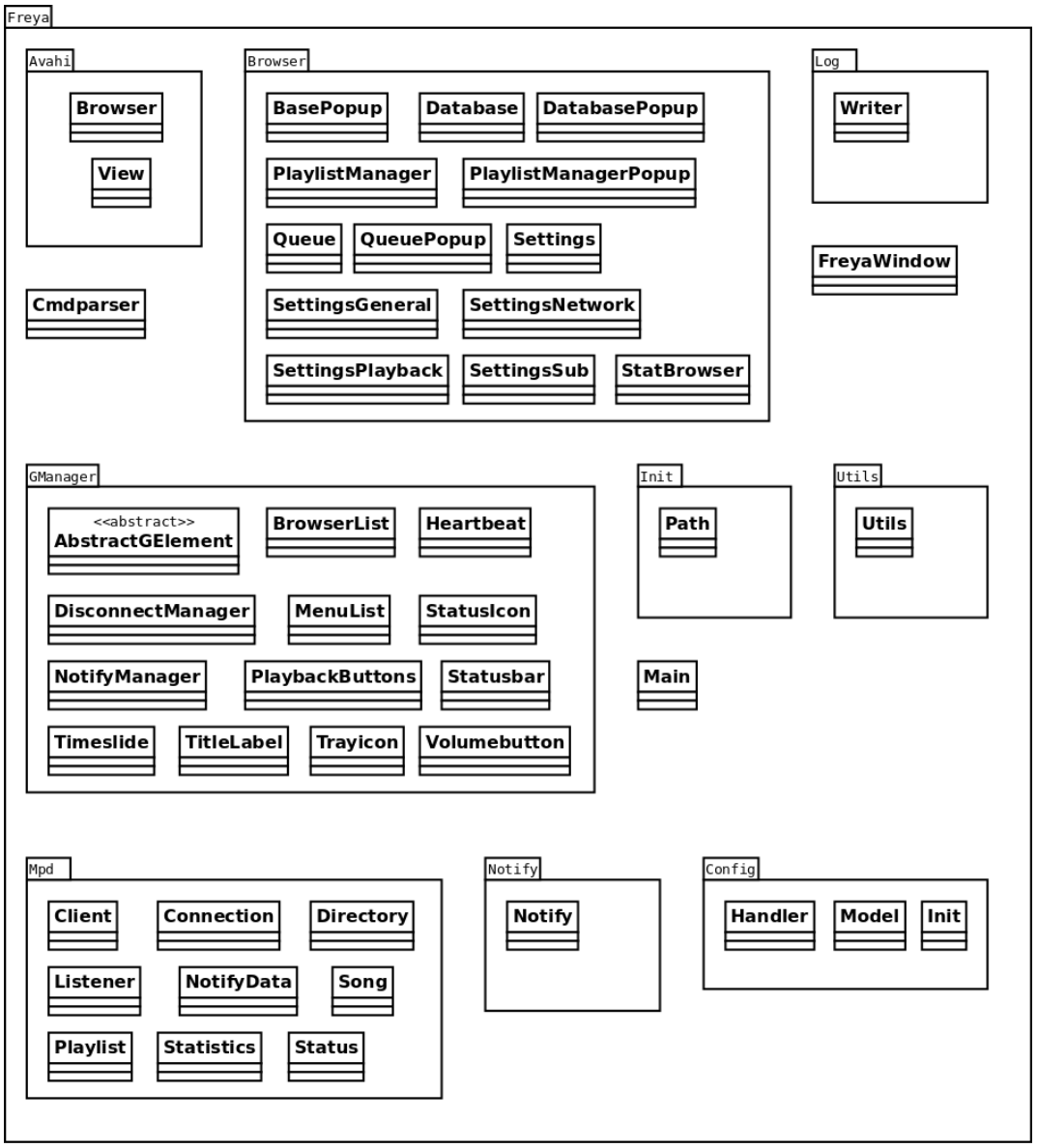
- Ableitung von den oben beschriebenen abstrakten Klassen AbstractItemlist und AbstractFilebrowser, um alle Funktionen von AbstractItemGenerator nutzen zu können

7.5 Config

7.6 GUI Elementklassen

7.7 Browserimplementierungen

7.8 Glossar


Freya
Avahi
Browser
View
Cmdparser
Browser
BasePopup
Database
DatabasePopup
PlaylistManager
PlaylistManagerPopup
Queue
QueuePopup
Settings
SettingsGeneral
SettingsNetwork
SettingsPlayback
SettingsSub
StatBrowser
Log
Writer
FreyaWindow
GManager
<<abstract>>
AbstractGElement
BrowserList
Heartbeat
DisconnectManager
MenuList
StatusIcon
NotifyManager
PlaybackButtons
Statusbar
Timeslide
TitleLabel
Trayicon
Volumebutton
Init
Path
Main
Utils
Utils
Mpd
Client
Connection
Directory
Listener
NotifyData
Song
Playlist
Statistics
Status
Notify
Notify
Config
Handler
Model
Init