MPD-Client

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

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

19. Dezember 2011

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

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

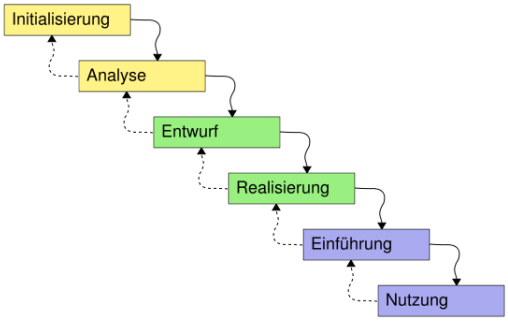
2.1 Definition

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.
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.

3 Richtlinien

3.1 Programmierrichtlinien

- Allman-Stil.
- Tabstop = 4 Leerzeichen.
- Keine globalen Variablen.
- Sinnvolle Variablenbenennung, “lowercase“.
- Klassenmethoden nur in Ausnahmefällen bzw. nur mit guten Gründen.
- Valgrind darf keine Laufzeitfehler bringen.
- “camelcase“ bei Objektnamen, C-Style bei Funktionsnamen - Präzise Namen.
- Modulare Gestaltung.
- Code-Sauberkeit ist wichtiger als Code Performance.
- “make“ sollte keine Warnungen ausgeben, die man leicht umgehen könnte.
- “make test“ soll vollständig durchlaufen.

3.1.1 Begründung

Diese Programmierrichtlinen sorgen für ein einfaches, übersichtliches und einheitliches Arbeiten. Jeder kann sich ohne größere Umstände in den Code eines anderen einlesen. 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

- Avahi-Daemon
- git
- Glade
- doxygen

3.2.1 Begründung

Avahi-Daemon ist ein Dienst, durch den man schnell und einfach MPD-Server im Netz finden und eine Verbindung zu den Servern aufbauen kann. Git dient zur Versionsverwaltung. Glade bietet eine perfekte Trennung von der grafischen Oberfläche zum Kontrollkern des Programms außerdem kann mit Glade sehr einfach eine grafische Oberfläche erstellt werden. Doxygen !!!!!!!Literate-Programming (KNUT)!!!!

3.3 Bibliotheken

- C++
- gtkmm3
- libmpd
- libxml2
- libnotify
- Avahi-glib

3.3.1 Begründung

C++ wurde gewählt um die Fähigkeiten der Authoren zu erweitern. Außerdem gibt es für Java nur wenige oder sehr schlechte Bibliotheken für dieses Projekt. Gtkmm3 bietet ein dynamisches Layout und ist leichtgewichtiger als qt. Swing ist unter C++ nicht nutzbar. Libmpd liefert libmpdclient mit. Libxml2 liefert eine standardisierte config nach Xml-Standards, ist sehr leichtgewichtig und überall installiert. Libnotify liefert Benachrichtigungen über interne Events und ist auf den meisten Linux Distributionen verbreitet. Avahi-glib dient als Server-Browser.

Als primäre Entwicklerpalttform wurde Linux gewählt.

4 Definition des Projekts

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.

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.

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. [1]

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

4.1.1 Der MPD kann:

- Musik abspielen
- Musik kontrollieren und in Warteschlangen reihen (lokal oder über TCP)
- Musik Dateien dekodieren
- HTTP(Hyper Text Transfer Protocol) streamen
  - Eine HTTP-URL kann zur Warteschlange hinzugefügt oder direkt abgespielt werden.

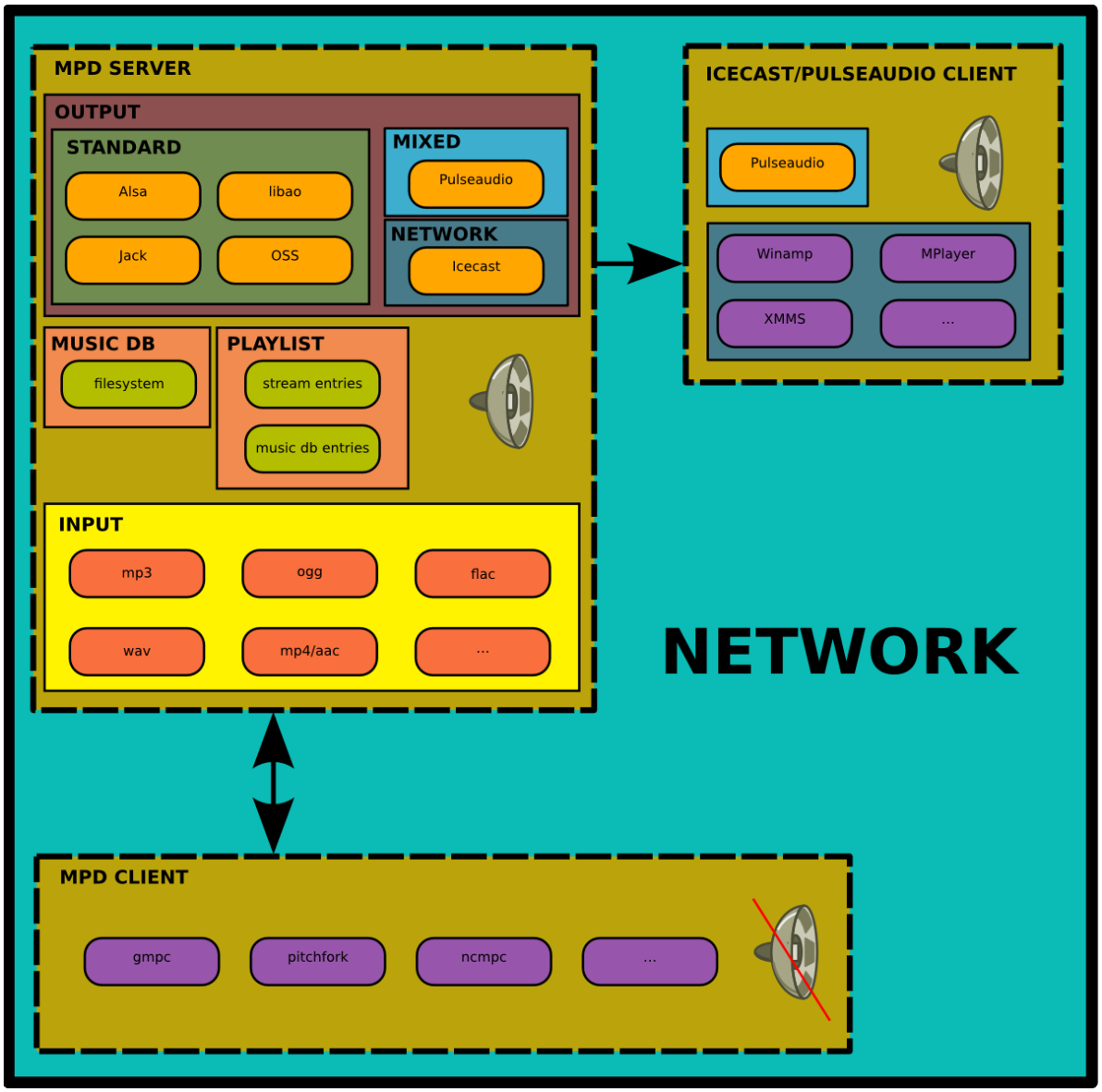
4.1.2 Der MPD kann nicht:

- 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 3̈.1.2 Der MPD kann nicht:ërwähnt wurden.

4.3 Grafische Übersicht



[2] 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.

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

5 Lastenheft

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 soll in eine rzentralen Datenbank angelegt und über die Soundkarte eines Servers abgespielt werden. 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, soll die letzte Sitzung wiederhergestellt werden, falls keine Daten eine beendeten Sitzung gefunden werden, sollen Standarteinstellungen verwendet werden.

Standartmäßig sollen den Benutzern folgende Funktionen zuf Verfügung stehen:

- Abspielen von Musik
- Steuerung von Musik (Play, Stop, Skip, ...)
- Decodieren von Musik
- Input-Stream via HTTP

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

- Finden von Album-Informationen
- Profil-Steuerung
- Visualisierung

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. Grundlage für die Verwendung der Software ist die General 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

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).

Es werden Basiskenntnisse zum Aufbau einer Netzwerkverbindung und zur Nutzung des Internets vorausgesetzt. 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 folgend 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 diese denn lokal gefunden werden kann. Der Benutzer soll sämtliche Einstellungen selbstverständlich zu jeder Zeit 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 sein Dateisystem 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 Warteschlangen 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. In dieser Log-Datei werden Nachrichten des Systems gespeichert, um eventuelle Fehler leicht finden und beheben zu können. Es soll zusätzlich der Zustand des Systems abgespeichert werden, wenn das System beendet wird um das System beim nächsten Start in diesen Zustand versetzen zu können.
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 soll über das Unix-System geregelt werden.
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. keine Fehler enthalten. Sie soll aber auch, für den Fall das dennoch Fehler auftreten, robust und tollerant 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 leicht und intuitiv bedienbar sein. Sollte es Funktionen geben, die nicht unter den Begriff SStandardffallen, sollte eine kanppe und präzise Beschreibung der Funktion vorhanden sein. Nach allem dem muss die Software tortzdem noch Effizient sein, geringe Wartezeiten, wenig Hardwareanforderungen, etc. Das Design der Software muss zwar ansprechend sein, ist im edeffekt allerdings zweitrangig.

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. Um einfaches und sicheres arbeiten ermöglichen zu können, soll die Versionsverwaltungssoftware ̈gitbenutzt 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-Client 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.

6 Pflichtenheft

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

- Server-Verbindung
- Client-Einstellungen
- Musik-Steuerung

6.1.3 Wunsch-Kriterien

- Liedinformationen taggen
- Musikstatistik
- Album Covers
- Liedsuche

6.1.4 Abgrenzungskriterien

- Musik-Visualisierung
- Chat
- Social-Network-Schnittstelle

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. Grundlage für die Verwendung der Software ist die General Public License (GPL) 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

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).

Es werden Basiskenntnisse zum Aufbau einer Netzwerkverbindung und zur Nutzung des Internets vorausgesetzt. 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 folgend Punkte erfüllen:

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

6.3 Produktumgebung

6.3.1 Software

- Avahi Daemon
- MPD-Client

Ein MPD-Server ist nicht unbedingt von nöten.

6.3.2 Hardware

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

6.3.3 Orgware

- git (Versionsverwaltungssoftware)
- cmake (Buildsystem)
- doxygen (Dokumentation)
- Editor nach Wahl
- Glade (GUI)

6.4 Produktfunktionen

Funktionen des MPD-Clients.

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 diese denn lokal gefunden werden kann. Der Benutzer soll sämtliche Einstellungen selbstverständlich zu jeder Zeit ändern können.

6.4.1 Starten und Beenden

- F_0010 Der Benutzer kann das System zu jedem Zeitpunkt starten.
- F_0011 Der Benutzer kann das System zu jedem Zeitpunkt beenden.
- F_0012 Beim ersten Start wird ein Standart-System-Zustand geladen.
- F_0013 Beim Beenden wird der aktuelle System-Zustand gespeichert.
- F_0014 Bei jedem weiteren Start wird der letzte System-Zustand geladen.

6.4.2 Abspielen von Musik (Buttons)

- F_0020 Der Benutzer kann Musik abspielen (Play)
- F_0021 Der Benutzer kann Musik stoppen (Stop)
- F_0022 Der Benutzer kann Musik pausieren (Pause)
- F_0023 Der Benutzer kann Musik vor und zurück schalten (Skip)
- F_0024 Der Benutzer kann Musik vor und zurück spuhlen (Seek)
- F_0025 Der Benutzer kann Musik zufällig abspielen (random)
- F_0026 Der Benutzer kann Musik wiederholen (repeat)

- F_0027 Der Benutzer kann Musik im Consume-Mode abspielen
- F_0028 Der Benutzer kann Musik im Single-Mode abspielen

6.4.3 Abspielen von Musik (Short-Cuts)

- F_0030 Play (ctrl + G)
- F_0031 Stop (ctrl + S)
- F_0032 Previous (ctrl + P)
- F_0033 Next (ctrl + N)
- F_0034 Random (ctrl + Z)
- F_0035 Single (ctrl + Y)
- F_0036 Repeate (ctrl + R)
- F_0037 Consume (ctrl + T)

6.4.4 Queue (Warteschlange)

- F_0040 Der Benutzer kann einzelne Lieder aus der Queue entfernen
- F_0041 Der Benutzer kann alle Lieder aus der Queue entfernen
- F_0042 Der Benutzer kann kann die Queue als Playlist speichern
- F_0043 Der Benutzer kann Interpret, Album und Titel beliebig anordnen

6.4.5 Playlist

- F_0050 Der Benutzer kann eine neue Playliste erstellen
- F_0051 Der Benutzer kann eine vorhandene Playlist ersetzen
- F_0052 Der Benutzer kann eine Playlist löschen

6.4.6 Dateibrowser

- F_0060 Der Benutzer kann durch sein Dateisystem navigieren
- F_0061 Der Benutzer kann einzelne Dateien zur Queue hinzufügen
- F_0062 Der Benutzer kann mehrere Dateien zur Queue hinzufügen
- F_0063 Der Benutzer kann Dateien ersetzen
- F_0064 Der Benutzer kann Die Anzeige aktualisieren
- F_0065 Der Benutzer kann die Anzeige neu einlesen

6.4.7 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
  - Letzter Datenbank-Update

6.4.8 Einstellungen

- F_0080 Der Benutzer kann Netzwerk-Einstellungen vornehmen
  - Server IP / Port
  - Reconnect Timout in Sekunden
  - Avahi-Browser (Serverauswahl)
  - Autoconnect
- F_0081 Der Benutzer kann Playback-Einstellungen vornehmen
  - Crossface in Sekunden
  - Musik beim verlassen stoppen
- F_0082 Der Benutzer kann Allgemein-Einstellungen vornehmen
  - Notifications(libnotify) nutzen
  - Tray-Icon anzeigen
- F_0083 Der Benutzer kann die Standarteinstellungen wiederherstellen

6.4.9 Lautstärke

- F_0090 Der Benutzer kann die Lautstärke regeln

6.4.10 Suchen

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 gefilter.

- F_0110 Der Benutzer kann seine Queue durchsuchen.
- F_0111 Der Benutzer kann sein Dateisystem durchsuchen.

6.4.11 Sonstiges

- F_0120 Verbinden (ctrl + C)
- F_0121 Trennen (ctrl + D)
- F_0122 Beenden (ctrl + Q)

6.4.12 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 Produktdaten

6.5.1 Anzeige

6.5.1.1 Titelleiste

- D_0010 Titel
- D_0011 Interpret
- D_0012 Album
- D_0013 Liedposition
- D_0014 Lautstärke

6.5.1.2 Queue

- D_0020 Titel
- D_0021 Interpret
- D_0022 Album

6.5.1.3 Playlist

- D_0030 Name der Playlist
- D_0031 Zuletzt geändert

6.5.1.4 Statistik

- D_0040 Anzahl der Interpreten
- D_0041 Anzahl der Alben
- D_0042 Anzahl der Lieder
- D_0043 Musiklänge der Datenbank
- D_0044 Abspielzeit

- D_0045 Zeit Online bzw. mit MPD verbunden
- D_0046 Letzter Datenbank-Update

6.5.1.5 Fußleiste

- D_0050 Qualität in Mhz
- D_0051 Qualität in bit
- D_0052 Qualität in kbit
- D_0053 Outputart (Stereo, Sourround,...)
- D_0054 Zeit aktuell von insgesamt
- D_0055 Anzahl an Liedern
- D_0056 Komplette Abspielzeit
- D_0057 Lautstärke

6.5.1.6 Sonstiges

- D_0060 Nächster Song (Seitenleiste)

6.5.2 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.

6.5.3 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.

6.5.4 Persönliche Einstellungen

Client Einstellungen werden lokal gespeichert.

- config.xml
- ....

6.6 Qualitätsanforderungen

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

6.6.1 Q_0001 Korrektheit

Die Software muss 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 Q_0002 Wartbarkeit

Der Wartungsaufwandt der Software ist gering bis garnicht 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 Warbarkeit gehört ebenso die Modularität, d. h. die Software ist technisch so realisiert, dass sie leicht erweitert werden kann, Stichwort Model View Controller (MVC). Kritische Stellen werden von Fehlerbehandlungroutinen abgearbeitet. Alle diese Routinen schreiben Meldungen in eine Log-Datei.

6.6.3 Q_0003 Zuverlässigkeit

Das System funktioniert und reagiert tollerant 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 Q_0004 Effizienz

Der MPD-Client ist technisch effizient. 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 Hardware-resourcen.

6.6.5 Q_0005 Benutzbarkeit

Die Software ist leicht verständlich und intuitiv bedienbar. Nötige Kenntnisse zur Nutzung des MPD-Clients sind leicht zu erlernen.

6.6.6 Q_0006 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.

6.7 Globale Testszenarien und Testfälle

6.7.1 Cxxtest

CxxTest ist mit JUnit, CPPUnit oder xUnit zu vergleichen und somit ein leichtgewichtiges Framework für C++. Der Vorteil gegenüber ähnlichen bzw. anderen Testmöglichkeiten sind die folgenden:

- Es wird kein RTTI benötigt (Run-Time Type Information)
- Benötigt keine Member Template Funktionen
- Benötigt keine Exception-Behandlung
- Benötigt keine externen Bibliotheken (Memory Managment, File/Console I/O, Grafische Bibliotheken)
- Wird allein durch Header-Dateien (und ein Python-Skript) realisiert.
- Benötigt keine manuelle Regestrierung von Tests und Test-Suits

All diese Punkte machen CxxTest extrem portabel und nutzbar. Der Aufwandt zur Erstellung von Tests wird minimiert.

6.7.1.1 Testfälle

6.7.2 Testprotokoll

Um Fehler aufzuspühren, die die grafische Oberfläche betreffen, wurd 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.

6.7.2.1 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	Musik spielt ab. Musik stoppt. Musik spielt ab.	Ja
Play, Pause, Play	Musik spiel ab. Musik pausiert. Musik spiel ab.	Ja
Next, Next	Skip weiter. Skip weiter.	Ja
Previous, Previous	Skip zurück. Skip zurück	Ja
Next, Previous	Skip weiter. Skip zurück	Ja
Previous, Next	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

Mehrfache Ausführung:

6.8 Entwicklungsumgebung

6.8.1 Software

- unix System
- MPD-Server
- Avahi-Browser

6.8.2 Hardware

Keine Anforderungen spezifiziert

6.8.3 Orgware

- git (Versionsverwaltungssoftware)
- cmake (Compiler)
- doxygen (Dokumentation)

- Editor nach Wahl
- Glade

6.9 Glossar

7 Design-Dokument

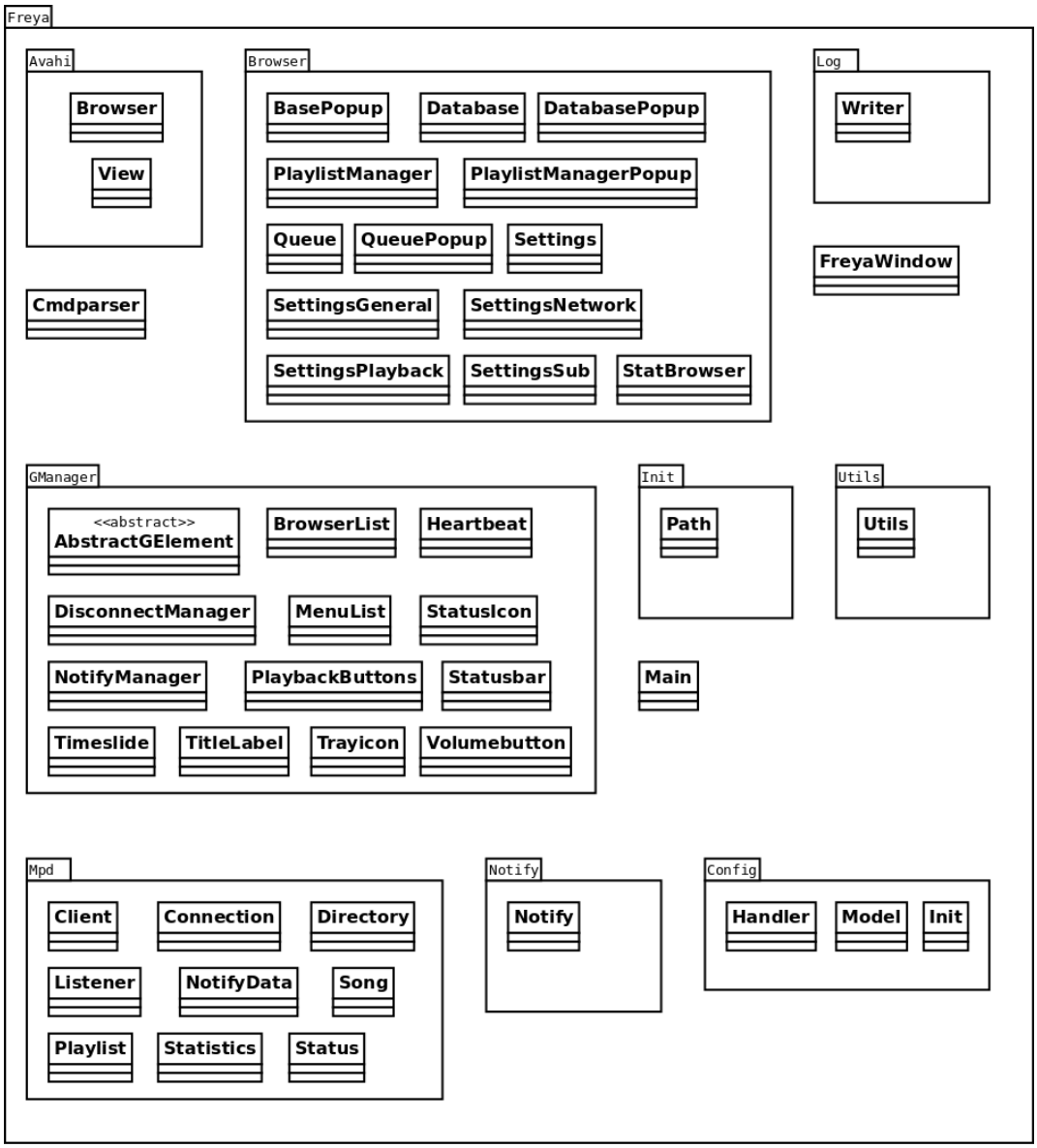
7.1 Einleitung

Dieses Dokument dient der Darstellung der Gesamt-Architektur des Software-Projekts FFreya-MPD-ClientSZunächst werden einige UML-Diagramme zur Übersicht gezeigt und anschließend einzelne Funktionen der Software genauer unter die Lupe genommen.
Im Anschluss daran wird die Benutzeroberfläche präsentiert und in Bereiche aufgeteilt genauer dargestellt. Dabei werden alle Schaltflächen und Info-Panel genau erläutert. Anschließend folgt eine Übersicht zu allen Short-Cuts die genutzt werden können. Danach werden einige Use-Case-Fälle angeschaut, um genau zu verstehen, wie die Software mit dem Nutzer interagiert und auf Befehle reagiert.

7.2 Softwarearchitektur

7.2.1 Architekutrübersicht

7.2.1.1 Namespace-Übersicht



7.2.1.2 Beschreibung

7.2.2 Funktionen

7.2.2.1 Beispiel-Abspielfunktionen

7.2.2.2 Beispiel-Playlist erstellen

7.2.2.3 Beispiel-Dateibrowser

7.2.3 Oberfläche

7.2.3.1 Titelleiste

7.2.3.2 Seitenmenü

7.2.3.3 Fußleiste

7.2.3.4 Anzeige

7.2.4 Short-Cuts

7.3 Use-Case-Fälle

7.3.1 Musik abspielen

7.3.2 Musik zufällig abspielen

7.3.3 Musik im Consume Mode abspielen

7.3.4 Playlist erstellen