Multi-User Music Player auf dem RaspberryPi

Musik Abspielen von einem SBC mit Steuerung aus dem Netzwerk dank asynchonen Websockets mit Python

David Döring

▲ *Hochschule Harz*

21. Juli 2017

Inhaltsverzeichnis

1 Einleitung

Der Raspberry Pi und ähnliche Single-board Computer (SBC) sind als vielseitig einsetzbare Linux-Rechner beliebt. Ursprünglich als Lernumgebung für Schulen gedacht haben die kleinen Platinen auch Bereiche wie die Steuerung von Robotern und Internet of Things (IoT)-Geräten erobert. Dank der in den letzten Jahren stark gestiegenen Leistung, einfachen Einrichtung und niedrige Preise auch den Markt der Endbenutzer erreicht. Dort werden sie oft im Betrieb an einem Fernsehgerät zur Verwaltung und Abspielen von Medien eingesetzt. Da besonders die Wiedergabe von Filmen und Spielen dabei im Mittelpunkt steht, ist eine Bedienung über angeschlossene Bildschirme und Eingabegeräte naheliegend. Die Fähigkeit der meisten SBC über lange Zeit mit sehr niedrigem Stromverbrauch betrieben zu werden, geht durch den Bedarf des für die grafische Oberfläche benötigten Bildschirms verloren. Die oft schwachen integrierten Grafikprozessoren stoßen bei der Benutzung vom modernen Bildschirmen mit hoher Auflösung schnell an ihre Grenzen. Einige SBC, wie moderne Netzwerk-Router besitzen keine integrierten Grafikprozessoren. Andere, wie der NanoPi NEO oder OrangePi Zero, bieten nur rudimentäre Videoausgänge und sind damit für den Betrieb als Mediencenter ungeeignet. Vorteile dieser Geräte sind zusätzliche Platzersparnisse und geringerer Stromverbrauch. Sie sind für den Betrieb als headless Computer vorgesehen, das bedeutet, als Computer ohne Bildschirm, Grafische Oberfläche oder Desktopumgebung. Die Steuerung erfolgt ausschließlich über Netzwerke.

Während das Nachrüsten von Videoausgängen oft unmöglich oder sehr umständlich ist, können qualitativ hochwertige Audioausgänge meist problemlos hinzugefügt werden. Darauf wird im Abschnitt 2.1 näher eingegangen. Die Steuerung der Audiowiedergabe ist in diesem Fall zunächst nur über die Konsole aus dem Netzwerk möglich, ist für den allgemeinen Gebrauch jedoch zu kompliziert. Einige bekannte Wiedergabeprogramme bieten bereits Funktionen zur Steuerung über eine Webschnittstelle an. Außerdem gibt es bereits für die Audiowiedergabe optimierte Distributionen für einige SBC, die auch Webfrontends anbieten.

Der eigene Ansatz für die Entwicklung eines Servers für Audiowiedergabe sollte in Python erfolgen. Eine Übersicht zur Wiedergabe von Musik mit Python findet sich in Abschnitt 2.2. Der Server soll über ein Webfrontend steuerbar sein. Für die Kommunikation zwischen Server und Webclient sollen Websockets benutzt werden. Mehr Informationen dazu befinden sich in Abschnitt 2.3.

In Kapitel 3 folgt eine Erläuterung der Umsetzung mit den zuvor beschriebenen Technologien. Abschließend wird in Kapitel 5 auf Probleme und Erweiterbarkeit eingegangen.

2 Grundlagen

2.1 Besonderheiten der Hardware

Viele SBC haben keine analogen Audioausgänge für den direkten Anschluss von Lautsprechern oder Verstärkern. Auch bei Geräten, die über die entsprechenden Anschlüsse verfügen, sind diese meist nicht für eine qualitativ hochwertige Wiedergabe von Musik an Lautsprechern ausgelegt. Seit dem Raspberry Pi Model B verfügen alle Raspberry Pis über eine Möglichkeit, I²S-Module anzuschließen.

Inter-IC Sound (I²S) ist der Standard für die digitale Kommunikation zwischen einer Audioquelle und einem Digital-Analog-Wandler (DAC). Der Digital-Analog-Wandler (DAC) generiert die analogen Wellen, welche in Lautsprechern Schallwellen erzeugen. Die Klangqualität ist zum größten Teil von der Qualität des DAC abhängig. Bei dem Raspberry Pi Model B gibt es neben dem Standard-Pin-Header weitere 8 Pins von welchen 3 für Inter-IC Sound (I²S) genutzt werden können, wenn eine entsprechende Option im Kernel aktiviert wurde. [9] Bei neueren Modellen entsprechen die GPIO-Pins 18, 19 und 20 des Prozessors den benötigten I²S-Pins und können einfach über den Pin-Header des Raspberry Pi an den Pins 12, 35 und 38 erreicht werden. [10] Es werden auch I²S-DAC produziert, welche einfach auf den Raspberry Pi gesteckt werden können. Ein Beispiel ist in Abb. 2.1 auf Seite 5 zu sehen. Auch bei anderen SBC gibt es oft Möglichkeiten, den I²S-BUS zu erreichen. So verfügen Beispielsweise die kostengünstigen und in vielen SBC verwendeten Prozessoren Allwinner H3 und H5 über die entsprechenden Pins.

Bei den meisten Distributionen ist eine Unterstützung von I²S im Kernel vorhanden. Die Pins müssen durch eine Konfiguration beim Boot aktiviert werden.

Neben I²S verfügen die meisten SBC auch über USB-Anschlüsse. Während die Einrichtung des I²S-Bus sich in der Software als aufwändig herausstellen kann und bei mangelnder Dokumentation die benötigten Pins nicht gefunden werden, gibt es die Alternative, einen USB-DAC zu verwenden. Diese werden einfach wie in Abb. 2.2 auf Seite 5 in einen USB-Anschluss gesteckt. Die benötigten Kernelmodule sind in der Regel bereits aktiv.

Angeschlossene Audiogeräte können mit `aplay -l` aufgelistet werden.

2.2 Audiowiedergabe mit Python

Für das Abspielen von Audiodateien mit Python gibt es eine Vielzahl von Bibliotheken, jedoch keine davon hat sich als Standard etabliert. Nachfolgend werden einige dieser

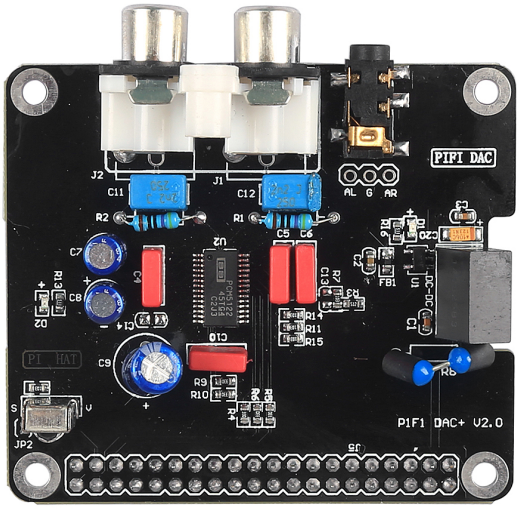

Abbildung 2.1: Ein I^2S-DAC mit passendem Arduino-Header
Quelle: https://www.sainsmart.com/sainsmart-hifi-dac-audio-sound-card-module-i2s-interface-for-raspberry-pi-2-b.html

Abbildung 2.2: Ein USB-DAC an einem Raspberry Pi
Quelle: https://learn.adafruit.com/usb-audio-cards-with-a-raspberry-pi/cm108-type

Bibliotheken mit Vor- und Nachteilen aufgelistet.

2.2.1 Snack

Snack ist ein Toolkit für verschiedene Sprachen, welches das einfache Abspielen und Steuern von Audiowiedergaben ermöglichen soll. Es ist in C geschrieben und verfügt über viele Module die unterschiedliche Dateiformate für Musik unterstützen. [11] Leider ist dieses Toolkit nur mit Python bis Version 2.2 kompatibel. Da in diesem Projekt die Verwendung von Python 3 nötig war, wie in anderen Kapiteln ersichtlich wird, konnte Snack nicht verwendet werden.

2.2.2 PyAudio

PyAudio bietet Zugriff auf die Funktionen der Systemweit verwendbaren PortAudio-Bibliothek. PortAudio dient als Schnittstelle um Programmen den Zugriff auf die Audiohardware zu erleichtern. Der Nachteil der PyAudio-Bibliothek ist der geringe Funktionsumfang von PortAudio. So kann Theoretisch alles darüber abgespielt werden, es sind jedoch keine Decoder enthalten. Um mit PyAudio beispielsweise eine MP3-Datei abzuspielen würde ein Decoder benötigt, mit welchem die Datei eingelesen und an PortAudio übertragen werden kann. Die Recherche und Integration solcher Decoder in das Programm hätte den Rahmen dieser Arbeit weit gesprengt und wurde daher nach einigen Versuchen verworfen. [12]

2.2.3 GST-Python

GST-Python erlaubt Zugriff auf die von vielen Linux-Programmen verwendete GStreamer-Bibliothek. Während GStreamer sehr umfangreich ist und über eine Vielzahl von Funktionen zu verfügen scheint, konnte keine Dokumentation wie diese nun durch GST-Python zu verwenden wären, gefunden werden. Die Implementierung scheint sich nicht nach üblichen Python-Standards zu richten. Möglicherweise ist diese Bibliothek auch nur mit Pyhton 2 kompatibel, wenn auch eine Portierung einfach sein sollte.

2.2.4 PyGame.Mixer

Die Pygame-Bibliothek ist eine umfangreiche Sammlung von Funktionen, welche zur Entwicklung von Spielen mit Python nötig sind. Die Verwendung des Mixer-Moduls sehr einfach. Es ist in der Lage, viele verschiedene Dateitypen zu öffnen und mit wenigen Befehlen abzuspielen. Es werden Playlists und das Abspielen mehrerer Dateien parallel unterstützt. Da es allerdings darauf ausgelegt ist, kurze Sounds in Spielen wiederzugeben, fehlen einige Funktionen, welche für einen Mediaplayer nötig sind. So ist es beispielsweise nicht möglich, die aktuelle Position im laufenden Musiktitel zu bestimmen oder allgemeine Informationen zu den Musikdateien auszulesen. Zudem

ist das Mixer-Modul an die übrige PyGame-Bibliothek gebunden und kann nicht ohne diese installiert werden. Die PyGame-Bibliothek stellt, aufgrund ihres großen Umfangs, viele Anforderungen an das System. So werden für die Installation von PyGame einige Grafik-Module bis hin zu einer Denktopumgebung im Betriebssystem benötigt. Diese Anforderungen sind zu hoch, für den Betrieb auf einem headless SBC. [13]

2.2.5 Steuerung anderer Mediaplayer

Python bietet viele Möglichkeiten, mit anderen Programmen im Betriebssystem zu interagieren. So ist es einem Python-Programm möglich, andere Mediaplayer zu starten und diese entweder über die Kommandos auf der Konsolenebene, simulierte Tasteneingaben oder Socketverbindungen zu steuern. Dieser Ansatz wurde nach der Prüfung vieler anderer Methoden gewählt. Als Mediaplayer wurde der VLC-Player verwendet, da dieser über viele benutzbare Interfaces, einen großen Funktionsumfang, maximale Kompatibilität mit vielen Medienquellen und den betrieb als „VLC-noX“ in headless Systemen erlaubt. [14] Dabei entstand die VLC-Bibliothek, auf welche im Kapitel 3.1 eingegangen wird.

2.3 Websockets

Herkömmliche Client-Server-Kommunikation mit JavaScript erlaubte bisher nur Anfragen vom Client an den Server. In vielen Anwendungen ist dies allerdings nicht ausreichend. So mussten bisherige Webclients die Updates vom Server brauchten, wie beispielsweise in einem Chatprogramm, in regelmäßigen Abständen Anfragen an den Server stellen, um diese Updates zu erhalten. Dies erschwerte die Entwicklung von Client-Server Anwendungen mit Webtechnologien und sorgte für ineffiziente Kommunikation.

Mit Websockets ist es dem Client möglich, einen Kanal zum Server aufzubauen. Über diesen können in beide Richtungen asynchron Daten gesendet werden. Bekommt der Server zum Beispiel durch einen anderen Client ein Nachricht, so kann er alle Clients über die Neuerung informieren. Auf dem Server werden dafür alle Websocket-Verbindungen gespeichert und in jeder auf neue Nachrichten gewartet.

Eine Websocket-Verbindung startet als normale HTTP-Get Anfrage. Der Header dieser Anfrage wird um „pgrade“-Argument mit dem Wert „`websocket`“ erweitert. Der Server antwortet mit einem identischen Header und dem zusätzlichen „`Sec-WebSocket-Accept`“-Argument das als Wert einen Hash der Universally Unique Identifier (UUID) des Clients antwortet. Der Server schließt die Übertragung an dieser stelle allerdings noch nicht ab, sondern behält die offene Verbindung, um in Zukunft darüber weitere Pakete an den Client senden zu können.

Ein Datenpaket im Websocket hat wiederum einen eigenen Header, welcher mit Hilfe einer Maske der Unterscheidung zwischen Server- und Client-Paketen ermöglicht. Der Server maskiert dabei kein Paket, währen der Client alle Pakete maskieren muss.

Für die Maskierung wird auf alle Daten im Paket ein XOR mit einer Maske im Header des Pakets angewendet. [4]

Da die Verbindungen offen bleiben, muss Serverseitig beachtet werden, dass Verbindungen asynchron bearbeitet werden. Bei einer einfachen synchronen Verarbeitung von anfragen, kann ein Server immer nur eine Anfrage zu einem bestimmten Zeitpunkt bearbeiten. Da Websockets als normale HTTP-Anfragen zählen, die allerdings erst bei schließen der Verbindung durch den Client beendet werden, würde es dazu kommen, dass, während ein Client verbunden ist, kein anderer eine Verbindung aufbauen kann. Wie dieses Problem mit Hilfe von Asyncio und AIOHTTP in Pyhton gelöst werden kann, wird in Abschnitt 2.4 beschrieben.

2.4 Python 3.6, Asyncio und AIOHTTP

Python bietet seit Version 3.4 die Asyncio-Standardbibliothek an. Damit ist es möglich, asynchrone, nebenläufige Koroutinen zu implementieren. Im Gegensatz zu sequentieller Programmierung von Aufgaben werden asynchrone Aufgaben gleichzeitig ausgeführt. Dies ermöglicht die Trennung verschiedener Aufgaben voneinander, welche gleichzeitig ausgeführt werden müssen. Verglichen mit Parallelen Prozessen, werden dafür nicht mehrere Prozessoren benötigt. Ein Vorteil von nebenläufigen Prozessen über parallelen Prozessen ist, dass die Kommunikation zwischen Prozessen sowie das Teilen von gemeinsamen Ressourcen deutlich einfacher möglich ist. Bei parallelen Prozessen muss darauf geachtet werden, dass nie 2 Prozesse gleichzeitig auf gemeinsam genutzte Daten zugreifen. Dafür werden Sperren eingesetzt, welche ein Prozess nutzen kann, um andere Prozesse an der Benutzung einer Variable zu hindern, während die vom ersten Prozess verwendet wird. Da bei nebenläufigen Koroutinen immer nur eine Routine zu einem bestimmten Zeitpunkt arbeitet, kann sichergestellt werden, dass nur diese in einem Moment Zugriff auf alle Daten hat.

Dafür wird von einem Prozess-Scheduler immer abwechselnd jeweils einem Prozess die Möglichkeit zur Ausführung gegeben. Dieser muss, wenn er eine Teilaufgabe erledigt hat, die Kontrolle zurück zum Scheduler geben, damit der diese einem anderen Prozess zuweisen kann. Die größte Gefahr die bei der Entwicklung mit Koroutinen besteht darin, dass eine Routine nie die Kontrolle dem Scheduler zurück gibt.

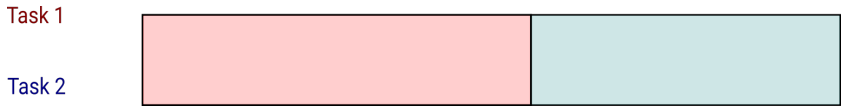

Abbildung 2.3: sequentielle Ausführung von Prozessen

Asyncio wurde speziell mit dem Hintergedanken der gleichzeitigen Verarbeitung mehrerer Ein- und Ausgaben entwickelt. So ist es einem Prozess möglich, auf eine neue Eingabe zu warten. Der Scheduler weist der Wertenden Koroutine nie die Konrtolle zu, bis die erwartete Eingabe erfolgt. So können effizient andere Aufgaben erfüllt werden, während die Koroutine auf eine Eingabe wartet.

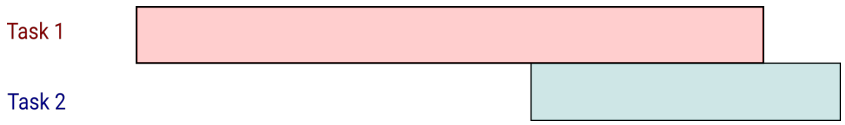

Abbildung 2.4: parallele Ausführung von Prozessen

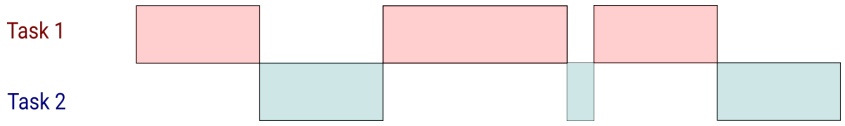

Abbildung 2.5: nebenläufig, asynchrone Ausführung von Prozessen

In Python wird eine Koroutine durch das Schlüsselwort `async` vor der normalen Methodendefinition gekennzeichnet. Durch den Befehl `yield` wird die Kontrolle an den Scheduler zurück gegeben. An dieser Stelle wird die Methode weiter ausgeführt, sobald der Scheduler ihr in Zukunft die Kontrolle zurück gibt. Mit dem Befehl `await` und einer weiteren Koroutine, übergibt die erste Koroutine einer anderen die Kontrolle und wartet darauf, dass diese abgeschlossen wird. Bis zu dem Abschluss kann die untere Koroutine die Kontrolle an den Scheduler zurück geben und tritt allgemein an die stelle der ausführenden Koroutine. Ruft die untere Koroutine während der Ausführung `yield` mit einem Wert auf, so ist es möglich, in der oberen diesen asynchron zu verarbeiten.

Quellcode 2.1: Beispiel einer Asyncio Koroutine

```
async def koroutine(self):
   print("Koroutine gestartet")
   # gebe erstmals die Kontrolle zum Scheduler zurück
   yield
   while True:
      print("Koroutine läuft")
      # Gebe die Kontrolle an den Scheduler zurück
      #  und warte auf den Abschluss der in asyncio
      #  vordefinierten sleep-Koroutine.
      # Sleep wartet n=5 Sekunden.
      await asyncio.sleep(5)
```

2.4.1 AIOHTTP

AIOHTTP ist enthält Implementierung verschiedener Hypertext Transfer Protocol (HTTP)-Clients und -Server, welche mit Asyncio kompatibel sind. So können Anfragen asynchron verarbeitet werden, was die gleichzeitige Benutzung eines Servers durch mehrere Clients oder die Bereitstellung mehrerer Server erlaubt. Auch eine Implementierung des WebSocket-Protokolls ist enthalten. Speziell für dieses ist die asynchrone Ausführung wichtig, wie in Abschnitt 2.3 erklärt wurde.

3 Umsetzung

Für die Umsetzung wurde zunächst eine Bibliothek zur Steuerung des VLC-Mediaplayers implementiert. Diese wird von der Implementierung des Websocket-Servers benutzt, welcher einen Websocket bereit stellt, über welchen wiederum der Mediaplayer gesteuert werden kann. Theoretisch wäre es möglich die Bibliothek zur Steuerung des VLC-Mediaplayers durch eine andere mit ähnlicher Funktionalität zu ersetzen, um andere Mediaplayer oder eine eigene, python-basierte Implementierung zum Abspielen von Audiodateien über den Websocket zu steuern. Bei der Implementierung des Websocket-Servers wird ausgenutzt, dass AIOHTTP mehrere Prozesse gleichzeitig ausführen kann (siehe Abschnitt 2.4) und Websockets auf normalen HTTP-Get-Anfragen basieren (siehe Abschnitt 2.3) um neben dem Websocket auch die für den Webclient benötigten Dateien bereitszustellen. Der Webclient ist in modernem HTML5 mit HTML-Imports umgesetzt und baut eine Verbindung zum Websocket auf, über welchen er Updates des Players erhält und Nutzereingaben an den Server überträgt. In Abbildung 3.1 wird zur Verdeutlichung der Verbindungen ein Ablauf mit zwei Clients dargestellt. In der Abbildung wird nur die Position im aktuellen Titel betrachtet. In der Praxis wären neben der Position auch äquivalente Abläufe für andere Elemente, wie die Playliste, den aktuellen Titel oder das aktuelle Albumbild nötig.

Der Mehrwert des asynchronen Websockets und des neuen Webfrontends gegenüber der HTTP-Schnittstelle und dem Webfrontend des VLC-Players besteht darin, dass diese Implementierung asynchron arbeitet. Das Webfrontend des VLC-Players kann nicht von mehreren Nutzern gleichzeitig fehlerfrei genutzt werden. Die Implementierung dieses Websockets ermöglicht die Bündelung der Anfragen mehrerer Nutzer, leitet diese an den sequentiell arbeitenden HTTP-Server des VLC-Players weiter und informiert alle Nutzer des Websockets asynchron über Neuerungen am Player.

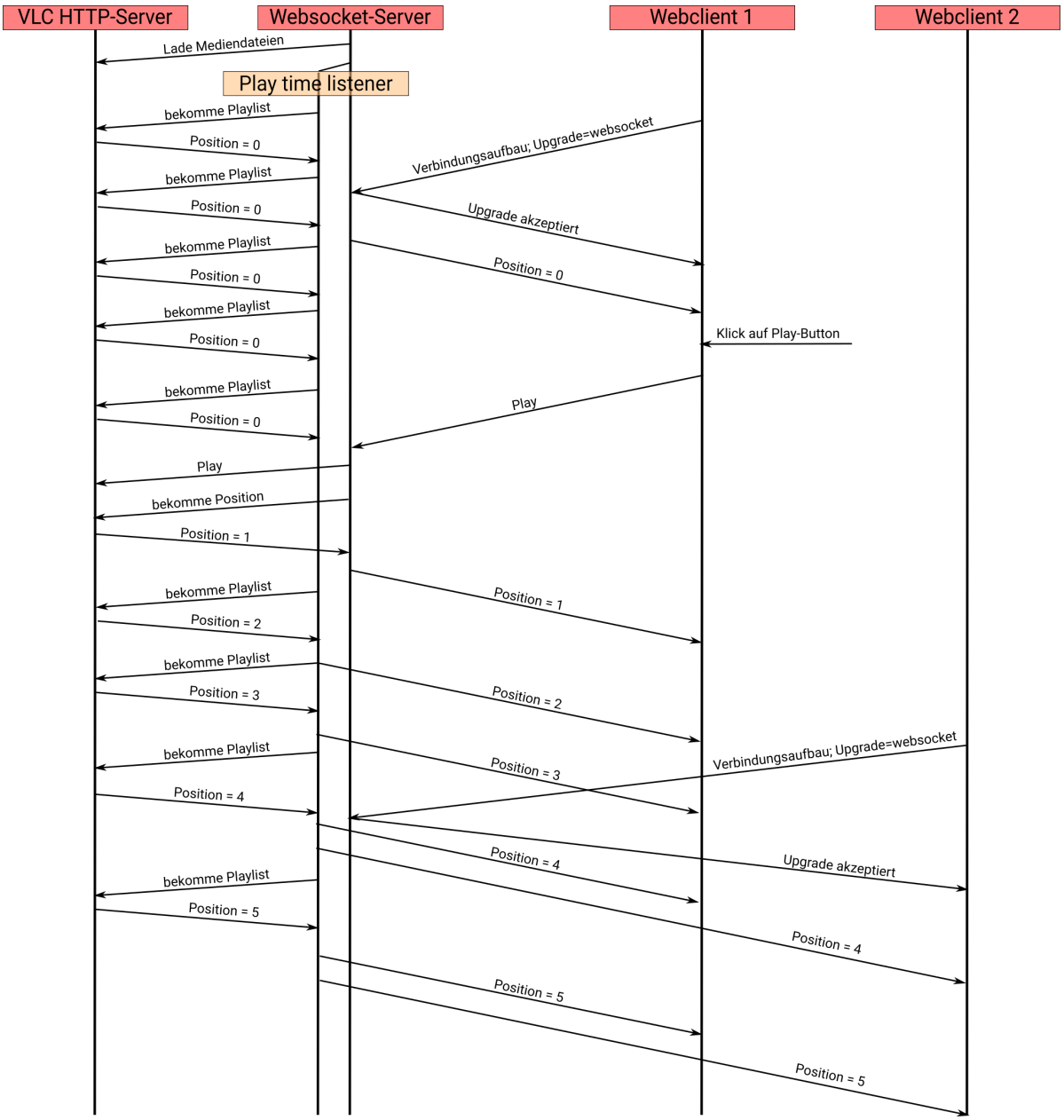

Abbildung 3.1: Sequenzdiagram der Verbindungen von zwei Clients mit dem Websocket

3.1 VLC-Bibliothek

Die VLC-Bibliothek prüft beim Ausführen, ob bereits eine Instanz des VLC-Players ausgeführt wird. Ist dies nicht der Fall, so startet sie den VLC-Mediaplayer mithilfe der seit Python 3.0 vorhandenen subprocess-Bibliothek in einem externen Prozess aus. Der VLC-Player wird in einem Linux-Screen (virtuelle Konsolensitzung) ausgeführt. Dies hat den Vorteil, dass bei Neustart des Servers der bereits laufende VLC-Prozess einfacher wiedergefunden werden kann. Ein Nachteil ist allerdings, dass diese Methode nur unter Linux funktioniert. Wird der VLC-Player bereits ausgeführt, so wird lediglich eine Verbindung zu diesem aufgebaut.

Diese Bibliothek hat starke Einflüsse durch ein von Marios Zindilis entwickeltes Skript zur Kontrolle des VLC-Players von. Das ursprüngliche Skript war für die Benutzung in Python 2 geschrieben und konnte den VLC-Player nur über das RC-Interface steuern. Auch bei dieser Steuerung gab es erhebliche Probleme, da kaum Buffering betrieben wurde und viele Funktionen, welche vom VLC-Player angeboten werden, nicht genutzt wurden. [2] Das ursprüngliche Skript wird durch das neue Skript um mehr Möglichkeiten beim Starten des VLC-Players, der Unterstützung des HTTP-Interfaces sowie ein überarbeitetes Buffering der Antworten des VLC-Players erweitert und im Python 3 implementiert.

Der Verbindungsaufbau zum VLC-Player ist momentan nur über das RC- und das HTTP-Interface möglich. Das RC-Interface wird momentan noch nicht vollständig von dieser Bibliothek unterstützt. Daher ist eine Verbindung über das HTTP-Interface zu empfehlen.

Das RC-Interface des VLC-Players ist auf die Benutzung durch eine Kommandozeile und für Verbindungen zum RC-Socket über ein Netzwerk vorgesehen. Wird das RC-Interface verwendet, so baut die VLC-Bibliothek bei Initialisierung eines VLC-Objektes eine Verbindung zu dem Socket auf. Dafür wird die in Python vorhandene Socket-Bibliothek verwendet. Nach dem Verbindungsaufbau können Befehle an den Socket gesendet werden. Anschließend kann durch das Auslesen des Socketbuffers die Antwort des VLC-Players bestimmt und für die Benutzung in Pyhton passend formatiert werden. Die verwendbaren Befehle können eingesehen werden, indem der VLC-Player auf der Kommandozeile mit dem RC-Interface ausgeführt wird und der Help-Befehl eingegeben wird. Alle wichtigen Befehle werden in der Bibliothek durch Methoden wiedergegeben. Der Anwender der VLC-Python-Klasse braucht kein weiteres Wissen über die verwendeten Kommandos.

Die Verbindung zum HTTP-Interface erfolgt über die Requests-Bibliothek. Es werden normale HTTP-Get-Anfragen an die HTTP-Schnittstelle des VLC-Players gestellt. Die Antworten werden im JSON-Format geladen und automatisch in Python-Objekte umgewandelt. Eine Dokumentation der möglichen Anfragen und zu erwartenden Antworten kann in den Dateien des VLC-Players gefunden werden. Auch in dem Repository des Quellcodes des VLC-Players sind sie vorhanden. [15]

Unabhängig davon, welches Interface beim Laden des VLC-Objektes gewählt wurde, können alle funktionen ausgeführt werden. Diese werden intern an das zu

verwendende Interface weitergeleitet.

Der Code der VLC-Bibliothek kann in der Datei `vlc.py` im Ordner `vlc` eingesehen werden.

3.2 Asynchroner Websocket-Server in Python

Das Server-Skript startet zu Beginn einen Asynchronen Server der AIOHTTP-Bibliothek und führt zusätzlich eine Liste von Koroutinen aus, die jeweils in bestimmten Abständen über die VLC-Bibliothek Anfragen an den VLC-Player stellen um Informationen von diesem zu erhalten. Da der VLC-Player keine Asynchrone Verbindung anbietet, ist nur dieses regelmäßige Überprüfen möglich.

Dies ist auch ein Grund für die Implementierung des Websockets, da ohne diesen das bereits vorhandene Webfrontend des VLC-Players nicht Asynchron funktioniert. Durch die regelmäßigen Anfragen des Websocket-Servers an den VLC-HTTP-Server und das anschließend asynchrone Verteilen an alle Clients wird aus dem synchronen Single-User-System ein asynchrones System, welches mehrere Nutzer erlaubt.

Der zu Beginn gestartete AIOHTTP-Server prüft bei jeder Anfrage, ob diese ein Upgrade zum Websocket anfordert (siehe Abschnitt 2.3. Wird kein Upgrade der Verbindung gefordert, werden die Dateien eines Unterverzeichnisses bereitgestellt, welche vom Browser des Nutzers geladen und als Webclient ausgeführt werden können (siehe Abschnitt 3.3).

Wird ein Upgrade zum Websocket gefordert, so wird die Anfrage mit dem Websocket-Header beantwortet und die Koroutine am Leben erhalten, bis die Verbindung vom Client geschlossen wird. Die offenen Websocket-Verbindungen werden in einer gemeinsamen Liste gespeichert, um den Zugriff durch andere Koroutinen zu erlauben. Bei Verbindungsabbau werden die Verbindungen aus der Liste gelöscht. Bei eingehenden Anfragen durch den Websocket werden diese in einer `on_ws_message`-Methode interpretiert und auf Befehle an das VLC-Objekt umgesetzt. Neuerungen am VLC-Objekt werden anschließend an alle verbundenen Clients übertragen.

Der Quellcode des Servers befindet sich in der Datei `serve.py` im Hauptordner des Projekts. Diese Datei muss gestartet werden, um das gesamte Projekt auszuführen. Sie Erlaubt das Anhängen von Parametern wie die Position der Abzuspielenden Mediendateien oder der vom Server zu verwendenden Ports.

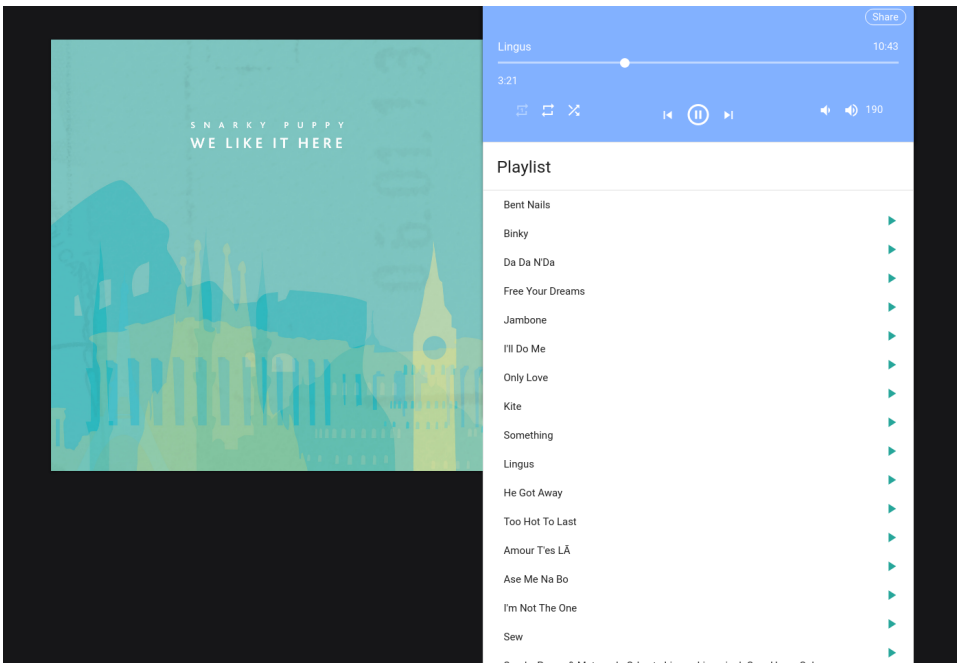

Abbildung 3.2: Ansicht des Webfrontends

3.3 Webclient

Der Webclient besteht aus 3 Modulen. Diese werden mit Hilfe von HTML-Imports miteinander verbunden. HTML-Imports erlauben eine Objektorientierte Strukturierung von HTML5-Projekten. Dabei wird in jedem Modul der benötigte HTML-Code in einem Template gespeichert und mit dazu gehörigem Javascript geladen und ausgeführt. [6] Ein Beispiel für ein selbst definiertes HTML-Element in einer Datei, welche importiert werden kann ist im Quellcode 3.1 auf Seite 15 zu sehen. Dabei werden eigene HTML-Tags definiert, welche mit `<link rel="import" href="Quelle.html"/>` in anderen HTML-Modulen geladen und anschließend als eigener HTML-Tag in die Seite eingefügt werden können.

Das Webfrontend basiert auf HTML-Code von Uğur Oruç und wurde restrukturiert sowie durch Funktionalität im JavaScript ergänzt. [1]

Das Hauptelement des Webclients stellt eine Verbindung zum Websocket her und zeigt dem Nutzer alle Steuerungselemente. Es implementiert die Funktionalität dieser Elemente und sendet bei Nutzereingaben Befehle über den Websocket an den Server. Bei eingehenden Befehlen vom Server werden diese auf die damit verbundenen Elemente in der HTML-Seite angewendet.

Das Hauptelement `<music-app>` läd die Definition des `<song-list>`-Elements und legt eine Instanz davon an. Dem `<song-list>`-Element wird beim empfangen einer neuen Playlist vom Server diese zusammen mit dem Template einer Funktion zum Wechseln des aktuellen Titels übergeben. Im `<song-list>`-Element wird für jeden Titel in der Playlist ein `<song-entry>`-Element angelegt. Diesem wird die passende Funktion zum

wechseln des Titels übergeben. Wird ein Button im `<song-list>`-Element angeklickt, so kann dieses die Funktion ausführen und der Titel wird gewechselt.

Der Quellcode des Webclients befindet sich im Unterordner `webclient`. Es ist zu beachten, dass dieser in einigen Browsern, inklusive dem Internet Explorer von Microsoft sowie in Firefox angezeigt wird. Diese Browser unterstützen das neue Konzept der HTML-Imports aktuell noch nicht. Die Funktion des Webclients wurde bisher nur in der aktuellen stabilen Version des Chromium-Browsers unter Linux und Android getestet.

Quellcode 3.1: Beispiel eines eigenen HTML-Elements, welches als Import geladen werden kann

```
<!DOCTYPE html>
<template id="halloWeltTemplate">
   <div><i>Hallo Welt!</i></div>
</template>
<script>
   var HalloWeltProto = Object.create(HTMLElement.prototype);
   HalloWeltProto.ownerDoc = document.currentScript.ownerDocument;

   HalloWeltProto.createdCallback = function () {
      // get reference to main document
      var template = this.ownerDoc.querySelector('#halloWeltTemplate');
      var instanceClone = document.importNode(template.content, true);
      this.appendChild(instanceClone);
   };
   HalloWeltProto.attachedCallback = function () {
      // Wird ausgeführt, wenn das Element dargestellt wird
   };
   HalloWeltProto.detachedCallback = function () {
      // Wird ausgeführt, wenn das Element entfernt wurde
   };
   HalloWeltProto.attributeChangedCallback = function (attrName, oldVal, newVal) {
      // Wird ausgeführt, wenn sich ein Attribut des Elements ändert
   };
   var HalloWelt = document.registerElement('hallo-welt', {prototype: HalloWeltProto});
   // Das Element kann jetzt als <hallo-welt></hallo-welt> eingefügt werden.
</script>
```

4 Installation

Zu den Bedingungen zählen zunächst der VLC-Player oder für den Betrieb auf einem headless Computer VLC-noX. Außerdem wird Python 3.6 benötigt. Die Installation der Abhängigkeiten in Debian basierten Systemen, wie Raspbian oder Armbian mit folgendem Befehl möglich:

```
sudo apt-get install vlc-nox python36
```

Optional kann auch `git` über `apt-get` installiert werden, um später den Quellcode des Projekts von Github zu laden.

Anschließend müssen einige Abhängigkeiten in Python installiert werden. Hierfür kann `pip` verwendet werden. Sind mehrere Versionen von Python installiert und Python 3.6 ist nicht als Standardversion gewählt, muss der Befehl `pip3.6` benutzt werden. In diesem Fall bietet sich auch an, eine virtuelle Umgebung für Python 3.6 anzulegen und in diese zu wechseln.

```
(sudo) pip install requests aiohttp
```

Ist der Quellcode des Projekts noch nicht vorhanden, so kann er über Git geladen werden:

```
git clone --recursive https://github.com/Anaeijon/Monty_Mumup.git
```

Das Argument `--recursive` ist dabei wichtig, um alle verwendeten Submodule automatisch zu laden.

Im Hauptordner des Projekts kann jetzt das `serve.py`-Script ausgeführt werden. Es benötigt einen Ordner als Parameter, in welchem nach Musik gesucht werden soll.

```
python ./serve.py ~/Pfad/zu/Musik
```

Anschließend wer Webclient über den Port 8000 erreichbar.

Das Projekt bietet auch die Möglichkeit, einen anderen Port über das Argument `--recursive` anzugeben. Um von anderen Geräten aus auf den Server zugreifen zu können, wird empfohlen, Port 80 zu verwenden. Da in den meisten Linux-Systemen die Benutzung dieses Ports dem Superuser `root` vorbehalten ist, muss der Server mit `sudo` ausgeführt werden. Da der VLC-Player allerdings nicht vom `root` ausgeführt werden darf, muss er zuvor separat vom Nutzer gestartet werden.

```
screen -dmS vlc_screen vlc --intf http --http-host localhost --http-port 8080 \
                       --http-password pass
sudo python ./serve.py --port 80 --vlc-screen None --vlc-port 8080 --vlc-password pass
```

Mit dem Argument `--extraintf` können weitere Interfaces zu VLC hinzugefügt werden.

5 Ausblick

5.1 Fehler und Probleme

Aktuell kann es in der Kommunikation mit dem VLC-Player zu Fehlern kommen. Probleme mit dem Buffering der Antworten aus dem RC-Interface sind bekannt. Daher wird bevorzugt das HTTP-Interface verwendet. Doch auch im HTTP-Interface kommt es dazu, dass der VLC-Player teilweise veraltete Informationen sendet.

Einige Funktionen des Webfrontends, wie das Ändern der Position im Titel mit dem Slider-Element können dafür sorgen, dass viele Befehle gleichzeitig übertragen werden. Es kommt dadurch zu einem Stau im Websocket-Server, welcher diese an den VLC-Player weiterleiten soll. Dadurch kann es zu großen Latenzen zwischen Nutzereingabe und Reaktion des Servers kommen.

5.2 Erweiterbarkeit

Es ist möglich, das verwendete VLC-Modul zu ersetzen. Da im Hauptprogramm keine VLC-Spezifischen Funktionen genutzt werden, kann an dieser Stelle auch ein anderer Medienplayer eingesetzt werden. Dieser muss nur die Grundfunktionen, welche vom Webfrontend angeboten werden, wie Play, Pause, Nächster Titel und weitere bereitstellen und durch eine Klasse in Python gesteuert werden. Auch eine eigene Implementierung einer Medienplayer-Klasse in Pytho, beispielsweise auf Basis der GStreamer-Bibliothek (siehe Abschnitt 2.2.3) wäre möglich.

Die eigene Bibliothek zur Steuerung des VLC-Players ist nicht mit Asyncio entwickelt wurden. Dadurch kann es zu Latenzen während der Kommunikation mit dem VLC-Player kommen. Um dies zu vermeiden, müssten die verwendeten HTTP-Get-Anfragen aus der Requests-Bibliothek durch entsprechende Anfragen aus der AIOHTTP-Bibliothek ersetzt und alle Methoden der VLC-Bibliothek zu Koroutinen erweitert werden. Aktuell ist ein Vorteil der sequentiellen Implementierung der Anfragen an den VLC-Player, dass nicht sicher gestellt werden kann, ob dieser in der Lage ist, auf asynchrone Anfragen korrekt zu reagieren.

Die Python-Implementierung dieses Websockets würde Obsolet werden, wenn der VLC-Player neben dem HTTP-Interface auch noch um ein Websocket-Interface erweitert würde. Die Implementierung müsste in Lua oder C geschehen.

Aktuell kann nur beim Start des Servers ein Ordner angegeben werden, welcher vom Player geladen und als Playlist angezeigt wird. Es ist nicht möglich, weitere Titel hinzuzufügen. Diese Funktion sowie das Wechseln zwischen verschiedenen

vordefinierten Playlists sollte in Zukuft über das Webfrontend verfügbar gemacht werden. Bis dahin kann der VLC-Player neben dem HTTP-Interface mit einem zusätzlichen Interface wie „ncurses“ oder „telnet“ ausgeführt werden, über welche ein Administrator weitere Titel zur Medienliste hinzufügen kann.

5.3 Fazit

Auch wenn der verwendete VLC-Player bereits über ein eigenes Webfrontend verfügt, so bietet diese Implementierung einen Mehrwert. Nicht nur das Design sondern auch das Konzept der Client-Server-Kommunikation über Websockets ist dem alten Überlegen. Gerade bei der Benutzung durch mehrere Nutzer oder von mehreren Interfaces kommt es seltener zu Fehlern im Webclient.

Bei der Entwicklung sind der Webclient und die VLC-Bibliothek als eigenständige Module entstanden, welche auch in anderen Projekten verwendet werden können.

6 Glossar

DAC Digital-Analog-Wandler (engl. Digital-Analog-Converter) sind Module, welche digitale zu analogen Signalen umsetzen. Sie werden hauptsächlich eingesetzt um in elektronischen Geräten analoge Signale für Lautsprecher zu erzeugen, welche als Schallwellen wiedergegeben werden. 4

Distribution Eine Distribution ist eine Bündelung von Software, welche einander benötigt, um sinnvoll genutzt werden zu können. Meist wird der Begriff benutzt um verschiedene Varianten von Linux-Systemen zu beschreiben, die neben dem Linux-Kernel noch weitere Softwarepakete enthalten um als vollständiges Betriebssystem nutzbar zu sein. 3, 4

Hash Eine Hashfunktion bildet eine große Datenmenge auf eine kleinere ab. Es ist damit möglich, bis zu einem gewissen Grad an Sicherheit, eine Eingabe eindeutig zu identifizieren, ohne die Eingabe vollständig speichern oder übertragen zu müssen. 7

headless Computer, welche über keine grafische Oberfläche verfügen werden als „headless“ bezeichnet. 3, 7, 16, 19

HTML Hypertext Markup Language (HTML) ist eine Extensible Markup Language (XML)-basierte Auszeichnungssprache. Sie die Grundlage für Seiten im World Wide Web (WWW) und wird von Webbrowser interpretiert. 20

HTTP Das Hypertext Transfer Protocol (HTTP) ist das Übertragungsprotokoll, welches genutzt wird, um im WWW Webseiten von einem Server zu laden. 9, 20

I²S I²S steht für Inter-IC Sound und ist ein Standard für die digitale Übertragung von Tönen zwischen Prozessoren. 4

IoT Eingebettete Computer in alltäglichen Gegenständen können Daten sammeln und sich über das Internet vernetzen, um den Nutzer über die gewohnte Funktionalität hinaus zu unterstützen, dabei jedoch nicht aufzufallen. 3

Open-Source Software deren Quellcodes oder Hardware deren technische Spezifikationen und Layouts von jedem eingesehen, verbreitet, modifiziert und genutzt werden können wird als Open-Source bezeichnet . 19

Raspberry Pi Eine sehr bekannte Reihe von Open-Source SBC welche durch die Raspberry Pi Foundation entwickelt werden. 3–5

SBC Ein Computer, bei welchem alle für den Betrieb benötigten Komponenten auf einer Platinen zusammengefasst sind, wird als Single-board Computer (SBC, Einplatinencomputer) bezeichnet. 3, 4, 7, 19

URL Uniform Resource Locators (URLs) (engl. einheitlicher Ressourcenzeiger) dienen im WWW der eindeutigen Definition und referenzierung von Ressourcen. 20

UUID Ein Universally Unique Identifier ist eine 128-Bit-Zahl die dazu dient, Computer und Daten weltweit eindeutig zu identifizieren. Durch den verwendeten Prozess bei der Erstellung einer UUID soll mit annähernd absoluter Sicherheit garantiert werden, dass es unter UUIDs keine Duplikate gibt. 7

Webbrowser Programm zur Darstellung von Webseiten. 19, 20

Webseite Über Uniform Resource Locators (URLs) erreichbares Dokument, das mit Hypertext Markup Language (HTML) formatiert wurde und von Webbrowser aufgerufen werden kann. 19, 20

Websocket Websockets sind eine Erweiterung des klassischen HTTP-Protokolls, die eine bidirektionale Kommunikation zwischen Client und Server ermöglichen. 3, 7

WWW Das World Wide Web (WWW) besteht aus Webseiten, die per Hyperlinks miteinander verknüpft sind und über das Internet abgerufen werden können. 19, 20

XML Extensible Markup Language (engl. Erweiterbare Auszeichnungssprache). 19

XOR Eine XOR-Verknüpfung bezeichnet eine bitweise Operation bei der eine exklusive Disjunktion angewendet wird. Bitweise werden beide Eingaben verglichen. Sind die Eingaben an dieser Stelle gleich, so ist das Resultat an der Stelle 0. Unterscheiden sich die Eingaben, ist das Resultat an der Stelle 1. Beide Eingaben produzieren gemeinsam eine Ausgabe. Wendet man XOR auf eine der Eingaben und die vorherige Ausgabe an, so ist die andere Ausgabe das Resultat. 8

7 Literaturverzeichnis

[1] Uğur Oruç: ***Materialize-Music-Player***
Quelcode: https://github.com/Ketcap/Materialize-Music-Player

[2] Marios Zindilis: ***Control VLC with Python***
https://zindilis.com/blog/2016/10/23/control-vlc-with-python.html

[3] J. Ernesti, P. Kaiser: ***Python 3 : das umfassende Handbuch***
4., aktualisierte und erweiterte Auflage 2015, 1., korrigierter Nachdruck 2016
Bonn : Rheinwerk Verlag, 2016

[4] I. Fette, A. Melnikov: ***RFC 6455 : The WebSocket Protocol***.
Internet Engineering Task Force (IETF), December 2011
https://tools.ietf.org/html/rfc6455

[5] W3C: ***HTML5, A vocabulary and associated APIs for HTML and XHTML***.
W3C Recommendation 28 October 2014
https://www.w3.org/TR/html5/

[6] W3C: ***Custom Elements***.
W3C Working Draft 26 February 2016
https://www.w3.org/TR/2016/WD-custom-elements-20160226/

[7] Python Software Foundation: ***Python Language Reference****, Version 3.6.1*.
https://docs.python.org/3.6/

[8] Python Software Foundation:
asyncio — Asynchronous I/O, event loop, coroutines and tasks*, Version 3.6.1*.
https://docs.python.org/3/library/asyncio.html

[9] Torsten Jaekel: ***Audiophile, High Quality RPi-DAC for Raspberry Pi***
http://www.tjaekel.com/T-DAC/raspi.html

[10] Daniel Smith: ***I2S Audio DAC + Amp for Raspberry Pi 2 / B+***
http://www.pagemac.com/projects/i2s_amp

[11] Kåre Sjölander: ***The Snack Sound Toolkit***
http://www.speech.kth.se/snack/

[12] ***PyAudio Documentation***
http://people.csail.mit.edu/hubert/pyaudio/docs/

[13] ***Pygame documentation - pygame.mixer***
https://www.pygame.org/docs/ref/mixer.html

[14] ***vlc-noX: VLC: VideoLAN Client - without X dependencies***
https://download.videolan.org/vlc/0.8.6/SuSE/11.1/repodata/repoview/vlc-noX-0-1.1.2-2.9.html

[15] ***VLC http requests README.txt***
https://github.com/videolan/vlc/blob/master/share/lua/http/requests/README.txt