Music Player Daemon Client

Software Engineering II Studienarbeit WS 2011/2012, Inf 3

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

9. November 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)
- Netzwerk Programmierung (Verteilte Architektur z.B. Client-Server)
- Grafisches-User-Interface (Swing, Web-basiert,...)

1.2 Prozess-Anforderungen

- Dokumentation aller Phasen(Analyse bis Testen)
- Auswahl eines konkreten Prozessmodells (Begründung der Wahl)
- Erstellung der Dokumente und UML-Diagramme (Freie Wahl der Werkzeuge)
- Fertige Implementierung (Es kann mehr spezifiziert sein als implementiert)
- Spezifikation von Testszenarien (Und Beleg der erfolgreichen Ausführung)
- Lauffähiges System

[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 Fähigkeiten im Bereich des 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

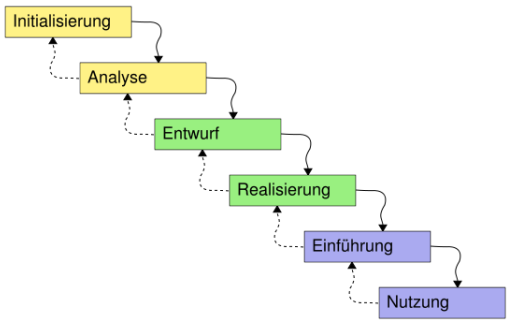
Das Wasserfallmodell ist ein lineares (nicht iteratives) Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Dabei gehen die Phasenergebnisse wie bei einem Wasserfall immer als bindende Vorgaben für die nächsttiefere Phase ein.

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

Der Name „Wasserfall“ kommt 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 „Aufwärtslaufen“ der 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 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.

3.1 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 Standardrechnern 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 existieren 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 Konfiguration 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

3.1.1 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.

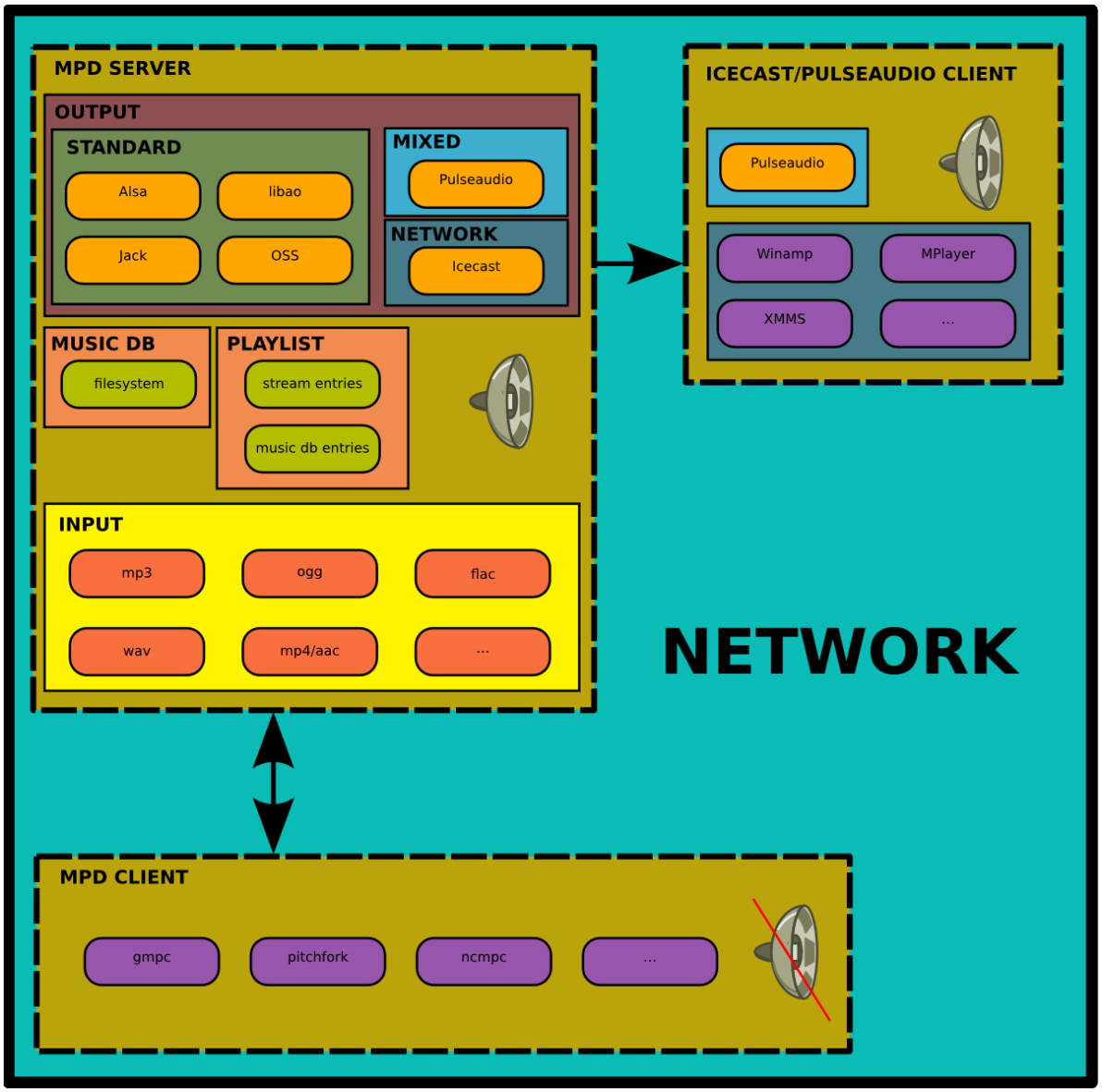
3.1.2 Der MPD kann nicht:

- 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

3.2 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:" erwähnt wurden.

3.3 Grafische Übersicht



2

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

4 Anforderungen

4.1 Anforderungen aus Entwicklersicht

- Testfreundliche Implementierung für Prototypen
- Plug-In-Freundlichkeit
- Menschlich lesbare Configs
- Modular und Plattform-unabhängig
- Model-View-Controller

4.2 Anforderungen aus Anwendersicht

- Standartfunktionen eines Music-Players
- Benutzerfreundlichkeit
- Grafische Oberfläche
- Playlisten erstellen und abspielen
- Musik favorisieren
- Recently-Played Funktion
- Dynamische Playlist
- Band-Informationen aus dem Internet

5 Funktionen des MPD-Client

5.1 MUSS für unseren MPD-Client

- Standartfunktion eines Music-Palyers (Abspielen, Skippen,...)
- Update-Funktion (...)
- Connection-Manager (Server-Client connection)
- Profil basierend (Configs und Playlists einem Benutzer zuordnen)

5.2 KANN für unseren MPD-Client

- Songinformationen und Lyrics aus dem Internet
- Sortierfunktion (Interpret, Album, Titel, Genre)
- Playlist-Verwaltung

:::::::::HIER BEGRÜNDUNG WARUM WIR WAS IMPLEMENTIERT HABEN UND WAS WIR NICHT IMPLEMENTIERT HABEN MIT BEGRÜNDUNG::::::::

6 Abhängigkeiten

6.1 Allgemeine Abhängigkeiten

- gtkmm-3.0
- libmpdclient
- libglyr

6.2 Entwicklungsabhängigkeiten

- git (Versionsverwaltung)
  - git commit -a -m (“Commit-Message“ Übertragen)
  - git push (Änderungen auf den Github-Server laden)
  - git pull (Änderungen von dem Github-Server laden)
- cmake (Buildsystem)
  - cmake . (Buildfiles erstellen)
  - make (Kompilieren)
  - sudo make install (Installieren)
- Editor nach Wahl (gVim, nano, Codeblocks, etc)
- Glade (Oberflächendesigner)
  - Erstellt ein XML File, kann von gtk geladen werden.
  - Callbacks und Signale müssen im Code gehandelt werden.
- Primäre Entwicklerplattform: Linux

BEGRÜNDUNG!!!

7 Programmierrichtlinien

- Programmiersprache C/C++
- Allman-Stil: http://de.wikipedia.org/wiki/Einr
- Tabstop sind 4 Leerzeichen
- Striktes Prototyping
- Keine globalen Variablen
- Sinnvolle Variablenbenennung, Lowercase
- Klassenmethoden nur in Ausnahmen bzw. nur mit guten Gründen
- Valgrind darf keine Laufzeitfehler bringen, auch keine Memory Leaks.
- Camelcase bei Objektnamen, C-Style bei Funktionsnamen
- Modulare Gestaltung - Keinen GUI Code mit FeatureCode misschen
- “make“ sollte auch keine Warnings ausgeben, die man leicht vermeiden könnte
- Testfälle pro Klasse
- Doxygen für Dokumentation des Quelltextes