HTL anichstraße
Biomedizintechnik · Elektronik · Elektrotechnik · Maschinenbau · Wirtschaftsingenieure

htl bildung mit zukunft

Diplomarbeit

Bidirektionale Videosprechanlage

Erweiterung einer Raspberry Pi basierten Videogegensprechanlage

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstraße

Abteilung Elektrotechnik

Ausgeführt im Schuljahr 2019/20 von:

Andreas Grain 5AHET (HV)
Matthias Mair 5AHET

Betreuer:

DI(FH) Mario Prantl

Innsbruck, am 2020-03-06

Abgabevermerk:

Betreuer:

Datum:

Erklärungen

Eidesstattliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht haben.

Ort, Datum	Andreas Grain
Ort, Datum	Matthias Mair

Gender-Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Inhaltsverzeichnis

Kurzfassung/Abstract

Deutsch

Die vorliegende Diplomarbeit beschäftigt sich mit der Erweiterung einer RPi-basierten Videosprechanlage um eine Smartphone-Applikation, sowie der grundlegenden Überarbeitung der Stations-Hardware. Zusätzlich soll ein Linux-basierter, aus dem Internet erreichbarer Server zum Verteilen der Video-Streams eingerichtet werden.

Dieses Projekt basiert auf einer früheren Diplomarbeit von Sebastian Wagner und Tobias Pfluger an der HTL Anichstraße aus dem Jahr 2017/18, welche ein voll funktionsfähiges Videosprechanlagensystem hervorbrachte. Jedoch wurde damals die Hardware-Entwicklung sehr vernachlässigt.

Die Hardware-Erweiterung besteht grundsätzlich aus zwei Teilen: zum einen werden die vielen Elektronik-Bausteine in einer zentralen Platine vereint, was eine einfachere und kostengünstigere Fertigung erlaubt. Zusätzlich wird die aktuelle Hardware der Station um einen Watchdog-Timer erweitert, welcher im Fehlerfall die Anlage zurücksetzt.

Softwareseitig wird die Anlage um eine Smartphone-App ausgebaut, mit welcher der Fernzugriff auf das System ermöglicht werden soll.

English

This diploma thesis deals with the extension of a RPi-based video intercom system by a smartphone application, as well as the fundamental revision of the station hardware. In addition, a Linux-based server, accessible from the Internet, will be set up to distribute the video streams.

This project is based on an earlier diploma thesis by Sebastian Wagner and Tobias Pfluger at the HTL Anichstraße from the year 2017/18, which produced a fully functional video intercom system. However, at that time the hardware development was rather neglected.

The hardware extension basically consists of two parts: on the one hand, the many electronic components are combined in a central circuit board, which allows easier and more cost-effective production. In addition, the current hardware of the station is extended by a watchdog timer, which resets the system in case of an error.

On the software side, the system will be extended by a smartphone app, which will enable remote access to the system.

Projektergebnis

1 Einleitung

Jedes moderne Wohngebäude ist inzwischen mit einer Gegensprechanlage ausgestattet. Eine solche Anlage erlaubt es dem Wohnungsbesitzer mit jemandem vor der Haustür zu reden, bevor dieser hereingelassen wird. Manche Wohnungen besitzen bereits eine Videosprechanlage, die nicht nur den Ton, sondern zusätzlich ein Video von außen dem Bewohner bereitstellt. Dem Gast wird jedoch immer noch nur der Ton von innen übertragen. Diese Applikation ist ortsgebunden, d.h. es müssen sowohl der Gast, als auch der Bewohner vor der entsprechenden Station stehen.

Im Zuge einer früheren Diplomarbeit des Schuljahres 2017/18 an der HTL Anichstraße entwickelten Sebastian Wagner und Tobias Pfluger eine Videogegensprechanlage, welche mehrere Innenstationen und die bidirektionale Video- und Tonübertragung unterstützt. Die Idee ist, dass in einem Wohnungskomplex pro Partei eine Innenstation verbaut wird, sowie eine Außenstation am Hauseingang. Die einzelnen Innenstationen, also Wohnungsparteien, können dann nicht nur mit der Außenstation, sondern auch mit anderen Innenstationen per Video und Ton kommunizieren. Die Stationen wurden mit Hilfe eines RPi realisiert, um die Kosten gering zu halten.

Am Anfang des 5. Schuljahres im Herbst 2019 bat uns unser FI-Professor DI(FH) Mario Prantlan, dieses Projekt durch eine Smartphone-App und eine Hardware-Überarbeitung zu verbessern. Für die Entwicklung der App sollte das Xamarin-Rahmenwerk verwendet werden, damit die Applikation auf sowohl Android-, als auch iOS-Geräten lauffähig ist. Der Benutzer soll über die App das Video der entsprechenden Station abrufen und mit dem Gesprächspartner sprechen können. Hierfür wird ein aus dem Internet erreichbarer Medien-Verwaltungsserver benötigt, der die Medien-Streams der Einzelstationen empfängt und an die Station des Gesprächspartners überträgt. Die Hardware-Überarbeitung hat mehrere Ziele; zum Einen soll dadurch die Komplexität des internen Aufbaus einer Station verringert werden. Zum Anderen soll mit Hilfe eines Watchdog-Timers das Langzeit-Betriebsverhalten verbessert werden.

1.1 Funktionalität

Ortsunabhängiger Anlagenzugriff: Um dem Anwender mehr Komfort bieten zu können, beschränkt sich die Anlage und dessen Funktion nicht mehr auf die fest installierte Station, sondern ist auch über ein mobiles Gerät bedienbar. Dies bietet den Vorteil, dass auch wenn man sich gerade nicht in der Nähe der Innenstation befindet, Gäste empfangen werden können.

Paketdienst: Die Paketübergabe erfolgt üblicherweise persönlich. Wenn der Empfänger nicht zuhause ist, nimmt der Paketzusteller die Ware wieder mit, deponiert sie bei einer Paketabholstelle und hinterlässt dem Empfänger lediglich eine Benachrichtigung. Diese Vorgangsweise ist besonders ärgerlich, wenn man den Paketzusteller um wenige Minuten versäumt hat oder nicht schnell genug zur Innenstelle gelangen konnte. Beispielsweise befindet sich der Bewohner gerade auf dem Dachboden oder im Keller. Mit einer Smartphone-App ist eine sofortige Kontaktaufnahme mit dem Paketzusteller möglich und verhindert eine unnötige zusätzliche Bearbeitung des Paketversandes.

Sicherheit: Die neu gewonnene Ortsunabhängigkeit der Anlage bietet eine höhere Einbruchssicherheit, da man jederzeit Überblick über gewünschte bzw. ungewünschte Besucher hat. Über die Smartphone-Applikation ist ein Öffnen des Türschlosses aus mehreren Gründen deaktiviert:

- Wenn der Benutzer nicht zuhause ist, ist eine Türöffnung in den allermeisten Fällen nicht erwünscht.
- Wenn der Benutzer zuhause ist, muss er sowieso zur Eingangstür gehen, um den Besuch in Empfang zu nehmen. Die im Eingangsbereich installierte Station dient hier zur Öffnung der Haustür.
- Falls unbefugter Zugriff auf die Applikation erfolgen sollte, besteht kein besonderes Sicherheitsrisiko hinsichtlich Einbruch in die Wohnung.

1.2 Aufgabenteilung

Andreas Grain ist der Projekt-Hauptverantwortliche und zuständig für die App-Programmierung. Dies beinhaltet die Einarbeitung in und Verwendung des Xamarin-Rahmenwerks zur Erstellung einer Multiplattform-Smartphone-Applikation. Diese soll den Video-Stream

der gewählten Station abrufen und anzeigen, sowie den Mikrofon-Ton des Mobilgerätes zur Anlage zurückschicken. Weiters ist er für die Einarbeitung in das GStreamer-Rahmenwerk zur zentralen Verarbeitung der Video- und Audio-Daten der Stationen und Mobilgeräte über einen linuxbasierten Server zuständig. Darüber hinaus organisiert Andreas Grain alle Treffen mit dem Betreuer, achtet auf die Einhaltung des Zeitplans und ist zuständig für das Erstellen des XƎLATEX -Dokuments

Matthias Mair ist verantwortlich für die Hardware-Überarbeitung der Station. Diese beinhaltet die Einarbeitung in das PCB-Entwicklungsprogramm EAGLE, sowie das Recherchieren und Auswählen möglicher Verstärker-Schaltungen und -ICs für die Mikrofon- und Lautsprecher-Verstärkung. Im Zuge der Hardware-Überarbeitung sollen die vielen Einzelmodule in einer übersichtlichen Platine vereint werden. Für die Leiterplatte soll die weit verbreitete SMD-Technologie verwendet werden, da diese heutzutage Marktstandard für integrierte Geräte ist und im Vergleich zu traditionellen THT-Platinen viel platzsparender ist. Zusätzlich wird die Station um einen Watchdog-Timer erweitert, den Matthias Mair entwerfen und die dazugehörige Software schreiben soll.

Teil I

Hardware - Matthias Mair

2 Probleme des Ist-Standes

3 Platine

3.1 Spannungsversorgung

3.2 Mikrofon-Verstärkerschaltung

3.3 Lautsprecher-Verstärkerschaltung

3.4 Mikrocontroller

3.4.1 Programmierung

3.4.2 Watchdog

Teil II

Software - Andreas Grain

4 Auswahl Rahmenwerk

4.1 Vergleich

Das Rahmenwerk dient der Vereinfachung des Entwicklungsprozesses. Je nach Anwendung sind von diesem verschiedene Anforderungen zu erfüllen, daher ist die Wahl des richtigen Rahmenwerks besonders wichtig. Dieser Abschnitt beschäftigt sich mit dem Vergleich der einzelnen Möglichkeiten, sowie einer endgültigen Auswahl eines der Rahmenwerke zur Verwendung im Diplomprojekt. Die geforderten Funktionen für dieses Projekt beinhalten:

- eine einfache Möglichkeit der Multiplattform-Entwicklung der App. Wenn möglich soll nur eine App geschrieben werden, die dann auf allen Zielplattformen (Android, iOS) lauffähig ist.
- Die Programmierung sollte in einer bereits bekannten Programmiersprache möglich sein. Für die Fachrichtung Elektrotechnik der HTL Anichstraße werden momentan die Sprachen C, C++ und C#, in untergeordnetem Ausmaß auch JavaScript für Webseiten-Entwicklung unterrichtet.
- Das Rahmenwerk und die dazugehörige Entwicklungsumgebung sollten für nichtkommerzielle Anwendungen wie diese Diplomarbeit frei zur Verfügung stehen.
- Die Entwicklungswerkzeuge sollten das Windows-, optional das Linux-Betriebssystem unterstützen.

Für die App-Programmierung stehen viele verschiedene Rahmenwerke bereits zur Verfügung.

Microsoft Xamarin ist ein Rahmenwerk, mit dem man Multiplattform-Applikationen für unter Anderem Android, iOS, UWP und noch viele weitere Plattformen erstellen kann. Es basiert auf Microsofts NetFx und dem Mono-Projekt, welches sich als Ziel gesetzt hat, das

NetFx auf andere Plattformen zu portieren. Xamarin bietet mehrere Projekttypen an, darunter Xamarin.Android und Xamarin.iOS für native App-Entwicklung und Xamarin.Forms für großteils plattformunabhängige Entwicklung. Eine Xamarin.Forms-Solution besteht daher aus mehreren Teilen:

- Portable/.NET-Standard Projekt, das den plattformunabhängigen Code beinhält
- natives Xamarin-Projekt für jede Zielplattform

Der Vorteil Xamarins ist, dass der Großteil des geschriebenen Codes im plattformunabhängigen Projekt bleibt und nur für wenige Funktionen auf native Programmierung zurückgegriffen wird, wie zum Beispiel für hardwarenahe Audio-Aufnahme. Außerdem ist das Xamarin-Projekt Open Source, das bedeutet jeder kann den Quellcode betrachten und unter Umständen Verbesserungen vorbringen.

Webtechnologiebasierte Rahmenwerke: In dieser Kategorie existieren sehr viele verschiedene Rahmenwerke, darunter Apache Cordova, Adobe PhoneGap, sowie Facebook React Native. Am Beispiel von Apache Cordova werden hier die Vor- und Nachteile dieses Ansatzes erläutert.

Cordova ist ein Appentwicklungs-Rahmenwerk welches ursprünglich von der Firma Nitobi entwickelt wurde. Im Jahr 2011 wurde Nitobi von Adobe aufgekauft und das Rahmenwerk wurde zu PhoneGap umgetauft. Später wurde eine quelloffene Version unter dem ursprünglichen Namen Cordova veröffentlicht. Apps werden mittels gängiger Webtechnologie, wie zum Beispiel JavaScript, CSS und Ähnlichen, entwickelt und realisiert, weshalb das Rahmenwerk alle gängigen Plattformen unterstützt. Der Vorteil von Cordova und ähnlichen Rahmenwerken liegt im enormen Support dieser Webtechnologien, jedoch sind Sprachen wie JavaScript für Back-End-Programmierung weniger geeignet. Noch dazu kommt, dass an der HTL Anichstraße großteils die Programmierung nur in C und C# gelehrt wird, welshalb die Entwicklung mit JavaScript im vereinbarten Zeitrahmen der Diplomarbeit nicht realistisch umsetzbar ist.

Aufgrund der oben erläuterten Vor- und Nachteile der möglichen Ansätze wurde sich für Xamarin zur Verwendung in diesem Diplomprojekt entschieden.

4.2 Build-Umgebung

4.2.1 Grund-Setup

Aufgrund obiger Aufstellung wurde Visual Studio 2019 der Firma Microsoft als Entwicklungsumgebung gewählt. Visual Studio ist das offizielle Werkzeug für die Programmierung mit dem Xamarin-Rahmenwerk und die wahrscheinlich bestbekannte Entwicklungsumgebung überhaupt. Die Community-Edition für Schüler und Private steht gratis zum Download zur Verfügung. Diese beinhält alle wichtigen Entwicklungswerkzeuge wie zum Beispiel die automatische Code-Vervollständigung. Für die Nutzung wird nach den ersten 30 Tagen ein Microsoft-Konto benötigt.

Der Programmcode wird von Visual Studio 2019 in sogenannten Solutions organisiert. Am Beispiel einer Xamarin.Forms-Applikation lässt sich das sehr gut erläutern; die Solution enthält auf oberster Ebene drei Projekte: das portable Projekt, das Android- und das iOS-Projekt. Ein Projekt kann bereits für sich lauffähig oder nur Teil eines größeren Programms sein.

Visual Studio 2019 unterstützt viele verschiedene Einsatzbereiche, die bei der Installation ausgewählt werden müssen. Eine volle Installation mit allen Funktionen und Projektarten ist zwar möglich, benötigt allerdings bis zu 210GB Speicherplatz des Systems. Eine typische Xamarin-Installation benötigt etwa 6.25GB Festplattenspeicher. Für eine solche Installation muss im Visual Studio Installer das „Mobile development with .NET“-Paket ausgewählt und installiert werden.

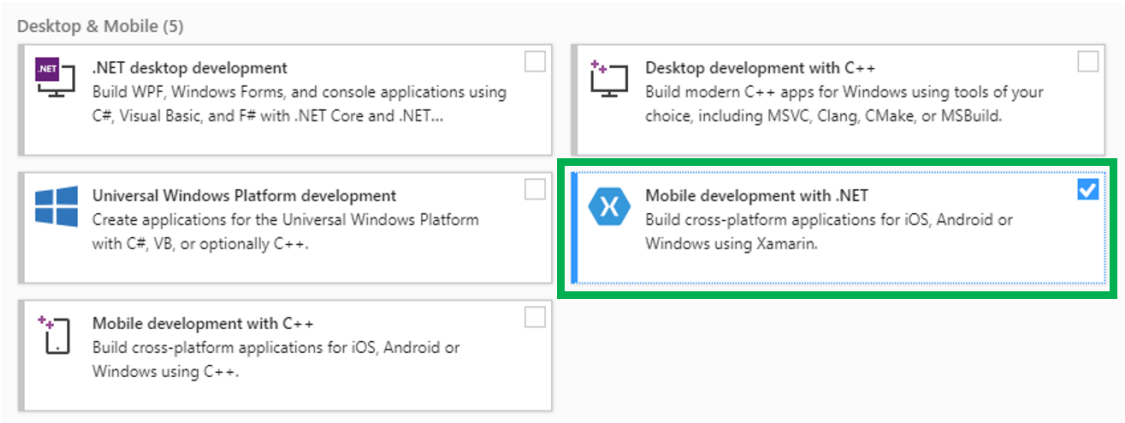

Abbildung 4.1: Auswahl des Xamarin-Rahmenwerks bei der Installation

Es empfiehlt sich, das Paket für „.NET desktop development“ dazu zu installieren, damit neue, noch nicht bekannte NetFx-Technologien in einer einfacheren Konsolen-Anwen-

dung ausprobiert werden können. Das Paket benötigt ungefähr weitere 0.8GB.

Auf nähere Details der Installation von Visual Studio wird hier nicht eingegangen, diese können aber auf der Microsoft-Dokumentationsseite [vgl. 1] nachgeschlagen werden.

4.2.2 Android SDK

Um Xamarin-Applikationen für Android-Geräte entwickeln zu können wird das von Google bereitgestellte Android SDK benötigt, um das Programm in ein APK umzuwandeln. Die einzelnen Komponenten können im Android-SDK-Manager des Visual Studio installiert werden. Dieser ist im Menüband unter „Tools\Android\Android SDK Manager“ zu finden.

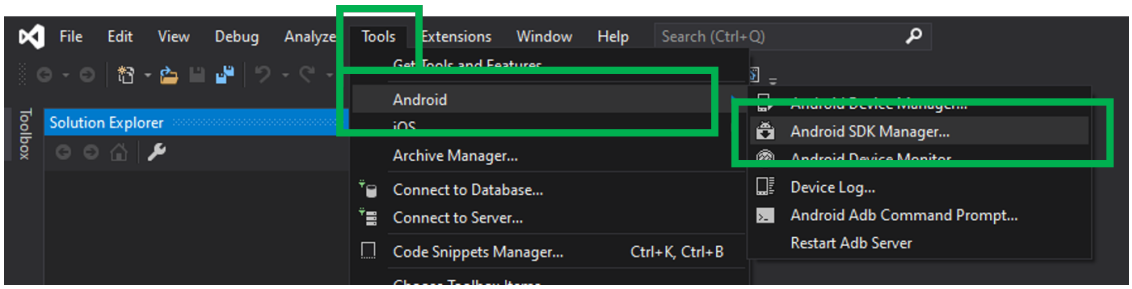

Abbildung 4.2: Android SDK Manager

Der SDK-Manager ist in zwei Seiten unterteilt - „Platforms“ und „Tools“.

Auf der „Platforms“-Seite können die benötigten SDK-Teile für jede Android-Version installiert werden. Benötigt wird nur die Version, mit der das Projekt später kompiliert werden soll. Außerdem kann man auf dieser Seite des SDK-Managers System-Abbilder für den Android Emulator installieren.

Über die „Tools“-Seite können verschiedenste Entwicklungswerkzeuge installiert werden. Grundsätzlich erforderlich sind „Android SDK Tools“, „Android SDK Platform Tools“ und „Android SDK Build Tools“ um Android-Apps mit Xamarin entwickeln zu können. Für dieses Projekt werden Push-Benachrichtigungen via Google Cloud Messaging verwendet, daher werden noch zusätzlich die Pakete für die „Google Play Services“ sowie die „Google Cloud Messaging for Android Library“ installiert. Wie Push-Benachrichtigungen genauer funktionieren wird später im Kapitel *6 Push-Benachrichtigung* auf Seite 27 behandelt.

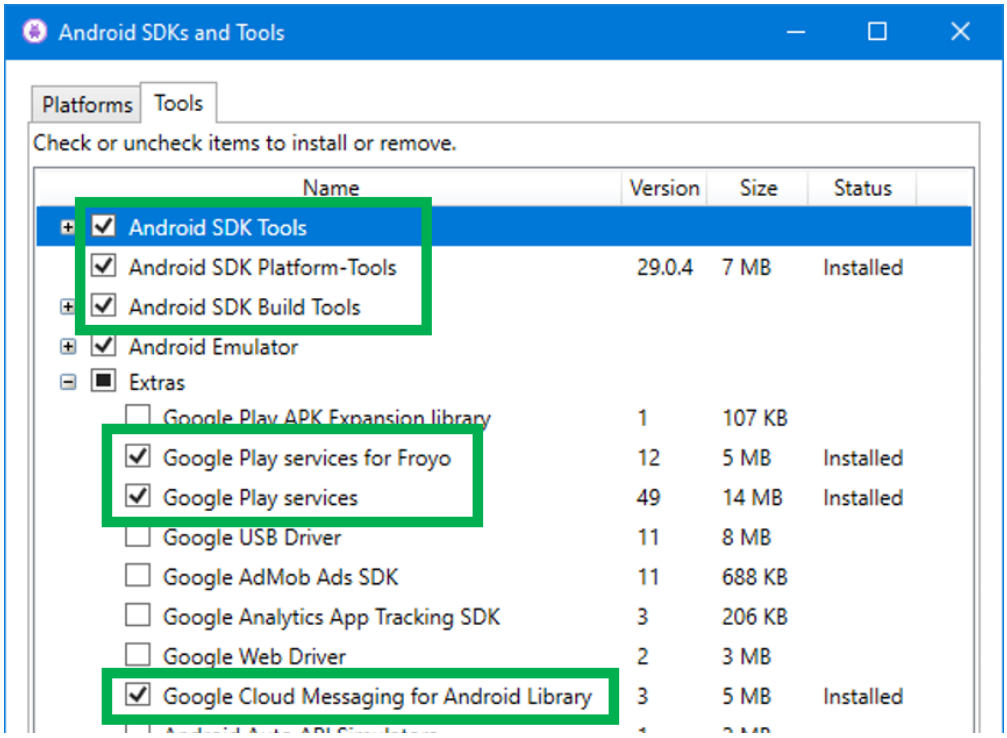

Abbildung 4.3: Notwendige Teile des Android SDK

4.2.3 Build-Vorgang

Um den geschriebenen Code in eine lauffähige Applikation umzuwandeln wird ein sogenannter Compiler benötigt. Die Aufgabe des Compilers ist es, die geschriebene Hochsprache entweder direkt in CPU-Maschinensprache bzw. eine Art Zwischencode zu kompilieren, d.h. umzuwandeln. Im Fall einer einfachen C#-Konsolenanwendung ist der Build-Vorgang noch relativ simpel:

1. Alle einzelnen Code-Dateien werden nacheinander vom Compiler in CIL, eine Zwischensprache des NetFx, umgewandelt.

2. Der Linker fügt die einzelnen Teile zu einem Gesamten zusammen. Das Resultat ist eine Datei, die wie eine ausführbare .EXE-Datei erscheint.

3. Wenn das zusammengefügte Programm gestartet wird, kompiliert die CLR den Zwischencode in ausführbaren Maschinen-Code. Die CPU des Systems kann diesen dann ausführen.

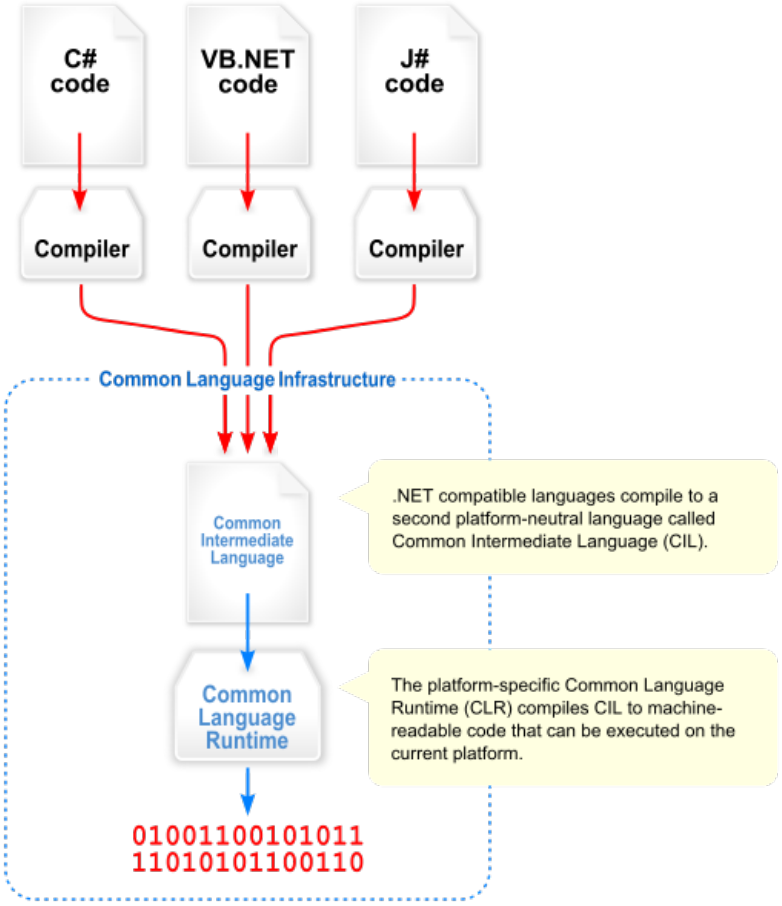

Abbildung 4.4: Build-Vorgang einer NetFx-Applikation[2]

Ähnlich funktioniert dieser Vorgang auch bei einer Xamarin.Forms-App, allerdings wesentlich komplexer. Eine Xamarin.Forms-Solution besteht aus drei Einzel-Projekten, wobei das .NET-Standard-Projekt als Teil der anderen zwei verbaut werden muss. Dadurch ergibt sich ein in Abbildung *4.5 Build-Vorgang einer Xamarin.Forms-Solution* dargestellter Build-Vorgang.

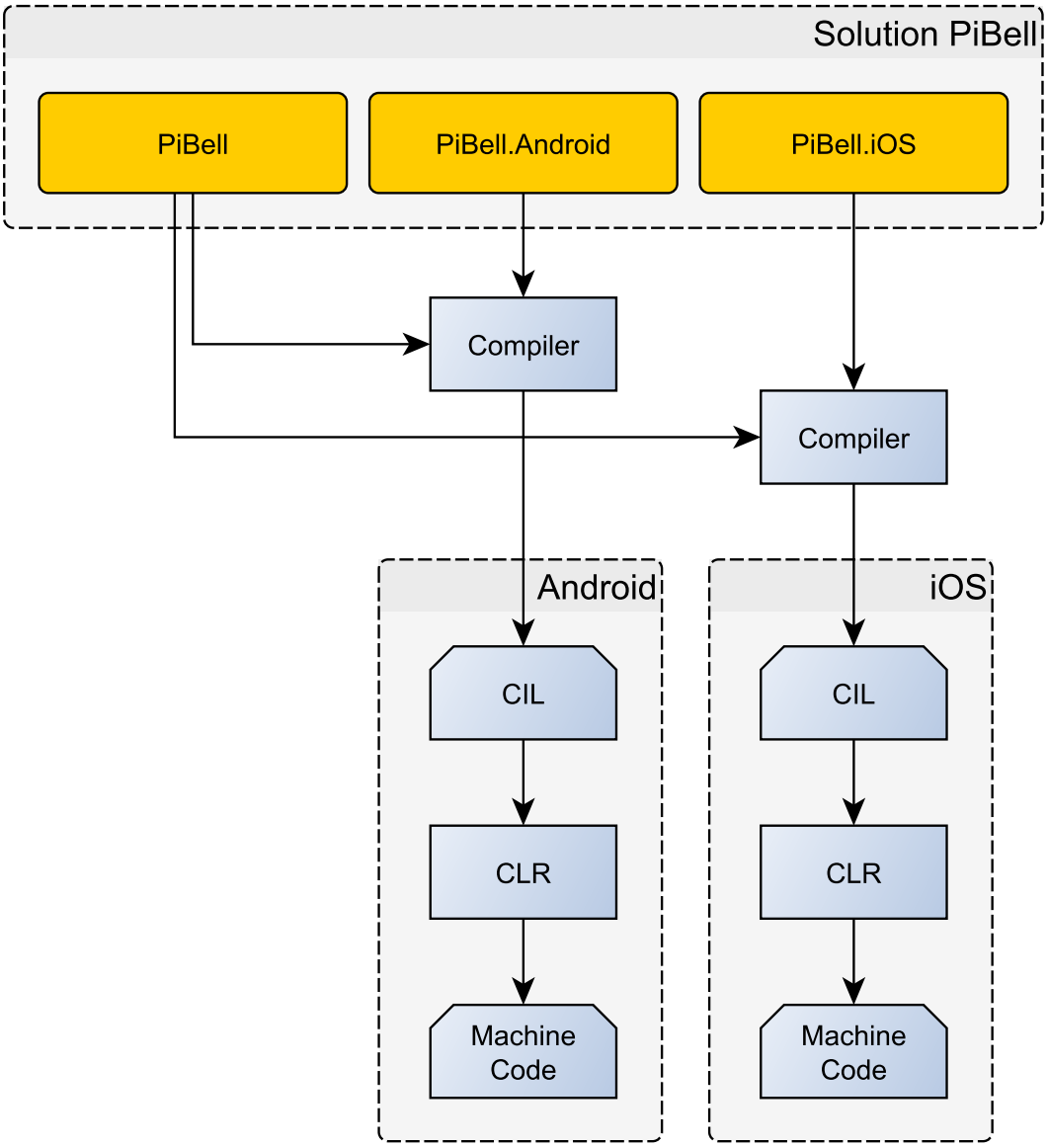

Abbildung 4.5: Build-Vorgang einer Xamarin.Forms-Solution

5 Verwendete Software-Module

5.1 Real Time Streaming Protocol

RTSP ist ein Netzwerkprotokoll zum Aufbauen und Verwalten von Netzwerk-Verbindungen zur Übertragung kontinuierlicher Medien-Daten (Streams). Dieses Netzwerkprotokoll ist im IETF-Dokument *RFC 2326 - Real Time Streaming Protocol (RTSP)* festgelegt. Es wurde gemeinsam von H. Schulzrinne (Columbia University), A. Rao (Netscape Communications) und R. Lanphier (RealNetworks) entwickelt. [vgl. 3]

Im Zuge dieser Diplomarbeit wird die erste Version des RTSP-Protokolls verwendet. Neben der verwendeten Version 1 existiert auch eine neue Version 2, welche im IETF-Dokument *RFC 7826 - Real-Time Streaming Protocol Version 2.0* definiert ist. Die neue Version wurde von H. Schulzrinne (Columbia University), A. Rao (Cisco Networks), R Lanphier u. a. erarbeitet. [vgl. 4] Es wurde dennoch die erste Version verwendet, da die Stationen dieses bereits verwenden und die neue Version nicht rückwärtskompatibel ist.

RTSP wird in der Industrie vielseitig verwendet, um Live-Übertragungen von Überwachungskameras zu verwalten. Eine weitere Anwendung des RTSP Protokolls ist das Streamen von Medien-Daten zu einem Server, der diese dann beispielsweise transcodiert oder aufnimmt.

5.1.1 Anfragen-Aufbau

Das RTSP-Protokoll definiert mehrere Anfragen zur Verwaltung der Netzwerkverbindung. Diese Anfragen werden, ähnlich wie beim HTTP, in unverschlüsseltem Plain-Text verschickt. In Tabelle 5.1 sind sämtliche Anfragen des RTSP-Protokolls und deren Übertragungsrichtung aufgelistet. Die in Tabelle 5.1 verwendeten Kurzbezeichnungen haben folgende Bedeutung:

Richtung: C…Client, S…Server

Objekt: P…Präsentation, S…Stream

Methode	Richtung	Objekt	Verbindlichkeit
DESCRIBE	C→S	P,S	empfohlen
ANNOUNCE	C→S, S→C	P,S	optional
GET_PARAMETER	C→S, S→C	P,S	optional
OPTIONS	C→S, S→C	P,S	erfordert, (S→C: optional)
PAUSE	C→S	P,S	empfohlen
PLAY	C→S	P,S	erfordert
RECORD	C→S	P,S	optional
REDIRECT	S→C	P,S	optional
SETUP	C→S	S	erfordert
SET_PARAMETER	C→S, S→C	P,S	optional
TEARDOWN	C→S	P,S	erfordert

Tabelle 5.1: Anfrage-Arten des RTSP-Protokolls

Das Aussehen einer solchen Anfrage wird anhand des Beispiels der DESCRIBE-Methode veranschaulicht. Der Client beginnt die Anfrage mit dem Url des Medien-Streams, den er abrufen möchte und teilt dem Server mit, welche Formate er versteht.

```
C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
    CSeq: 312
    Accept: application/sdp, application/rtsl, application/mheg
```

Der server antwortet auf diese Anfrage mit dem Session Descriptor, der alle wichtigen Informationen des Medien-Streams beinhaltet, wie zum Beispiel Video- und Audio-Format, Übertragungsmethode, sowie allgemeine Informationen wie Titel und Beschreibung.

```
S->C: RTSP/1.0 200 OK
    CSeq: 312
    Date: 23 Jan 1997 15:35:06 GMT
    Content-Type: application/sdp
    Content-Length: 376

    v=0
    o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
    s=SDP Seminar
    i=A Seminar on the session description protocol
    u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
    e=mjh@isi.edu (Mark Handley)
    c=IN IP4 224.2.17.12/127
    t=2873397496 2873404696
```

```
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31
m=whiteboard 32416 UDP WB
a=orient:portrait
```

5.2 LibVLC Sharp

LibVLC ist eine C-Bibliothek für die Medienverarbeitung und -Wiedergabe. Mit Hilfe von Bindings kann diese Bibliothek auch in anderen Programmiersprachen verwendet werden. Eines dieser Binding ist LibVLC Sharp, welches die Funktionen der LibVLC in C# zur Verfügung stellt. Es handelt sich bei der LibVLC um ein OpenSource-Projekt der Organisation VideoLAN, das fortlaufend weiterentwickelt und unterstützt wird. Der sehr bekannte VLC Media Player nutzt für alle Funktionen die LibVLC und dient daher sozusagen als Grafische Oberfläche zur LibVLC. [vgl. 5]

Im Zuge dieser Diplomarbeit wird das C#-Binding der LibVLC verwendet, um am Smartphone den Video-Stream der fest installierten Station abrufen und darstellen zu können. Dies wird näher im Kapitel *7 Programm-Dokumentation* beschrieben.

5.3 GStreamer

GStreamer ist ein extrem leistungsfähiges und vielseitiges Framework zur Erstellung von Medien-Streaming-Anwendungen. Viele der Vorzüge des GStreamer-Frameworks liegen in seiner Modularität: GStreamer kann neue Plugin-Module nahtlos integrieren. Aber da Modularität und Leistungsfähigkeit oft mit einem Preis für eine größere Komplexität einhergehen, ist das Schreiben neuer Anwendungen nicht immer einfach. [6, aus dem Englischen übersetzt]

Gstreamer ist Pipeline-basiert, d.h. die Funktion wird mit einer Vielzahl von hintereinander geschalteten Filtern bestimmt. Eine Pipeline hat immer einen Eingang (Source) und einen oder mehrere Ausgänge. Zwischen diesen können sich beliebig viele Filter befinden, die unterschiedlichste Funktionen haben können:

- Signale de- und encodieren (Codec, z.B. x264enc)
- Größe und Position von Videos verändern (Caps, z.B. video/x-raw)
- die Pipeline in zwei Teile auftrennen, z.B. in Video und Audio (Demux)

- Daten zu einem Netzwerk-Paket zusammenbündeln (z.B. rtph264pay)
- etc.

Ein Filter hat bestimmte Ein- und Ausgangsformate die er unterstützt. Als Beispiel nehmen wir den H264-Encoder: dieser unterstützt für das Eingangssignal RAW-Video, also decodierte Binärdaten. Am Ausgang kommt logischerweise ein H264-codierter Video-Stream heraus.

Das Gstreamer-Rahmenwerk wird aktuell vom Gstreamer-Team fortlaufend upgedatet und verbessert. Die erste Version des Gstreamer-Rahmenwerks wurde im Jahr 2001 mit der Versionsnummer 0.1.0 veröffentlicht. Im Jahr 2012 kam eine große Umstellung, bei der grundlegende Teile des Rahmenwerks ausgetauscht wurden, wodurch einige Teile nicht mehr kompatibel waren. Ab diesem Zeitpunkt begann die Versionsnummer mit 1.x.x, um die neuen Versionen deutlich von den Vorgängern abzutrennen. Die aktuelle Version des Gstreamer-Rahmenwerks zum Verfassungszeitpunkt dieser Arbeit ist die Version 1.16.2.

Aktuell wird die Version 1.16.2 verwendet. Da aber alle 1.x.x-Versionen mehr oder weniger miteinander kompatibel sind, kann sich das ändern, sobald ein neues Update veröffentlicht wird. Es gibt keinen besonderen Grund auf genau dieser Version zu bleiben.

Es wurden während der Recherche einige mögliche Alternativen für das GStreamer-Rahenwerk gefunden:

- FFmpeg, ein Multimedia-Rahmenwerk, das vor allem im Bereich Transcodierung verwendung findet. Es bietet alle Funktionen des GStreamer-Rahmenwerks an.
- LibVLC, die Grundlage des VLC Media Players, welche vor allem zur Wiedergabe verwendet wird. Diese Bibliothek kann zur Transcodierung und Live-Übertragung genutzt werden, ist allerdings nicht dafür optimiert.

GStreamer wird in diesem Projekt verwendet, da bei Live-Video-Übertragung möglichst wenig Latenz vorkommen soll. Das Gstreamer-Rahmenwerk ist in dieser Hinsicht den anderen Multimedia-Rahmenwerken weit voraus. Zusätzlich gibt es von GStreamer keine einschränkenden Vorgaben was Ein- und Ausgangsformat betrifft. Es kann jede beliebige Quelle mit jedem beliebigen Output verknüpft werden, solange die richtige Pipeline verwendet wird. Weiters ist das Projekt quelloffen, d.h. man ist in Bezug auf die Installationsdateien nicht auf den Hersteller angewiesen, sondern kann das Programm selbst für die jeweilige Plattform kompilieren, wenn der Hersteller das noch nicht getan hat. Außer-

dem kann jeder Entwickler Vorschläge für Änderungen am Code, sowie Bugfixes vorbringen.

5.3.1 Pipeline-Beispiel

Die detaillierte Funktionsweise des GStreamer-Rahmenwerks lässt sich am besten anhand eines realistischen Beispiels erklären. In diesem Beispiel wird ein in Echtzeit generiertes Testvideosignal zuerst auf 1280x720 Pixel vergrößert und danach als H264-encodierter Stream via RTP verschickt.

```
gst-launch-1.0 videotestsrc ! video/x-raw,width=1260,height=720,
   framerate=20/1 ! autovideoconvert ! queue ! x264enc tune=
   zerolatency bitrate=4096 speed-preset=superfast ! queue !
   rtph264pay config-interval=1 mtu=1300 ! udpsink host=127.0.0.1
   port=5000
```

Der Filter „queue“ wird hier verwendet, um vor und nach dem Encoding-Vorgang ein kurzes Video-Segment in einem Zwischenspeicher zu behalten. Dies ist notwendig, da der H264-Codec mehrere Video-Frames auf einmal komprimiert.

5.4 Live555 Proxy

6 Push-Benachrichtigung

6.1 Push Notification Service

6.2 AppCenter Push

7 Programm-Dokumentation

7.1 Übersicht

Das im Zuge dieser Diplomarbeit entwickelte Software-Projekt basiert auf einer der von Microsoft bereitgestellten Xamarin.Forms-Vorlagen, welche als Teil von Visual Studio bereitgestellt werden. Es gibt mehrere Auswahlmöglichkeiten, wobei alle Vorlagen bis auf „Blank“ Beispielcode beinhalten. Da der Beispielcode der anderen Vorlagen für dieses Projekt nicht relevant ist und ohnehin gelöscht werden müsste, wird die „Blank“-Vorlage verwendet. Diese Vorlage beinhält nur eine Haupt-Oberflächenseite ohne jeglichen Beispielcode.

Der immer wieder vorkommende Name PiBell ist eine freigeistliche Erfindung. PiBell ist der Projektname, der sich aus Raspberry Pi und „Bell“, dem englischen Wort für Klingel oder Glocke, zusammensetzt.

Visual Studio organisiert alle Programmteile in einer sogenannten Solution, welche den Projektnamen trägt. In diesem Fall beinhält die Solution „PiBell“ drei Programmteile oder „Projects“:

PiBell ist das portable Projekt, auch .NET-Standard-Projekt genannt. Dieses beinhaltet den ganzen plattformunabhängigen Code, sowie die Definition der UI-Oberfläche mittels XAML-Dateien. In diesem Projekt befindet sich der größte entwickelte Code-Anteil, da die meisten Funktionen, wie das Abspielen von Videos mit Hilfe der LibVLC-Bindings, plattformunabhängig funktionieren und deswegen geteilt werden können.

PiBell.Android beinhaltet allen Code, der plattformspezifisch für Android geschrieben wurde. Unter anderem ist hier enthalten die Android-Implementation des Mikrofon-Aufnahme-Services, sowie das Berechtigungs-Management. Die Mikrofon-Aufnahme erfolgt auf einer sehr hardwarenahen Ebene, was die plattformabhängigkeit erklärt.

PiBell.iOS beinhaltet wie PiBell.Android den plattformspezifischen Code für die iOS-Plattform. Aufgrund mangelnder Entwicklungswerkzeuge wurde dieser Teil nicht großartig behandelt.

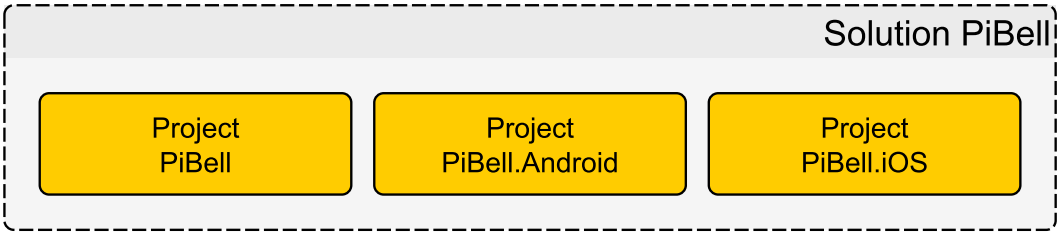

Abbildung 7.1: Aufbau der Solution

7.2 Portable Project

7.3 PiBell.Android

7.4 PiBell.iOS

8 Testen

9 Ausblick

9.1 Gesicherte Verbindung

Literatur

[1] Microsoft. (Dez. 2019). „Install Visual Studio“, Adresse: https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019.

[2] J. Piiroinen. (Feb. 2008). „Overview of the Common Language Infrastructure“, Wikimedia Commons, Adresse: https://commons.wikimedia.org/wiki/File:Overview_of_the_Common_Language_Infrastructure.svg.

[3] H. Schulzrinne (Columbia University), A. Rao (Netscape Communications) und R. Lanphier (RealNetworks). (Apr. 1998). „RFC 2326 - Real Time Streaming Protocol (RTSP)“, Internet Engineering Task Force, Adresse: https://tools.ietf.org/html/rfc2326.

[4] H. Schulzrinne (Columbia University), A. Rao (Cisco Networks), R Lanphier, M. Westerlund (Ericsson) und M. Stiemerling, Ed. (University of Applied Sciences Darmstadt). (Dez. 2016). „RFC 7826 - Real-Time Streaming Protocol Version 2.0“, Internet Engineering Task Force, Adresse: https://tools.ietf.org/html/rfc7826.

[5] VideoLAN. (März 2020). „libVLC - VideoLAN Wiki“, Adresse: https://wiki.videolan.org/LibVLC.

[6] GStreamer. (März 2020). „GStreamer: a flexible, fast and multiplatform multimedia framework“, Adresse: https://gstreamer.freedesktop.org/documentation/.

Abkürzungen

APK Android Package

CIL Common Interface Language

CLR Common Language Runtime

NetFx .NET-Rahmenwerk (.NET-Framework)

EAGLE Einfach Anzuwendender Grafischer Layout-Editor

FI Fachspezifische Softwaretechnik

IC integrierte Schaltung (Integrated Circuit)

PCB Leiterplatte (Printed Circuit Board)

RPi Raspberry Pi

RTSP Real Time Streaming Protocol

SDK Software Development Kit

SMD oberflächenmontiertes Bauelement (Surface-mounted Device)

THT Through-Hole Technology

UWP universelle Windows-Plattform (Universal Windows Platform)

Abbildungsverzeichnis

Tabellenverzeichnis

Teil III

Appendix

A Verwendete Entwicklungswerkzeuge

B Datenblätter

C Kostenübersicht

D Fertigungsdokumentation