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 2020-03-08

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 zu verstehen ist.

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. Die verwendete Hardware bietet Verbesserungspotential und wird im Zuge dieser DIplomarbeit weiterentwickelt.

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. The hardware used offers potential for improvement and will be further developed in the course of this diploma thesis.

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 Prantl an, 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 Einbruchs 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 Linux-basierten 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 XELaTeX -Dokuments

Matthias Mair ist verantwortlich für die Hardware-Überarbeitung der Station. Diese beinhaltet die Einarbeitung in das PCB-Entwicklungsprogramm EAGLE von der Firma Autodesk mit der Version 9.5.2, 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 Rahmenwerk

4.1 Anforderungen

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.

4.2 Vergleich

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

4.2.1 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. Zusätzlich ist es für diese Anwendung kostenfrei.

4.2.2 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 App-Entwicklungs-Rahmenwerk welches ursprünglich von der Firma Nitobi unter dem Name PhoneGap entwickelt wurde. Im Jahr 2011 wurde Nitobi von Adobe aufgekauft. Später wurde neben der Closed-Source-Version auch eine quelloffene Version unter dem Namen Cordova veröffentlicht. [1]

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, weshalb die Entwicklung mit JavaScript im vereinbarten Zeitrahmen der Diplomarbeit nicht realistisch umsetzbar ist.

Aufgrund der oben erläuterten Vor- und Nachteile der möglichen Ansätze wurde die Verwendung von Xamarin für diese Diplomarbeit festgelegt.

4.3 Build-Umgebung

4.3.1 Visual Studio 2019

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.

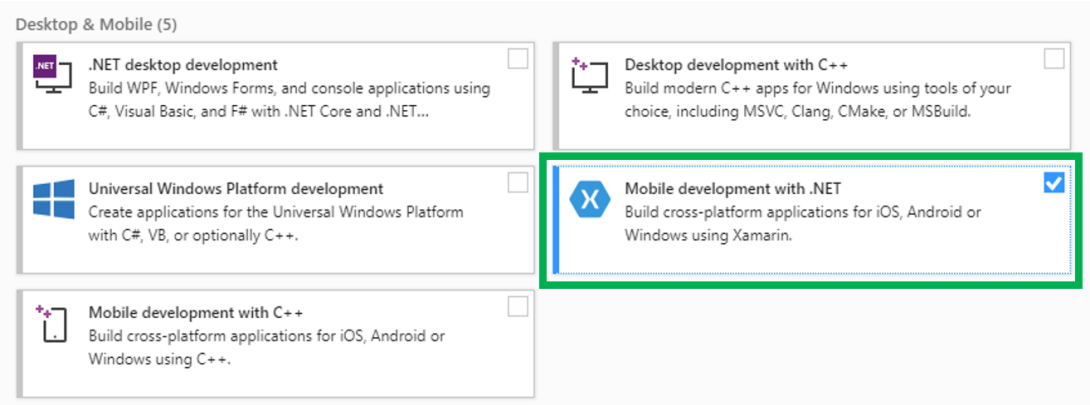

Abbildung 4.1: Auswahl des Xamarin-Rahmenwerks bei der Installation

Es empfiehlt sich, das Paket für „.NET desktop development“ dazu zu installieren, damit neue, noch nicht bekannte NetFx-Technologien in einer einfacheren Konsolen-Anwendung 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. 2] nachgeschlagen werden.

4.3.2 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.

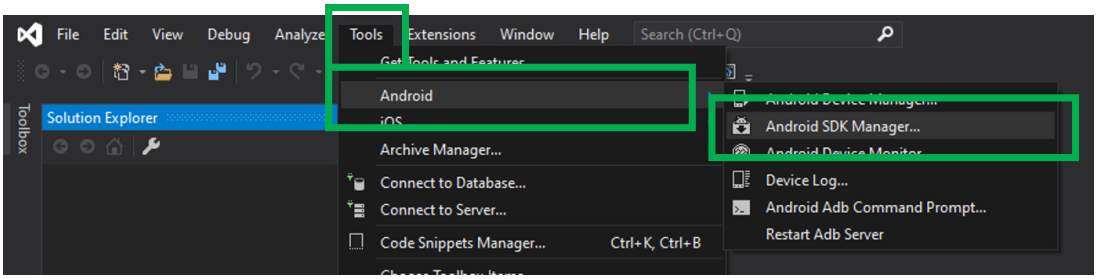

Abbildung 4.2: Android SDK Manager

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 29 behandelt.

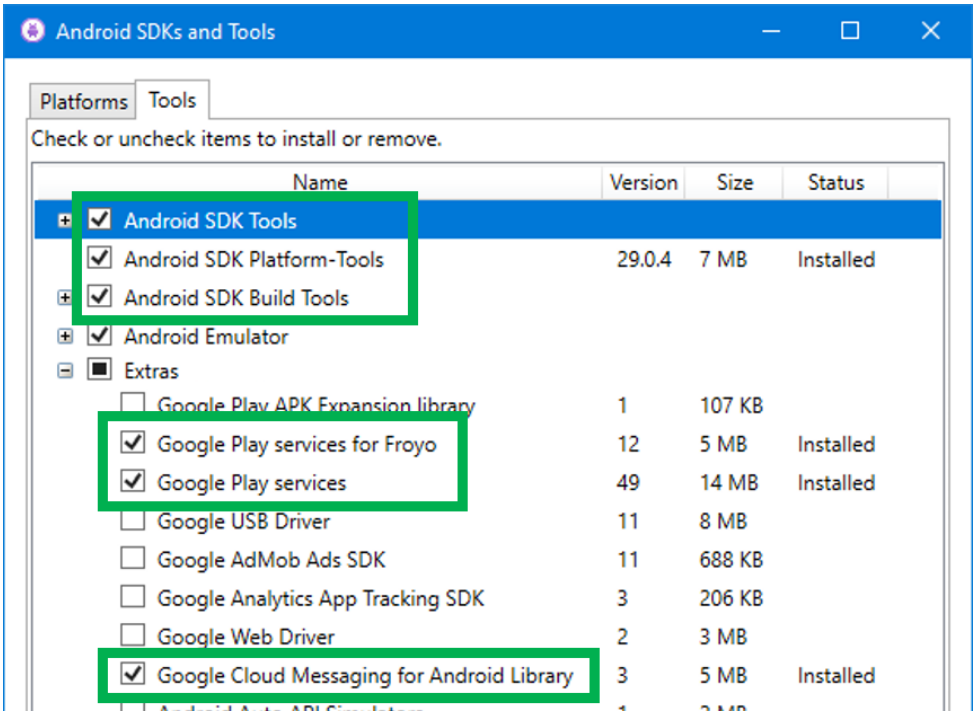

Abbildung 4.3: Notwendige Teile des Android SDK

4.3.3 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.

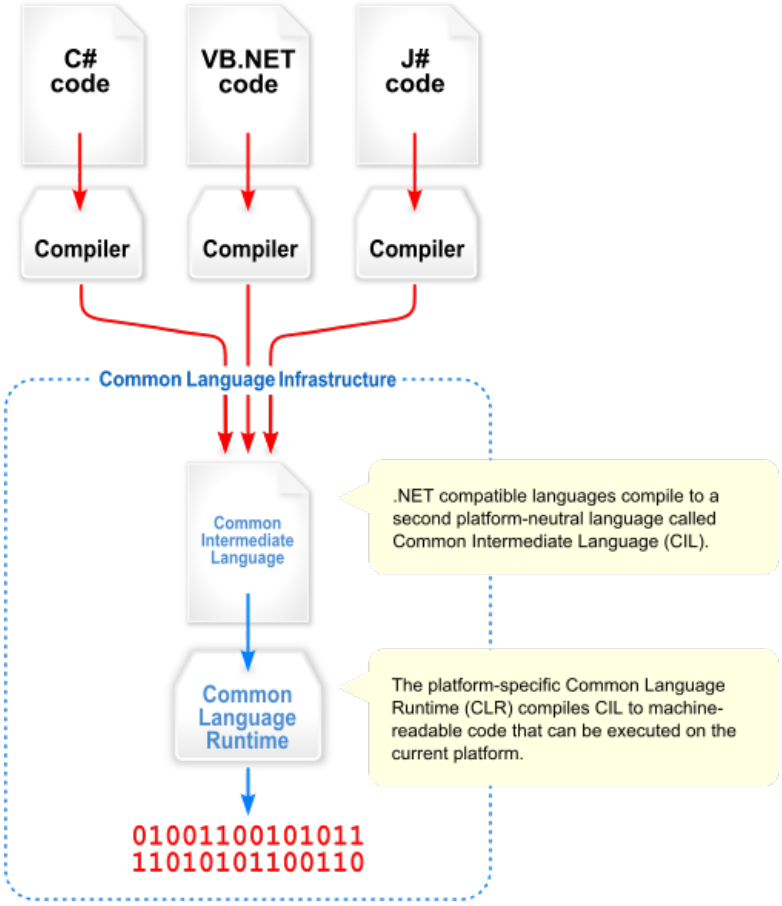

Abbildung 4.4: Build-Vorgang einer NetFx-Applikation[3]

Ähnlich funktioniert dieser Vorgang auch bei einer Xamarin.Forms-App, allerdings wesentlich komplexer. Eine Xamarin.Forms-Solution besteht aus drei Einzel-Projekten (PiBell, PiBell.Android, PiBell.iOS), 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.

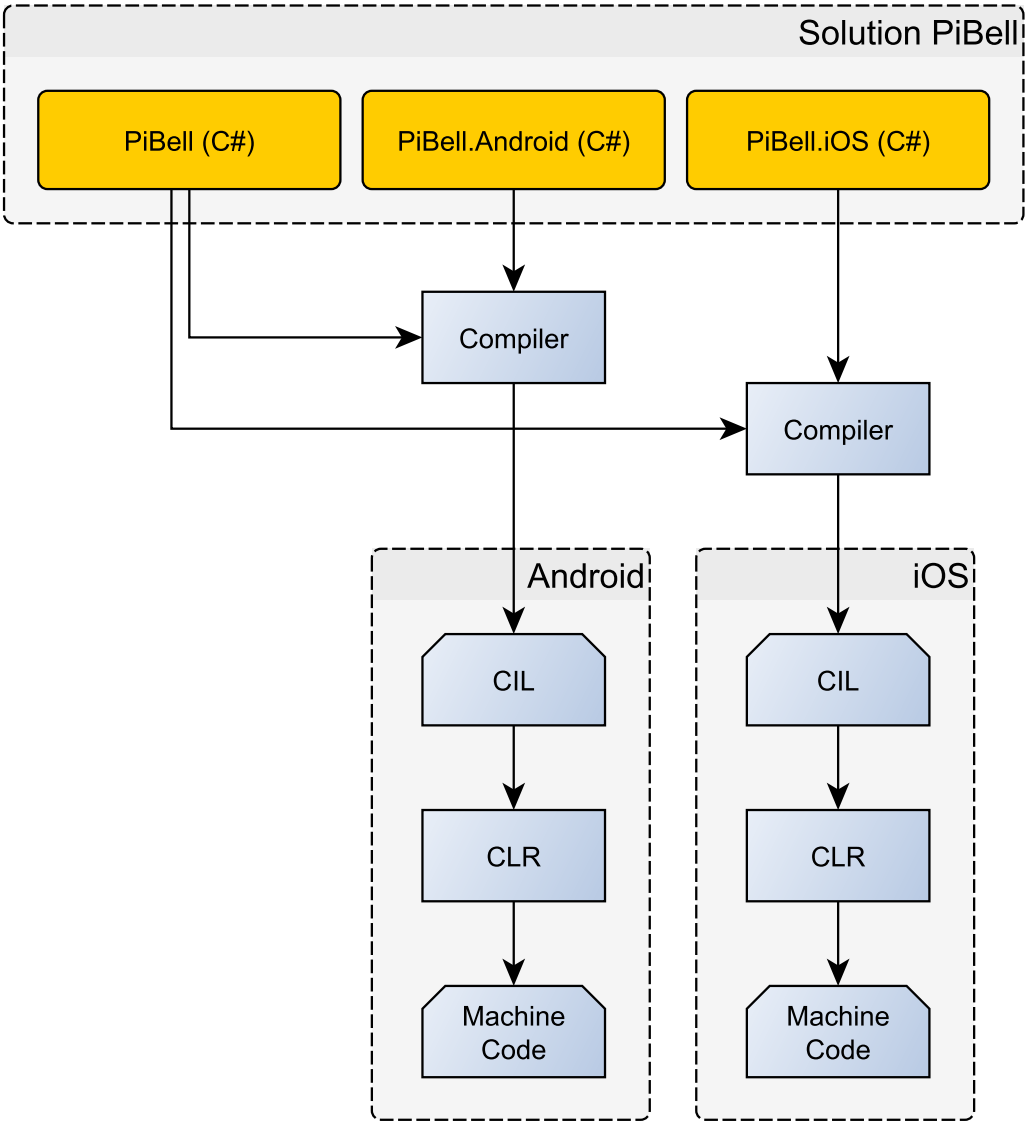

Abbildung 4.5: Build-Vorgang einer Xamarin.Forms-Solution

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. 4]

5.1.1 RTSP Version 2

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. 5] 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.

5.1.2 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.

```
1 C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
2     CSeq: 312
3     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.

```
1  S->C: RTSP/1.0 200 OK
2      CSeq: 312
3      Date: 23 Jan 1997 15:35:06 GMT
4      Content-Type: application/sdp
5      Content-Length: 376
6
7      v=0
8      o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
9      s=SDP Seminar
10     i=A Seminar on the session description protocol
```

```
11      u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
12      e=mjh@isi.edu (Mark Handley)
13      c=IN IP4 224.2.17.12/127
14      t=2873397496 2873404696
15      a=recvonly
16      m=audio 3456 RTP/AVP 0
17      m=video 2232 RTP/AVP 31
18      m=whiteboard 32416 UDP WB
19      a=orient:portrait
```

5.2 LibVLC Sharp

LibVLC ist ein Multimedia-Rahmenwerk der Non-Profit-Organisation VideoLAN, welche durch den VLC Media Player und dessen Funktionsumfang bekannt wurde. Der VLC Media Player funktioniert sozusagen als grafische Oberfläche zur LibVLC-Programmbibliothek. Die Bibliothek beinhält viele nützliche Funktionen, darunter die folgenden:

- Videos nahezu beliebigen Formates decodieren und darstellen
- Video und Audio aufnehmen und Transcodieren für die weitere Verwendung
- Live-Streaming von Video-Daten, Webcam-Bild und Mikrofon-Ton.

LibVLC ist eine Bibliothek für die C-Sprache. Um diese in einem C#-Projekt verwenden zu können sind Bindings notwendig, welche Zugriff auf die C-Bibliothek geben. LibVLC wurde von der Non-Profit-Organisation VideoLAN entwickelt und erstmals am 1. Februar 2001 veröffentlicht wurde.[6]

Im Xamarin.Forms-Projekt wird die zum Zeitpunkt der App-Entwicklung aktuelle Version 3.4.2 der LibVLC Sharp Bindings verwendet. Bindings erlauben es dem Programmierer auf Programmbibliotheken zuzugreifen, welche für eine andere Sprache geschrieben wurden. Man kann sich Bindings wie Kleber vorstellen, der die Funktionen der einen Programmiersprache mit denen der anderen Sprache verbindet. Die Version 3.4.2 ist inzwischen nicht mehr die neueste Version, jedoch ist sie in Hinsicht auf Funktionalität deckungsgleich mit der neuesten Version.

5.2.1 Erwähnenswerte Funktionen

Die LibVLC-Sharp-Bindings ermöglichen die Verwendung des MVVM-Modells, welches vom WPF- und Silverlight-Architekten John Gossman im Jahr 2005 auf seinem Blog vorgestellt wurde. [vlg. 7, The Evolution of Model-View-ViewModel] Die Philosophie hinter diesem Modell ist die Oberfläche (View) bis auf wenige Schnittstellen komplett vom eigentlichen Code zu trennen. Dies ermöglicht es ohne größeren Aufwand die Oberfläche zu ändern oder gar auszutauschen. Folgende Terminologie wird in diesem Zusammenhang verwendet:

- Models … beinhalten die Programmdaten. Meist sind einfache Klassen oder Strukturen hierfür verwendet.
- Views … Die grafische Oberfläche und deren Elemente wie Buttons, TextBoxen, …
- ViewModels … wandeln die Daten der Models so um, dass sie von der Oberfläche dargestellt werden können. ViewModels sind meist als Klasse ausgeführt.

Die Daten der Oberfläche und die Daten des Models bleiben zueinander konsistent. Sobald sich ein Wert auf einer Seite ändert, wird er sofort auf der anderen aktualisiert.

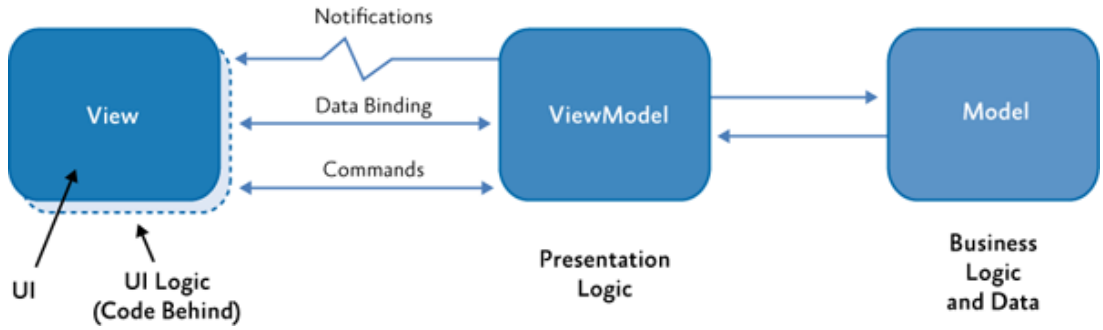

Abbildung 5.1: Datenfluss MVVM

Das MVVM-Modell bietet sich auch an, wenn die Daten des Programms oft in eine andere Form gebracht werden müssen, bevor sie dargestellt werden können. Zum Beispiel wird vom Model die aktuelle Zeit als Ganzzahl abgespeichert, welche nur die vergangenen Millisekunden seit einem bestimmten Zeitpunkt zählt; um die Zeit allerdings darstellen zu können, muss diese Ganzzahl zuerst in ein lesbares Datumsformat gebracht werden. Diese Aufgabe würde das ViewModel übernehmen.

5.2.2 Nachteile

Sowohl die ursprüngliche LibVLC-Bibliothek, als auch die entsprechenden C#-Bindings sind auf der offiziellen Dokumentationsseite nur spärlich dokumentiert.[vgl. 8] Beispielsweise verwenden alle Beispielprojekte der LibVLC Sharp Bindings das MVVM-Modell, daher ist nicht bekannt, ob eine Implementierung ohne dieses Modell überhaupt möglich ist. Weiters sind die einzelnen Funktionen zwar auf der Dokumentationsseite aufgelistet, allerdings nicht sehr ausführlich beschrieben. [vgl. 8]

5.2.3 Alternativen

Während der Recherche wurden wenige Alternativen gefunden. Diese Alternativen sind jedoch seit langem nicht mehr in Entwicklung und sind daher nicht für die Verwendung empfohlen. Die LibVLC-Bibliothek ist momentan die einzige kostenfreie Variante RTSP-Streams mit Xamarin auf dem Mobilgerät abspielen zu können. Es gibt Bibliotheken von VASTreaming, jedoch sind diese kostenpflichtig zu erwerben, was für diese Diplomarbeit keine Option ist. [vgl. 9, Pricing]

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. [10, 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. Außerdem 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.

```
1 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.

5.3.2 gst-rtsp-server

gst-rtsp-server ist ein Zusatzmodul für das GStreamer- Rahmenwerk, welches die Verwendung des RTSP-Protokolls

5.4 Live555 Proxy

Ein RTSP-Stream ist grundsätzlicherweise eine Unicast-Verbindung zwischen Server und Client. Falls mehrere Clients den gleichen Medien-Stream abrufen möchten, muss der Server diesen für jeden Client einzeln transcodieren und verschicken. Dies stellt eine große Belastung für den Server un dessen Internetanbindung dar. Mit Hilfe eines Proxy-Servers muss der Medien-Stream nur einmal verarbeitet werden, bevor er dann an die einzelnen Clients verteilt wird.

Der Live555 Proxy Server ist eine quelloffene Applikation, welche für genau diese Aufgabe von LiveMedia Inc. entwickelt wurde. Mit Hilfe des Live555 Proxy Servers wird ein RTSP-Stream einmalig eingelesen und an alle verbundenen Clients verteilt. Dadurch bleibt die CPU-Belastung des Servers unabhängig von der Anzahl der verbundenen Clients, wodurch er unter Umständen mehrere verschiedene Streams bereitstellen könnte. Für jeden Medien-Stream, den der Server anbietet, wird eine weitere Live555-Proxy-Server-Instanz benötigt.

Der Live555 Proxy Server wurde im Jahr xxxx erstmals von LiveMedia Inc. veröffentlicht. Neben dem Live555 Proxy Server entwickelt das Unternehmen mehrere Projekte:

- Live555 Media Server, mit dem lokale Medien-Dateien per RTSP dem Netzwerk bereitgestellt werden können.
- liveCaster, zur Multicast-Übertragung von MP3-Dateien
- diverse Kommandozeilenprogramme zum Senden und empfangen von Echtzeit-Streams

Die von LiveMedia Inc. entwickelten Programme und Bibliotheken finden in vielen bekannten Applikationen Anwendung, darunter auch der VLC Media Player mit der dazugehörigen LibVLC. In diesem Programm werden die Live555-Bibliotheken zum Empfangen und Entschlüsseln von RTSP-Streams genutzt.

6 Push-Benachrichtigung

6.1 Push Notification Service

Eine Notification dient dazu den Benutzer über etwas zu informieren. Dies kann ein Kalender-Eintrag, eine eingetroffene SMS oder ein anderes Ereignis sein. Diese Art der Benachrichtigung wird meist über eine Local Notification erzeugt. Eine Local Notification wird aufgrund eines lokal am Gerät auftretenden Ereignisses ausgelöst.

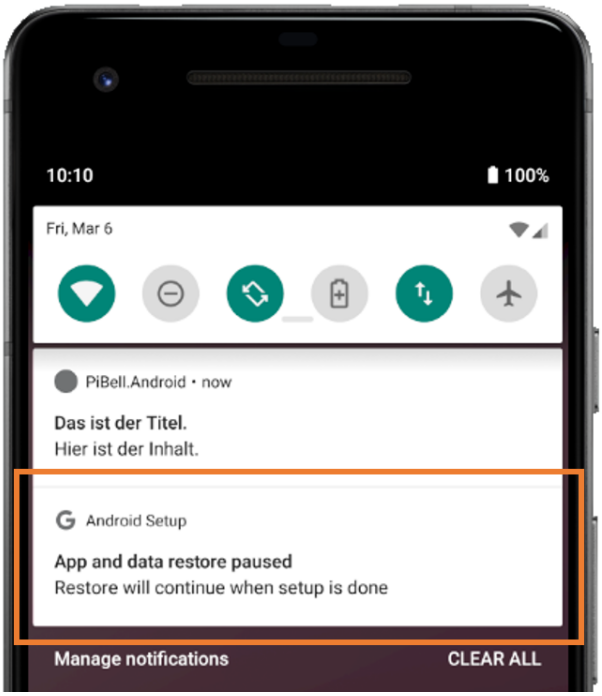

Abbildung 6.1: Lokale Benachrichtigung

Im Gegensatz dazu dient eine Push Notification dazu, den Benutzer über ein externes Ereignis auf einem anderen Gerät aufmerksam zu machen. Diese Art der Benachrichtigung findet meist anwendung in der Katastrophenwarnung, um den Benutzer vor einer herannahenden Gefahr zu warnen. Um den Benutzer also darüber zu informie-

ren, dass jemand an der Haustür klingelt, wird eine Push Benachrichtigung verwendet, da das Anläuten-Ereignis bei der Außenstation erfolgt.

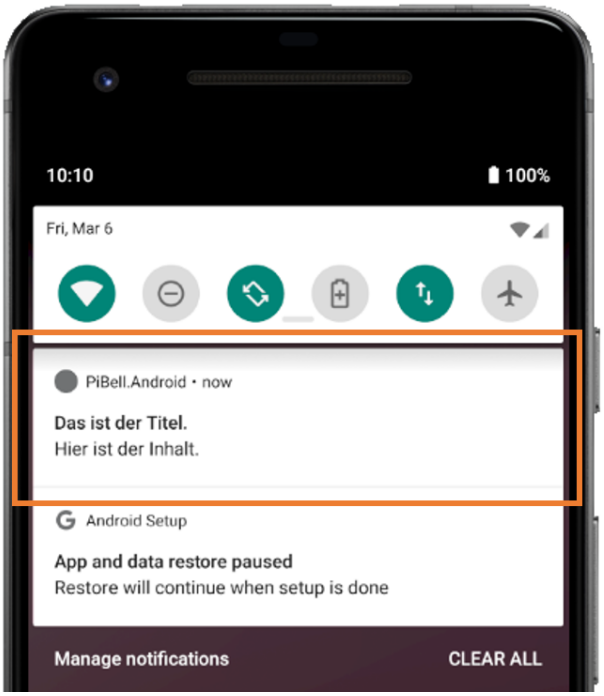

Abbildung 6.2: Push-Benachrichtigung

6.1.1 Funktionsweise

Eine Push Notification wird immer von einem sogenannten Push Notification Service an ale Zielgeräte geschickt. Dieser Service wird vom jeweiligen Betriebssystem-Hersteller bereitgestell. Im Fall der hier entwickelten App sind das Google und Apple, mit ihren Push Notification Services GCM und APNs.

Der Service ist dafür verantwortlich die Benachrichtigung an alle registrierten Zielgeräte zu schicken. Wenn das Zustellen nicht möglich ist, bricht der Service die Zustellung nach mehreren Versuchen für dieses Gerät ab. Alle anderen Geräte erhalten die Benachrichtigung. Daher ist es nicht es nicht sichergestellt, dass eine Push-Benachrichtigung bei jedem Gerät ankommt.

Zusätzlich zum Nachrichtentext können bei einer Push Notification zusätzliche Daten mitgeschickt werden. Diese können zum Beispiel die App über die genaue Benachrichtigungsursache informieren.

6.2 AppCenter Push

AppCenter Push hat in erster Linie die Funktion, den Zugriff auf die verschiedenen Push Notification Services zu vereinheitlichen und damit einfacher zu gestalten. Microsoft AppCenter verwaltet alle Server-API-Schlüssel, Zielgerätelisten und ähnliche Einstellungen, um den Prozess des Benachrichtigung-Sendens zu vereinfachen. AppCenter Push ist entweder über das Web-Interface unter `https://appcenter.ms/` oder über die AppCenter API erreichbar und verwendbar.

Die AppCenter Push SDK abstrahiert den ganzen Prozess der Geräte-Registration, des Registrieren des Eingangs-Events und versteckt alles hinter einer einzelnen simplen Funktion `AppCenter.Start(„App Secret", typeof(Push));`. Das App Secret ist der von AppCenter zugewiesene Schlüssel mit dem sichergestellt wird, dass das Gerät für das richtige AppCenter Projekt registriert wird. Ohne diese SDK müsste in der App viel mehr konfiguriert werden, was einen höheren Programmieraufwand und größere Fehlerrate darstellt. Außerdem würde das Rad neu erfunden werden.

In der entwickelten App wird die Appcenter-SDK-Version 2.6.4 verwendet. Zu beachten ist, dass Appcenter Push nicht mehr weiterentwickelt wird und irgendwann dieses Jahr der Service abgeschaltet wird, wie es John Wargo in seinem Blog-Post am 3. Februar 2020 schrieb. Als Alternative wird von Microsoft der Azure Notification Hubs dienst angeboten. Bevor Appcenter Push endgültig terminiert wird, will Microsoft detaillierte Anleitungen zum Umstieg auf Azure bereitstellen.

6.2.1 Konfiguration des SDK

Um Appcenter Push in der Applikation verwenden zu können müssen mehrere NuGet-Pakete installiert werden. Das AppCenter Paket beinhaltet alle Kernfunktionen wie AppCenter.Start(), das AppCenter-Push-Paket erweitert das Grundpaket um Push-bezogene Funktionen und Datentypen.

NuGet-Paket	PiBell	PiBell.Android	PiBell.iOS
AppCenter	Ja	Ja	Ja
AppCenter Push	Ja	Ja	Ja

Tabelle 6.1: Installation NuGet-Pakete

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.

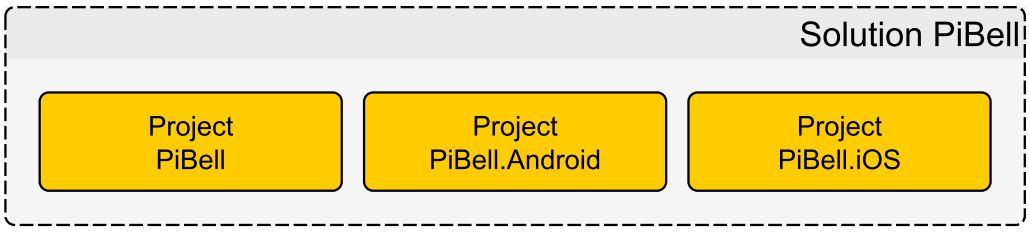

Abbildung 7.1: Aufbau der Solution

7.2 Portable Project

Im folgenden Teil werden die wesentlichen Teile des portablen Projektes PiBell beschrieben. In den einzelnen Programmteilen gibt es immer wiederkehrende Abläufe, wie Initialisierung, die nur einmal beschrieben werden. Die von Microsoft vorgefertigten Teile werden nicht behandelt. Der Fokus liegt damit auf den selbst erstellten Programmteilen. Die Programmteile werden generell in der selben Reihenfolge behandelt, wie sie vom Programm aufgerufen werden.

Textsegmente mit vorgestellter Zeilennummer sind direkt vom Programm eingefügt. Der beschreibende Text bezieht sich auf den Programmteil mit Hilfe der Zeilennummern.

7.2.1 App.xaml.cs

Im diesem Programmteil wird die Applikation grundlegend initialisiert. Die Initialisierung beinhaltet unter anderem das Laden aller benötigten Programm-Bibliotheken und NuGet-Pakete.

```
17 public App()
18 {
19     InitializeComponent();
20     Core.Initialize();
21     MainPage = new MainPage();
22
23
```

```
24      AppCenter.LogLevel = Microsoft.AppCenter.LogLevel.Verbose;
25      AppCenter.Start("android=2234efba-9d8c-45ec-bfad-f78aaa6ba632;" +
26                      "uwp={Your UWP App secret here};" +
27                      "ios={Your iOS App secret here}",
28          typeof(Analytics), typeof(Crashes), typeof(Push));
29  }
```

19-20: Die Funktionen InitializeComponent() und Core.Initialize sind vorgefertigte Funktionen; diese sind zu verwenden um die Applikation und alle benötigten Bibliotheken zu initialisieren.

21: Die Anweisung MainPage = new MainPage() erzeugt eine neue Oberfläche, mit der der Benutzer mit der App interagiert. Diese Oberfläche erscheint nach dem App-Start am Bildschirm und passt sich automatisch der physikalischen Bildschirmgröße an.

24-28: Mit AppCenter.LogLevel kann festgelegt werden, welche Log-Nachrichten angezeigt werden. In diesem Fall werden Alle Nachrichten aktiviert. Anschließend wird das AppCenter SDK mit AppCenter.Start(..) gestartet. Am Ende dieser Anweisung werden alle Module (Analytics, Crashes und Push), die dazu gestartet werden sollen angegeben.

7.2.2 MainPage.xaml

In einer XAML-Datei wird das Aussehen der Oberfläche nach dem WYSIWYM-Prinzip beschrieben. Das Format ähnelt sehr einer XML-Datei, in der Hinsicht, dass ein Element mit einer spitzen Klammer (<) beginnt und mit einem Schrägstrich und einer spitzen Klammer (/>) endet. Einige Elemente, wie zum Beispiel Listen oder ein Grid, können sogenannte Kind-Elemente beinhalten.

```
3   <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
4                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
5                xmlns:local="clr-namespace:PiBell"
6                xmlns:piBell="clr-namespace:PiBell;assembly=PiBell"
7                xmlns:shared="clr-namespace:LibVLCSharp.Forms.Shared;
                    assembly=LibVLCSharp.Forms"
8                x:Class="PiBell.MainPage"
9                Title="Main Page"
10               BackgroundColor="#ff1b1b1b">
```

3-8: Am Anfang der XAML-Datei wird der allgemeine Seiten-Typ, gemeinsam mit einigen Quell-Abkürzungen (Namenspräfix für Xamarin-fremde Elemente) definiert. In diesem Fall handelt es sich um eine simple ContentPage mit den Standard-Abkürzungen. Zur Verwendung der LibVLC wurde in Zeile 7 eine zusätzliche Quelle definiert. Die neu hinzugefügte Quelle erlaubt den Zugriff auf Elemente wie die VideoView, welche ein Video als Teil der Oberfläche darstellt.

9: Mit x:Class wird die Programmklasse angegeben, die den Code der Oberfläche beinhält. In dieser Klasse finden sich alle Event-Handler für jegliche Oberflächenereignisse, wie das drücken eines Buttons.

10: Das festlegen der Title-Variable ist in diesem Fall optional. Falls eine Master-Detail-Page-Struktur verwendet werden würde, wäre der Titel oben in der Kopfzeile der App zu sehen. Da es sich hier aber um ein Sigle-Page-Layout handelt, ist der Titel nicht zwingend notwendig.

11: Mit der Variable BackgroundColor wird die Hintergrundfarbe der Oberfläche definiert. Die gewünschte Farbe wird als HEX-Zahl mit dem Format Alpha-Rot-Grün-Blau angegeben. Der Alpha-Wert legt die Durchsichtigkeit der Farbe fest. Es wurde sich für ein dunkles Design entschieden, da dieses in einer wenig beleuchteten Umgebung die Augen besser schont.

```
12 <ContentPage.Content>
13     <Grid Margin="10" x:Name="MainGrid">
14         <Grid.RowDefinitions>
15             <RowDefinition Height=".7*" />
16             <RowDefinition Height="3*" />
17             <RowDefinition Height="2*" />
18         </Grid.RowDefinitions>
19         <Grid.ColumnDefinitions>
20             <ColumnDefinition Width="*" />
21         </Grid.ColumnDefinitions>
```

12: Mit dieser Zeile wird das Element ContentPage.Content geöffnet. Content steht hier für den gesamten Oberflächeninhalt, also alle Schaltflächen und Anzeigen.

13-21: Als erstes und einziges Kind-Element von ContentPage.Content wird die Komponente Grid gewählt. Diese ermöglicht es, mehrere Kind-Elemente in einer schach-

brettartigen Anordnung zu organisieren. Hierfür werden mit Hilfe der Row- und ColumnDefinitions die Höhen und Breiten der einzelnen Zeilen und Spalten festgelegt. Der Stern (*) in der Längenangabe bedeutet relatives Maß bezogen auf die verfügbare Breite bzw. Höhe.

```
23 <Entry x:Name="EditMrl" Text="{Binding Mrl, Mode=TwoWay}" Grid.Row="0
       " Grid.Column="0" TextColor="White" />
24
25 <shared:VideoView x:Name="VideoView" Grid.Column="0" Grid.Row="1"
       MediaPlayer="{Binding MediaPlayer}"/>
```

23: Hier wird das erste eigenständige Oberflächen-Element erzeugt. Es handelt sich um ein Text-Eingabe-Feld (Entry), mit dem zu Testzwecken die Serveradresse eingegeben werden kann. Dies ist notwendig, da die finale Serveradresse noch nicht bekannt ist und sich diese im Zuge der Entwicklung ständig ändert. Für den späteren Betrieb könnte dieses Problem umgangen werde, indem die aktuelle Server-Adresse per Push-Benachrichtigung mitgeschickt wird. Hier ist auch eine Anwendung des MVVM-Modells erkennbar: es wird mit Hilfe des Binding-Schlüsselwortes der Speicherort des angezeigten Textes festgelegt. Mode=TwoWay beschreibt die Richtung der Synchronisierung der Daten. Egal, ob sich der Wert im Speicher oder der Wert in der Anzeige ändert, wird die jeweils andere Seite aktualisiert. Mit Grid.Row und Grid.Column wird die Position im vorhin definierten Grid gesetzt.

25: Das zweite Oberflächen-Element, eine VideoView der LibVLC, wird hier erzeugt. Da die Komponente VideoView nicht standardmäßig in Xamarin enthalten ist, wird das am Anfang der Datei definierte Namenspräfix „shared:“ benötigt.

```
36 <ImageButton x:Name="BtConnect" Source="call.png" BackgroundColor="
       LimeGreen" Grid.Row="0"
37             Grid.Column="0" Margin="10"
38             Clicked="BtConnect_Clicked"/>
```

36: Ein ImageButton kann im Vergleich zu einem normalen Button als Inhalt ein Bild anzeigen. Die Bildquelle wird mit der „Source“-Property gesetzt. Es gibt mehrere mögliche Bildformate, wobei hier PNG aufgrund von Transparenz-Unterstützung gewählt wurde. Ohne Definition der „BackgroundColor“ würde der ImageButton mit der Default-Farbe grau gefüllt werden.

38: Mit dem EventHandler „Clicked“ kann die Funktion, die aufgerufen wird wenn der ImageButton gedrückt wird, spezifiziert werden.

```
43 <ImageButton x:Name="BtToggleSpeaker" Source="SpeakerMute.png"
      BackgroundColor="Transparent"
44              Grid.Row="1" Grid.Column="0" Margin="10"
45              Clicked="BtToggleSpeaker_Clicked">
46     <VisualStateManager.VisualStateGroups>
47         <VisualStateGroup x:Name="SpeakerStates">
48             <VisualState Name="Mute">
49                 <VisualState.Setters>
50                     <Setter Property="Source"
51                             Value="SpeakerMute.png" />
52                 </VisualState.Setters>
53             </VisualState>
54             <VisualState Name="Unmute">
55                 <VisualState.Setters>
56                     <Setter Property="Source"
57                             Value="SpeakerUnmute.png" />
58                 </VisualState.Setters>
59             </VisualState>
60         </VisualStateGroup>
61     </VisualStateManager.VisualStateGroups>
62 </ImageButton>
```

47-62: Mit dem VisualStateManager können für die meisten Oberflächen-Elemente bestimmte Zustände definiert werden (z.B. wenn sie gedrückt sind). Hier werden zwei Zustände definiert, die das angezeigte Bild dem Lautsprecher-Zustand anpassen.

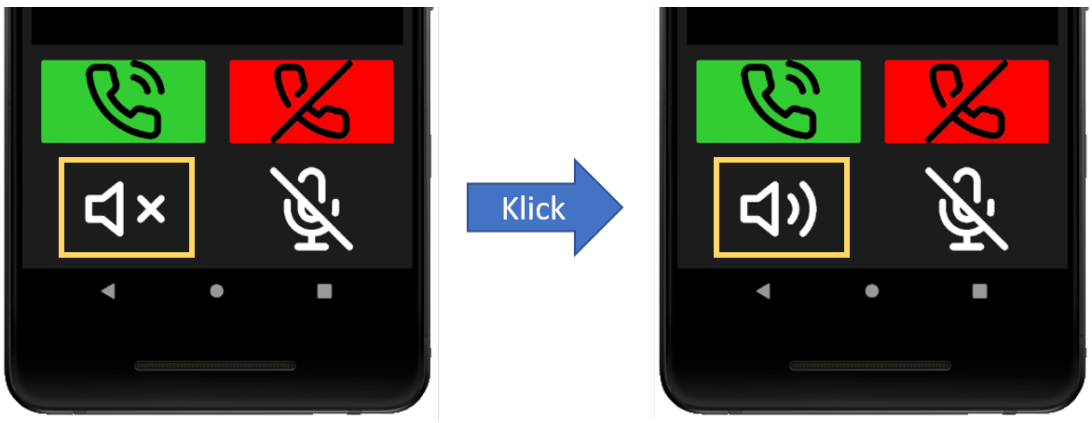

Abbildung 7.2: VisualStateManager

Die hier beschriebenen Konzepte werden mehrmals verwendet, um die Oberfläche aufzubauen. Die Datei als Gesamtheit ergibt die in Bild *7.3 App-Oberfläche* gezeigte Oberfläche.

Abbildung 7.3: App-Oberfläche

7.2.3 MainPage.xaml.cs

Dieser Teil beinhaltet den C#-Code, der für die Steuerung und Verwaltung der Oberfläche notwendig ist. Unter anderem wird hier das Verhalten beim Wechseln zwischen Hoch- und Querformat festgelegt.

```
36      if (!AppCenter.Configured)
37          Push.PushNotificationReceived += async (sender, e) =>
```

```
38          {
39  #if DEBUG
40              Console.WriteLine("DEBUG - Push-Notification recieved");
41  #endif
42              foreach (string key in e.CustomData.Keys)
43              {
44  #if DEBUG
45                  Console.WriteLine($"DEBUG - Custom Data: {key}:{e.
                        CustomData[key]}");
46  #endif
47                  if (key == "mrl")
48                      _player.Mrl = e.CustomData[key].ToString();
49              }
50
51              if (await DisplayAlert("Incoming Call", "Connect now?", "
                    Yes", "No"))
52              {
53                  _player.StartCall();
54                  _streamer.StartCall();
55              }
56          };
```

Hier wird das Verhalten bei eingehender Push Benachrichtigung festgelegt.

39,41,44,46: Die mit „#“ beginnenden Anweisungen sind sogenannte Preprocessor-Statements, mit denen dem Compiler spezielle Anweisungen gegeben werden können. Hier soll der Programmteil, der sich zwischen `#if DEBUG` und `#endif` befindet, nur im DEBUG-Modus verarbeitet werden. Wenn ein anderer Compiler-Modus ausgewählt ist, werden diese Teile ignoriert.

42-49: Mit einer Schleife werden alle Zusatzdaten der Benachrichtigung überprüft. Wenn der Datensatz `mrl` enthalten ist, wird dessen Wert in `_player.Mrl` gespeichert. Der Wert stellt die Server-Adresse dar, von der der Video-Stream abgerufen werden soll.

51-55: Der Benutzer wird per Popup gefragt, ob der Anruf entgegen genommen werden soll (Abb. *7.4 Abfrage Anrufannahme*).

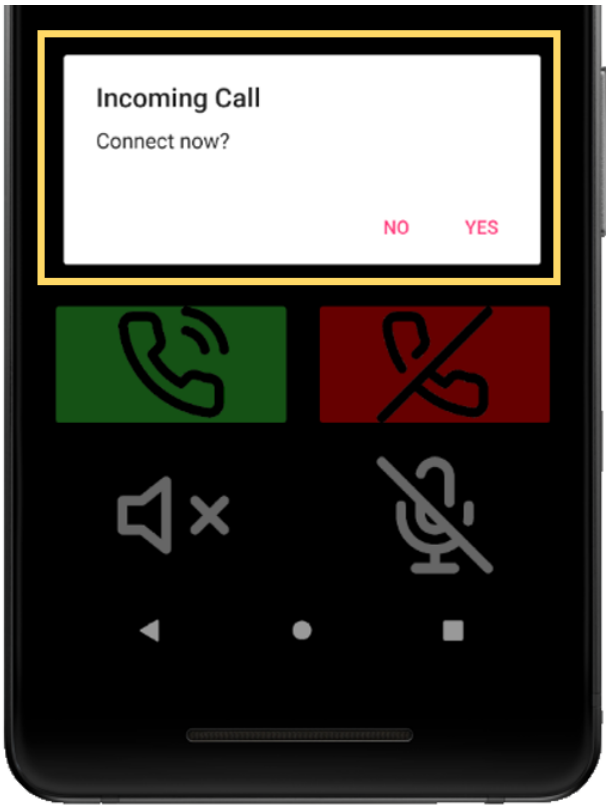

Abbildung 7.4: Abfrage Anrufannahme

Wenn die Applikation in den Hintergrund geht, verliert die LibVLC die Referenz auf die VideoView und kann den Video-Stream nicht mehr darstellen. Daher muss die Wiedergabe gestoppt werden, sobald die App in den Hintergrund geht und neu gestartet werden, wenn die App wieder im Vordergrund ist. Da die Benachrichtigung über den Vorder- bzw. Hintergrundstatus der Applikation nur über das native Projekt funktioniert, muss diese Information vom nativen Projekt zum portablen Projekt übertragen werden. Dies geschieht mit der MessagingCenter-Komponente.

```
57 MessagingCenter.Subscribe<string>(this, "OnPause", app =>
58 {
59     _wasConnected = _player.MediaPlayer.IsPlaying;
60     _player.MediaPlayer.Pause();
61     _prevPosition = _player.MediaPlayer.Position;
62     _player.MediaPlayer.Stop();
63
64     MainGrid.Children.Remove(VideoView);
65 });
66
67 MessagingCenter.Subscribe<string>(this, "OnRestart", app =>
```

```
68 {
69     RegenerateVideoView();
70     VideoView.MediaPlayer = _player.MediaPlayer;
71     if (_wasConnected)
72     {
73         _player.MediaPlayer.Play();
74         _player.MediaPlayer.Position = _prevPosition;
75     }
76
77     _prevPosition = 0;
78 });
```

57-65: Dieses Ereignis wird aufgerufen, wenn die Applikation in den Hintergrund wechselt. Bevor dies geschieht, wird die aktuelle Wiedergabe-Position in einer Variablen gespeichert und die Wiedergabe gestoppt. Anschließend wird die VideoView von der Oberfläche entfernt, damit sie später neu hinzugefügt werden kann.

67-78: Das `OnRestart`-Event wird aufgerufen, wenn die App vom Hintergrund wieder in den Vordergrund geht. Hier geschieht der selbe Vorgang wie davor, jedoch in sinngemäß umgekehrter Reihenfolge. Als erstes wird die VideoView der Oberfläche wieder hinzugefügt, damit die LibVLC eine Referenz dazu bilden kann. Falls davor ein Video gespielt wurde, wird dieses fortgesetzt und die Variable für die vorherige Wiedergabeposition wird gelöscht.

```
84 BindingContext = _player;
```

84: Der BindingContext gibt für das MVVM-Modell an, wo die Variablen, die mit der Oberfläche verknüpft sind, sich befinden. In diesem Fall sind alle Variablen in der Player-Instanz `_player` zu finden.

An folgender Funktion wird das Verhältnis von Oberflächenereignissen mit dem entsprechenden Code-Behind erklärt.

```
109 private void BtConnect_Clicked(object sender, EventArgs e)
110 {
111     _player.StartCall();
112     _streamer.StartCall("192.168.43.78");
113 }
```

109: Eine Funktion für ein Button-Event hat grundsätzlich als Rückgabe-Typ `void`, also nichts. Auf die Funktion kann nur aus der selben Klasse zugegriffen werden, was mit dem `private`-Schlüsselwort bekanntgegeben wird. Als Parameter werden Informationen zum Event-Auslöser und zum Event selbst mitgegeben.

111-112: Wenn der grüne „Anrufen“-Button gedrückt wird, beginnen `_player` und `_streamer` damit, die Daten abzurufen bzw. bereitzustellen.

7.2.4 MediaClasses.cs

7.3 PiBell.Android

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.4 PiBell.iOS

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

8 Testen

9 Ausblick

9.1 Gesicherte Verbindung

Literatur

[1] A. S. Incorporated. (Okt. 2011). „Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap“, Adresse: https://web.archive.org/web/20120413181632/http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html.

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

[3] 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.

[4] 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.

[5] 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.

[6] J.-B. Kempf. (Feb. 2016). „15 years of VLC and VideoLAN“, Adresse: http://www.jbkempf.com/blog/post/2016/15-years-of-VLC.

[7] Josh Smith (Microsoft). (Dez. 2016). „Patterns - WPF Apps With The Model-View-ViewModel Design Pattern“, Adresse: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern.

[8] Martin Finkel (VideoLAN). (März 2020). „LibVLCSharp README.md“, Adresse: https://wiki.videolan.org/LibVLC.

[9] VAStreaming. (März 2020). „highly optimized and efficient media streaming, encoding, decoding and processing libraries for developers“, Adresse: https://www.vastreaming.net/.

[10] 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

CPU Central processing unit

CSS Cascading Style Sheets

NetFx .NET-Rahmenwerk (.NET-Framework)

EAGLE Einfach Anzuwendender Grafischer Layout-Editor

FI Fachspezifische Softwaretechnik

HTTP Hypertext Transfer Protocol

IC integrierte Schaltung (Integrated Circuit)

IETF Internet Engineering Task Force

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

URL Uniform Resource Locator

UWP universelle Windows-Plattform (Universal Windows Platform)

Abbildungsverzeichnis

Tabellenverzeichnis

Teil III

Appendix

A Verwendete Entwicklungswerkzeuge

B Datenblätter

C Kostenübersicht

D Fertigungsdokumentation