KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS BRUGGE

Studie en ontwikkeling van een softwaregebaseerd systeem met gecentraliseerde opname voor opleiding van tolken

Klaas Govaerts

Promotor: Sammy Verslype

Co-promotor: Xavier De Donder

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: Elektronica-ICT, afstudeerrichting ICT

Academiejaar 2018-2019

Voorwoord

Deze thesis beschrijft het inhoudelijke werk dat werd geleverd bij Televic Education in Izegem, tijdens een vrijwillige stage van 10/09/2018 tot en met 21/09/2018 en een bedrijfscontact van 11/2/2019 tot en met 22/3/2019. Daarna volgden nog vier dagen om veiligheidslekken te dichten.

Er waren twee promotoren die mij hierbij bijstonden, één promotor van KU Leuven en één co-promotor van het bedrijf Televic Education.

Eerst en vooral wens ik deze promotoren te bedanken.

Xavier De Donder (Televic Education) was co-promotor. Hij heeft zijn best gedaan om mij op weg te zetten en was tijdens het bedrijfscontact altijd behulpzaam en bereikbaar voor vragen. Hij was een zeer toegewijde co-promotor.

Sammy Verslype (KU Leuven) was promotor. Bij hem kon ik altijd terecht met vragen over het schrijven van de tekst en hij volgde de vorderingen van de masterproef goed op. Hij gaf ook nuttige feedback over de voorlopige masterproeftekst.

Verder wil ik ook het bedrijf Televic Education bedanken om mij de mogelijkheid te bieden mijn masterproef bij hun af te leggen.

Het was een interessant onderwerp, dat nauw aansloot bij mijn interessesfeer. Op deze manier kon ik de kennis die ik heb verworven tijdens mijn opleiding aan de KU Leuven, inzetten voor een concreet project.

Uiteraard was er ook een groot deel onderzoek en zelfstudie. Zo is onder meer mijn kennis over C# en het .NET framework sterk toegenomen tijdens dit project.

Daarnaast heb ik ook geleerd hoe ik grotere opgaven kan aanpakken. De aanpak was om telkens een klein stuk functionaliteit toe te voegen en de werking hiervan onmiddellijk te testen. Er waren manuele testen en waar mogelijk ook unit tests.

Voor mij is het belangrijk dat technologie geïntegreerd wordt binnen onderwijs om het lesgebeuren aangenamer en efficiënter te maken Televic Education is wat dat betreft een voorloper en vandaar ook de keuze om bij dit bedrijf mijn masterproef af te leggen. Het was motiverend om te weten dat ik met de code die ik heb geschreven, een nuttige bijdrage kon leveren aan interpreterQ, een nieuw programma voor opleiding van tolken.

Tenslotte wil ik ook de collega’s bij Televic Education bedanken voor het warme onthaal.

Samenvatting

Tijdens deze masterproef werd software ontwikkeld voor opleiding van conferentietolken.

Tijdens een conferentie is er een persoon die het woord heeft (het “FLOOR” signaal). Dit geluidssignaal dient vertaald te worden door tolken. Deze tolken kunnen rechtstreeks naar het FLOOR signaal luisteren, of een tolk kan ook kiezen om naar een andere tolk te luisteren indien deze de taal van de spreker niet kent.

Het bedrijf Televic Conference (uit Izegem) levert reeds een hardwaregebaseerde conferentieopstelling.

Bij Televic Education (ook gevestigd in Izegem) werd eveneens een hardwaregebaseerde opleidingsopstelling gemaakt, gebaseerd op de conferentieopstelling van Televic Conference.

Het doel van de masterproef bij Televic Education was om deze opleidingsopstelling te simuleren in software, door middel van applicaties voor een webbrowser. Verder werden ook functionaliteiten toegevoegd, specifiek voor het opleiden van tolken binnen een klassikale context.

De leerlingen (tolken in opleiding) en leerkracht beschikken elk over een pc, waarop de applicaties worden uitgevoerd. Elke pc binnen de opstelling is verbonden met eenzelfde netwerk. Daarnaast maakt elke pc (met uitzondering van deze van de leraar) gebruik van een *Dante AVIO* adapter voor audiostreaming. Op de pc van de leraar wordt *Dante Virtual Soundcard* gebruikt. Als alternatief voor een pc kan in de toekomst geopteerd worden voor een tablet.

Tijdens de masterproef werden **3 verschillende webapplicaties** ontwikkeld.

De eerste webapplicatie is de “**Interpreter App**”, voor tolken in opleiding. Die stelt de gebruiker in staat om te luisteren naar het FLOOR signaal, of naar een andere tolk naar keuze en vervolgens de vertaling in te spreken via de microfoon. Eerst dient de student zijn naam in te geven via een loginscherm.

De tweede webapplicatie is de “**Delegate App**”. Die stelt de gebruiker in staat om in elk oor te luisteren naar een geluidssignaal naar keuze.

De derde webapplicatie is de “**Chairman App**”, voor leraars. Die bouwt verder op de Delegate App en biedt extra functionaliteit. De leraar kan een intercom starten, dit is een bidirectioneel gesprek met een student naar keuze, in het rechteroor. De leraar kan ook een “Activity” starten, dit is een klassikale opgave waarbij iedere student hetzelfde signaal hoort en dit dient te vertalen. Het is mogelijk om, tijdens een Activity, de vertalingen van de studenten op te nemen naar één gecentraliseerde server. Tijdens een opname, heeft de leerkracht de mogelijkheid om “markers” toe te voegen op een bepaald tijdstip bij een bepaalde student, bijvoorbeeld wanneer de student in kwestie een fout maakt.

De webapplicaties werden ontwikkeld op basis van HTML, CSS en JavaScript.

De webserver waarmee de gebruiker verbindt, werd geprogrammeerd in C# door middel van Visual Studio.

De audiostreaming tussen de verschillende pc’s gebeurt door middel van Dante, een protocol van het bedrijf Audinate. Op basis van de keuzes die gebruikers hebben ingesteld, worden automatisch de juiste audiostreams opgezet.

Kort samengevat: de hardware wordt nu gesimuleerd op de pc van de gebruiker. Dit biedt een low-cost alternatief voor de reeds bestaande hardwareopstelling.

Abstract

Keywords: Interpreter training, Audio streaming, Dante, Web Application Development, Televic

During this master thesis, software was developed for conference interpreters.

During a conference, the person currently speaking is called the “FLOOR” channel. This audio signal has to be translated by interpreters. An interpreter can listen to this FLOOR signal directly. An interpreter can also listen to another interpreter if he doesn’t know the language of the person currently speaking.

The Company “Televic Conference” (in Izegem, Belgium) already supplies a hardware based conference setup.

Televic Education (also in Izegem) supplies a hardware based interpreter training system, based on the conference setup by Televic Conference.

The goal of this master thesis at Televic Education was to simulate the interpreter training system in software by using web applications. Additional functionalities were added as well, specifically for the training of interpreters in a classroom context.

The students (interpreters in training) and the teacher each have a PC on which the web application will be executed. Each PC in the setup is connected to the same network, and has a static IP address. Furthermore, each PC (except the teachers’ PC) uses a *Dante AVIO* adapter for audio streaming. As an alternative for a PC, a tablet could be used in the future.

During the internship at Televic, **three types of web application** were developed:

*The first web application is the “**Interpreter App**”, for interpreters in training. Each Interpreter App has a booth number, starting from 1. Booth number 0 corresponds to the FLOOR channel. The users are able to listen to the FLOOR channel, or to the first 6 interpreter booths, and to translate this audio using their microphone. During the first connection, the students must enter their name via a login screen.

*The second web application is the “**Delegate App**”. This application allows the user to listen to an interpreter booth of their choice (and the FLOOR channel). A different signal can be chosen for each ear, using a dropdown list. A Delegate App does not have a microphone.

*The third web application is the “**Chairman App**”, for teachers. This application adds extra functionalities to the Delegate App. The teacher can start an intercom, this is a bidirectional conversation with a student of his choice, in the right ear. The teacher can also start an “Activity”, this is a classroom assignment in which each student hears the same audio signal and has to translate it. During an Activity, it is possible to record the translations of the students to one centralized server. During the recording, a teacher is also able to add “markers” at a specific time for a specific student, when this student makes an error.

When a user connects to the “virtual_iq_server” webserver (via HTTP GET), one of the 3 applications will be returned. The type of application that will be returned by the webserver is based on the IP address of the client. The mapping between IP address and application type is configured in the webserver.

After the user has connected, a WebSocket connection will be opened for realtime communication. All user commands (such as a relay change) will be sent through this WebSocket via a text command.

In the opposite direction, the server will also send status updates, formatted as JSON. Different types of messages can be sent. The first field contains the MessageType, and the second field contains the actual content. An example of a message, is a list of interpreters that are currently logged in.

The **recording server** uses Dante Virtual Soundcard to receive the audio streams. Dante Virtual Soundcard allows for 8 stereo virtual soundcards, corresponding to 16 input & 16 output channels. To start and stop the recording, an Application Programming Interface (API) will be provided, using Windows Communication Foundation (WCF). This WCF API allows for bidirectional communication. The client can send commands to the server, and the server will give status updates via callbacks.

First, the client has to provide a list of all recordings that should be executed. For each recording, the Dante Channel number should be provided, the name of the corresponding student, etc. Of course there are also commands to start and stop a recording. When starting a recording, a title has to be provided.

When a recording has successfully started/stopped (or in case an error has occurred), the client will receive a callback.

A new folder will be created, with the recording title. Each recording file will have the name of the corresponding student.

Furthermore, a single JSON file will be generated for each recording. This JSON file contains a list of all data that was provided by the API, as well as a chronological list of markers for all students.

The most important part of the setup is called the **virtual_iq_server**. The four functionalities are: maintaining the configuration, functioning as a webserver, controlling the Dante Controller, and controlling the recording API.

The configuration contains multiple entries, each corresponding to a PC in the network. A part of the configuration is static (such as the IP address, and the web application type). This part of the configuration will be kept in a JSON file, stored on the hard drive. Another part of the configuration is dynamically changed by user input such as the name of the interpreter that is logged in, or the chosen audio channel. The dynamic config is only stored in RAM.

The virtual_iq_server also functions as webserver, meaning that the users will connect to its IP address. The HTTP GET request will be parsed and the proper response will be provided, based on the config.

Based on the configuration, a list of desired audio subscriptions is automatically generated (desiredSubscriptions). This list is compared to the current subscriptions. The Dante Controller API is called to reflect the difference between those two.

Because the virtual_iq_server keeps track of the names of all the interpreters that are logged in, the virtual_iq_server will also control the recording server API.

The web applications were developed using HTML, CSS & JavaScript.

The webserver the user connects to, was developed in C# by using Visual Studio.

The audio streaming between the different PC’s happens by using Dante, a protocol by Audinate. Based on the users’ choices, the correct audio streams will automatically be configured.

To summarize: the hardware will be simulated on the user PC. This provides a low-cost alternative for the existing hardware setup.

INHOUD

Lijst met afkortingen

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASIO	Audio Stream Input/Output
CSS	Cascading Style Sheets
DLL	Dynamic-link library
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICC	Interpreter Control Center
IP	Internet Protocol
JSON	JavaScript Object Notation
LINQ	Language-Integrated Query
REST	REpresentational State Transfer
RSS	Really Simple Syndication
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
WASAPI	Windows Audio Session API
WCF	Windows Communication Foundation
WDM	Windows Driver Model
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XSS	Cross-site scripting

1 INLEIDING

1.1 Het bedrijf Televic

Televic is een bedrijf met hoofdzetel in Izegem. Hun specialisatie bestaat uit communicatieapparatuur voor specifieke niche markten. Binnen de Televic Group zijn er 4 verschillende divisies, namelijk Televic Healthcare, Televic Rail, Televic Conference en Televic Education. Deze masterproef werd ontwikkeld binnen deze laatste divisie.

Televic Education stelt verschillende oplossingen ter beschikking om technologie te integreren in het lesgebeuren. Een voorbeeld hiervan is assessmentQ. Dit is software om onder meer examens mee af te nemen.

Het werk dat in deze masterproef geleverd werd, zal deel uitmaken van het nieuwe product “interpreterQ”. Dit is software voor het opleiden van tolken.

1.2 Probleemstelling

Televic Education beschikt over een hardwaregebaseerde opstelling om conferentietolken op te leiden, gebaseerd op de opstelling van een reële conferentie.

Tijdens een conferentie levert de persoon die aan het woord is het “FLOOR” geluidssignaal. Daarnaast zijn er een aantal tolken die dit geluidssignaal vertalen naar een specifieke taal. Een tolk kan kiezen om rechtstreeks naar het FLOOR signaal te luisteren. Indien de tolk de taal van de spreker niet kent, kan een tolk ook kiezen om naar een andere tolk te luisteren. Op die manier gebeurt de vertaling via een omweg (bijvoorbeeld via het Engels).

Elke tolk heeft een interpreter desk ter beschikking.

Figuur 1.1: Lingua Interpreter Desk

In de reeds bestaande opleidingsopstelling van Televic Education heeft iedere student een microfoon ter beschikking. Dit is bijvoorbeeld de *Lingua Interpreter Desk*, geproduceerd door Televic Conference. Daarnaast beschikt iedere student ook over een hoofdtelefoon.

De leraar is in staat om opnames te starten. De studenten krijgen in dat geval een geluidsfragment te horen, dat ze dan simultaan dienen te vertalen en in te spreken in deze microfoon.

In de bestaande opstelling is er aan iedere *Interpreter Desk* ook een pc gekoppeld, die zijn eigen geluidssignaal decentraal opneemt. Vervolgens is de leraar in staat om de opnames te beluisteren en te verbeteren.

Initieel waren er vanuit Televic Education twee aparte masterproefvoorstellen, namelijk:

- “Software for centralized recording, monitoring and annotation in interpreter training”
- ”Virtual interpreter training system”

Deze twee masterproefvoorstellen werden gecombineerd tot één masterproefvoorstel, namelijk:

- “Virtual interpreter training system with centralized recording”

In deze masterproef werd dit laatstgenoemde voorstel verder uitgewerkt.

Deze masterproef omvat de studie en ontwikkeling van een software gebaseerd systeem met gecentraliseerde opname voor opleiding van tolken.

Het doel van de masterproef is om enerzijds de bestaande hardwaregebaseerde opstelling te simuleren in software, om zo een low-cost alternatief te bieden. Anderzijds worden nog een aantal functionaliteiten toegevoegd, specifiek bedoeld voor het opleiden van tolken.

In de nieuwe opstelling is er ook een centrale opnameserver, die de audiostreams van de tolken kan inlezen en opslaan als audiobestand, waarna de leraar deze kan verbeteren.

Voor de audiostreams tussen de verschillende gebruikers wordt gebruik gemaakt van Dante.

In de verdere tekst worden deze applicaties verder in detail uitgelegd.

1.3 Werking audiokaarten

1.3.1 Algemene werking van audiokaarten

Er bestaan twee soorten geluidsapparaten:

- Afspeelapparaten: Dit is de audio output van de pc (bijvoorbeeld een luidspreker).
- Opnameapparaten: Dit is audio input van de pc (bijvoorbeeld een microfoon).

Figuur 1.2: De verschillende types geluidsapparaten

1.3.2 Werking van Dante

Dante is een protocol voor audiostreaming. De ongecomprimeerde audio wordt verzonden via Layer 3 IP pakketten.

De communicatie gebeurt via een Dante Adapter. Elke adapter heeft een aantal inputkanalen en outputkanalen. Daarnaast heeft iedere adapter een unieke naam.

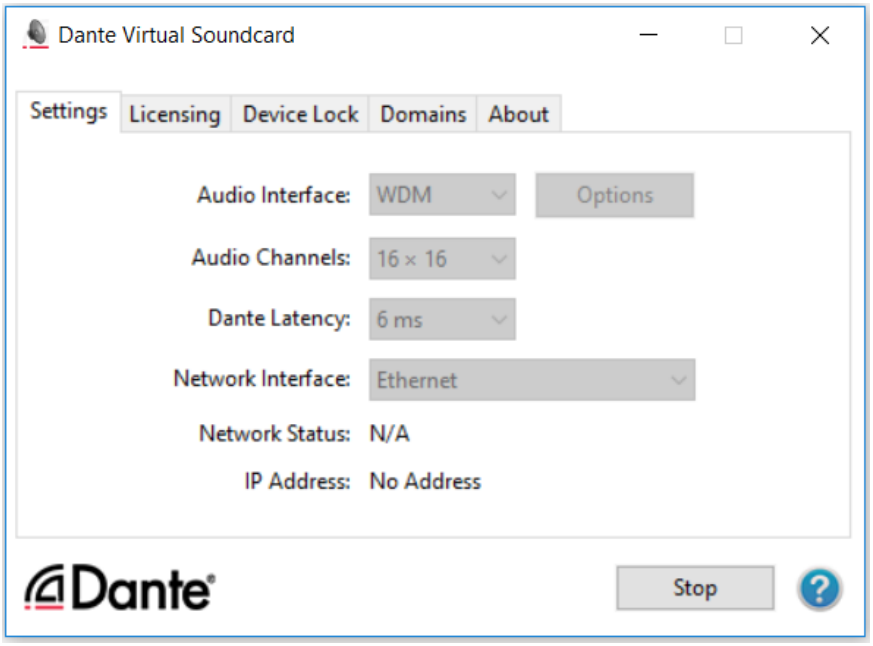
In deze opstelling zal er gebruik gemaakt worden van twee soorten Adapters, namelijk *de Dante Virtual Soundcard* en de *Dante AVIO USB*.

1.3.2.1 Dante Virtual Soundcard

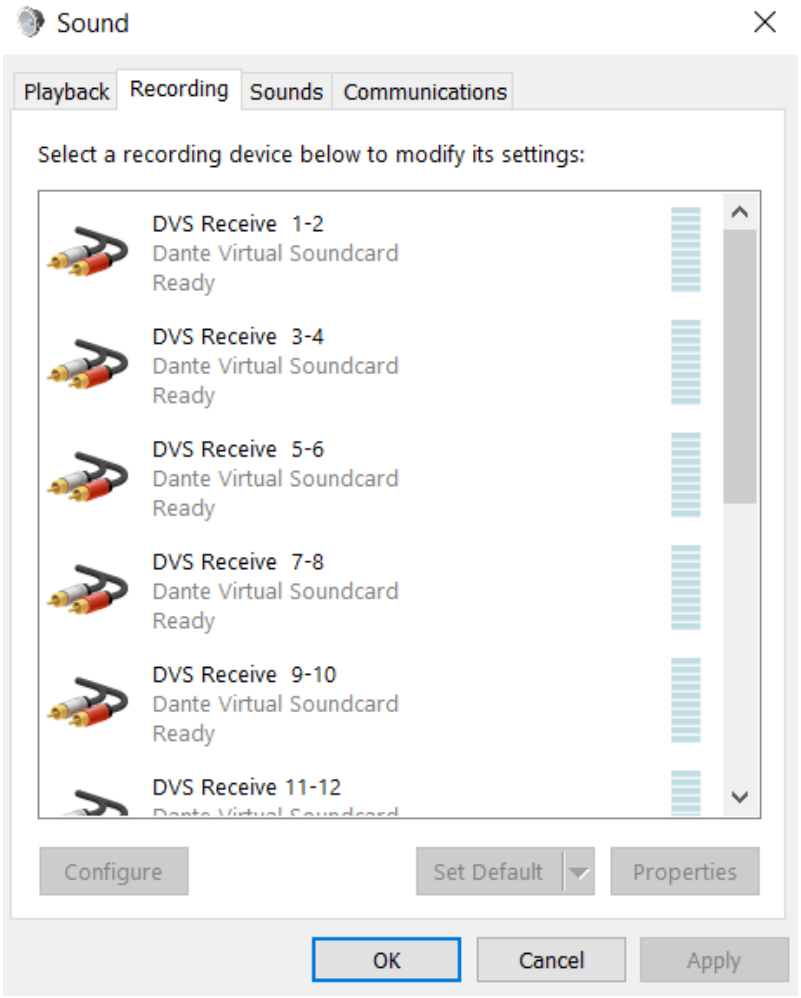
Dante Virtual Soundcard is een softwareprogramma dat dient geïnstalleerd te worden op de pc.

Binnen Dante Virtual Soundcard bestaat de keuze voor Windows Driver Model (WDM). In dat geval komen er 8 virtuele afspeelapparaten en 8 virtuele opnameapparaten bij. Deze virtuele apparaten maken gebruik van stereogeluid, wat dus betekent dat we 16 inputkanalen hebben en 16 outputkanalen.

De Dante Virtual Soundcard werd tijdens de testen uitgevoerd op een Televic bedrijfslaptop genaamd “TLV-TRA-RES9”.



Figuur 1.3: De GUI van Dante Virtual Soundcard



Figuur 1.4: De opnameapparaten van Dante Virtual Soundcard

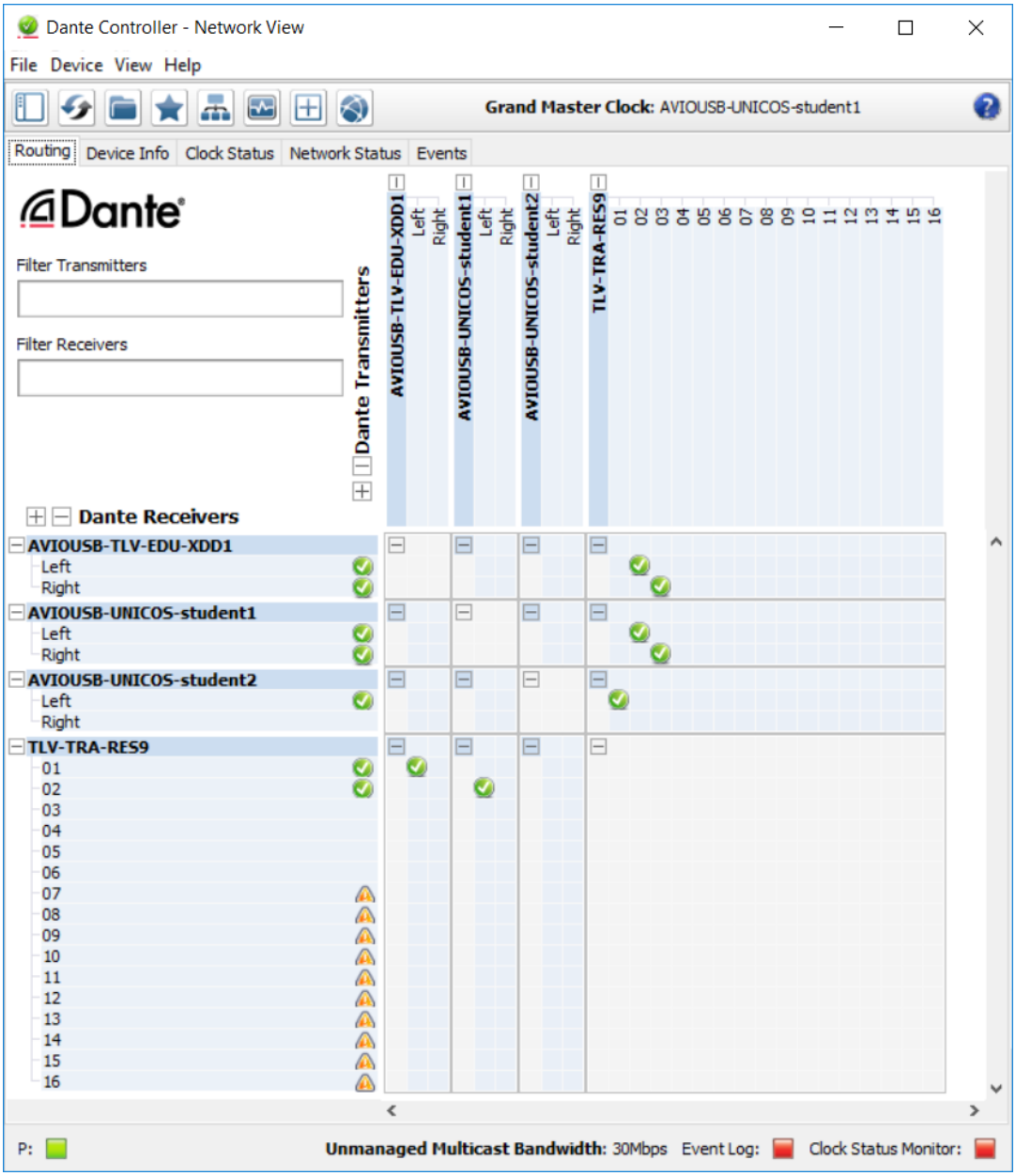
1.3.2.2 Dante AVIO USB

Dante AVIO is een fysieke adapter en vereist dus geen installatie op de pc. De RJ-45 poort wordt verbonden met het netwerk. De USB poort wordt verbonden met de pc zelf. Op de pc verschijnt een afspeelapparaat en opnameapparaat. Er zijn dus 2 inputkanalen en 2 outputkanalen.

Het is mogelijk om de Dante AVIO adapter een naam te geven. In deze testopstelling werd gekozen om de naam van elke Dante AVIO adapter te laten starten met “AVIOUSB-”.

Figuur 1.5: Dante AVIO USB

1.3.2.3 Dante Controller



Figuur 1.6: Schermafbeelding van Dante Controller

De apparaten die op het Dante netwerk zijn aangesloten, zijn in staat om audiostreams te verzenden en/of te ontvangen. Door middel van één centrale applicatie, de Dante Controller, kunnen deze streams geconfigureerd worden. In een matrix structuur (zie Figuur 1.6:) kan een mapping gemaakt worden tussen een zender en ontvanger. Het gaat hier telkens om streams van een mono signaal, van een outputkanaal naar een inputkanaal.

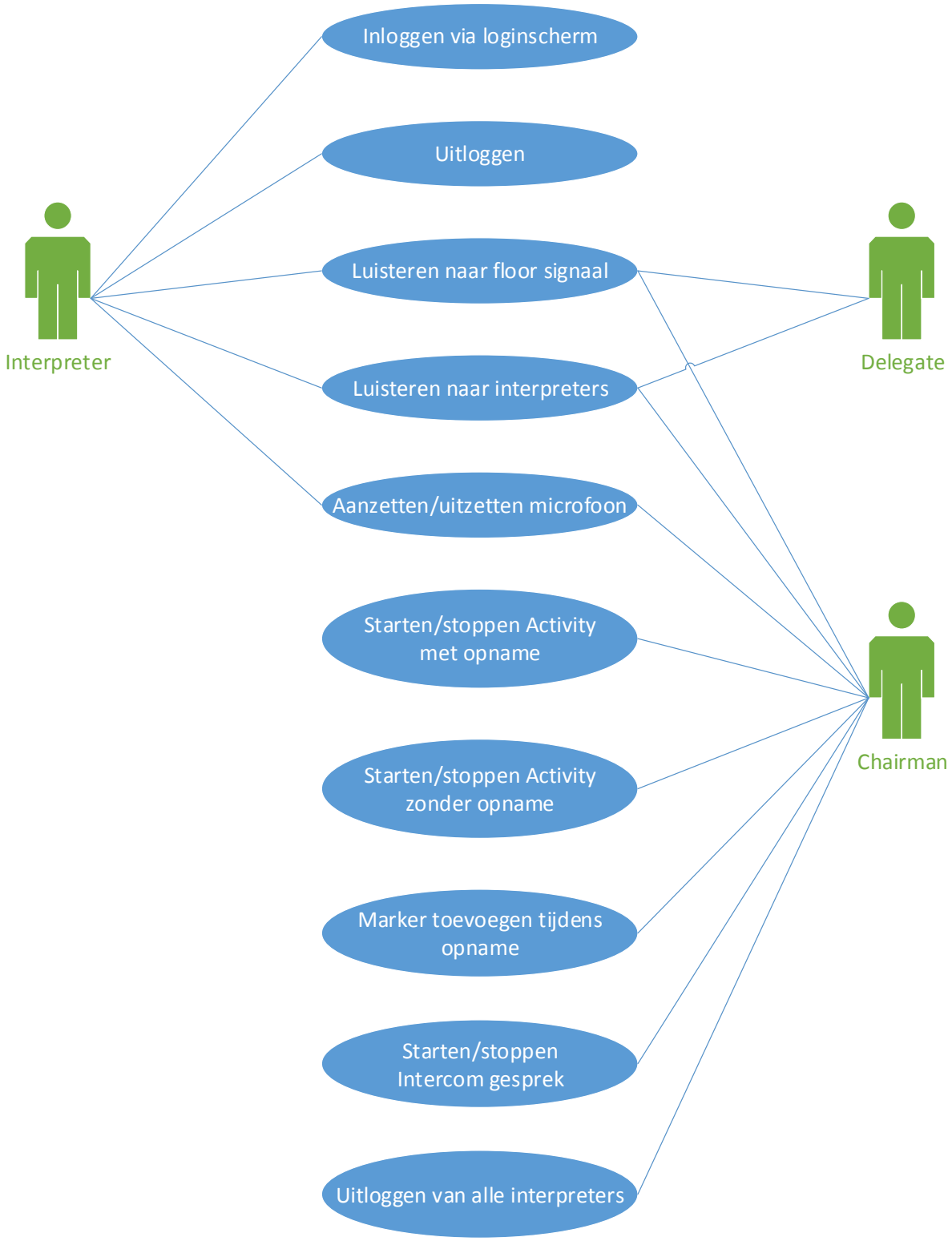
Via een inputkanaal kan de gebruiker slechts naar één outputkanaal tegelijk luisteren. Eenzelfde outputkanaal kan echter wel door meerderen tegelijk beluisterd worden.

De Dante controller stelt ook een API ter beschikking.

De communicatie tussen de microfoons en de opnameserver gebeurt door middel van Dante audiostreams.

2 DE OPSTELLING

2.1 Functionaliteit van de opstelling



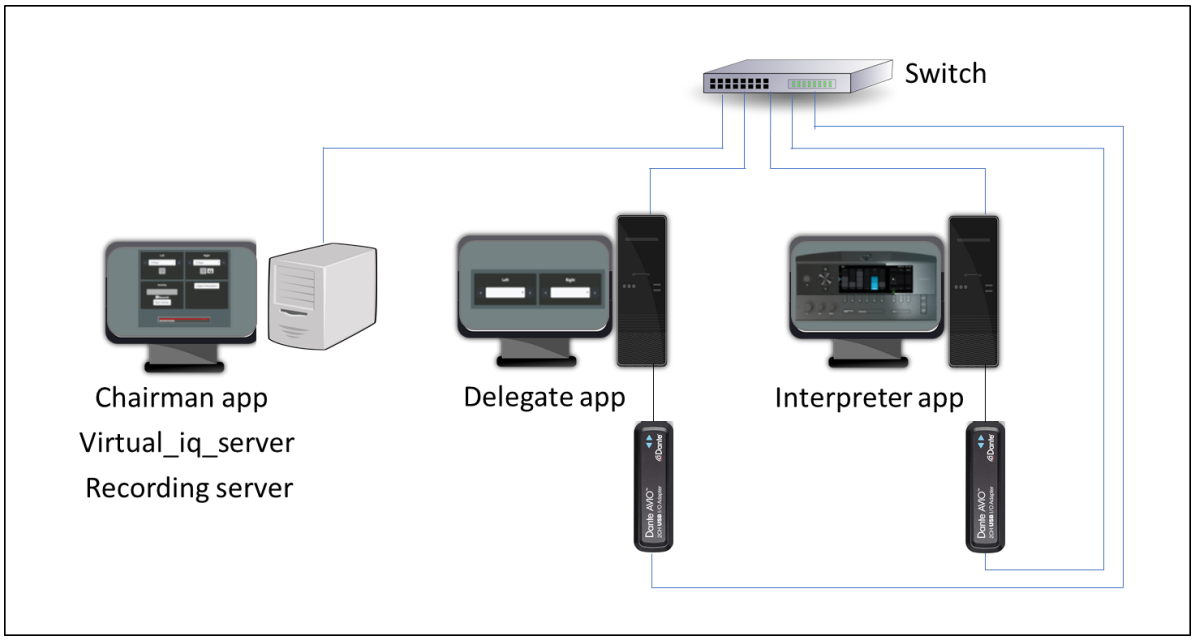
Figuur 2.1: Use case diagram van de verschillende gebruikers

Er is één webserver; de “virtual_iq_server”. Deze is host voor de 3 verschillende types webapplicatie. Een voor de interpreters (tolken), een voor delegates (luisteraars) en chairmans (leraars). Zie Figuur 2.1. De verschillende webapplicaties zijn:

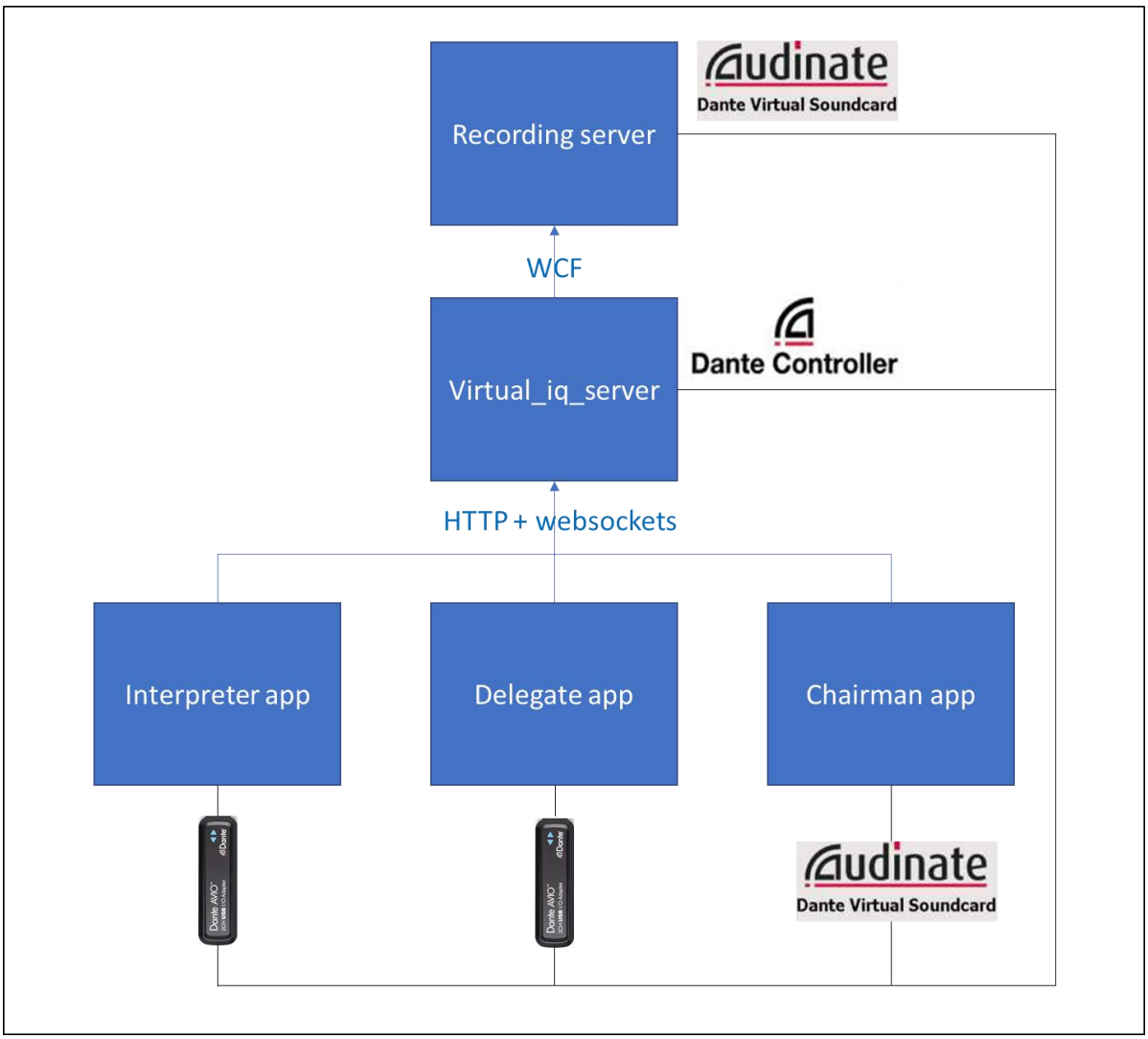
- De Interpreter App is bedoeld voor studenten. Iedere Interpreter App heeft een boothnummer, genummerd vanaf 1. De belangrijkste functionaliteit is de volgende:
    - Voor de student kan verbinden met de Interpreter App, dient hij eerst zijn naam in te geven, via een apart login scherm.
    - Een interpreter kan luisteren naar het floor signaal. (boothnummer 0)
    - Een interpreter kan luisteren naar andere tolken. (boothnummer 1 t.e.m. 6)
    - Een interpreter kan zijn microfoon aan- en uitzetten.
- De Delegate App is bedoeld voor leraars. Deze app is in staat om te luisteren naar tolken en naar het FLOOR signaal. Een verschillend signaal kan gekozen worden voor het linkse oor en het rechtse oor. Een Delegate App heeft geen microfoon.
- De Chairman App is bedoeld voor leraars. Dit is de Delegate App, maar dan met extra functionaliteit:
    - Een Chairman App heeft wel een microfoon. Deze kan aan- en uitgezet worden.
    - Een chairman kan een “Activity” starten. Dit is een klassikale opgave waarbij iedere tolk hetzelfde signaal hoort.
    - Er is ook de mogelijkheid om tijdens een “Activity” een opname te starten, zodat de vertalingen van de studenten achteraf beoordeeld kunnen worden door de leraar.
    - Een chairman kan tijdens een opname een marker toevoegen. Nuttig als de studenten bijvoorbeeld een fout in de uitspraak maken, waar de leraar later wil op terugkomen.
    - Een chairman kan een intercom starten, oftewel een bidirectioneel gesprek met een bepaalde student, op het rechteroor.

2.2 Structuur van de opstelling

In de opstelling wordt gebruik gemaakt van een netwerk van pc’s, met ieder een vast IP adres. In Figuur 2.2 wordt de fysieke opstelling weergegeven, in Figuur 2.3 de logische opstelling.



Figuur 2.2: De fysieke opstelling



Figuur 2.3: De logische opstelling

Er zijn een aantal client pc’s die verbonden zijn met hetzelfde subnet als de server. Wanneer de clients verbinden met de server worden ze op basis van hun IP adres doorverwezen naar één van de 3 types webapplicatie. De mapping tussen het IP adres en de bijhorende applicatie staat in een vast configuratiebestand op de virtual_iq_server.

De audiostreaming tussen deze verschillende onderdelen gebeurt via Dante. Iedere client pc is dan ook verbonden met een Dante AVIO USB adapter.

Het configureren van de correcte audiostreams gebeurt door de API die ter beschikking wordt gesteld door de Dante controller.

De opname gebeurt gecentraliseerd door een aparte server. De server wordt aangestuurd via een WCF API en leest de opnames in via Dante Virtual Soundcard.

Het aansturen van beide API’s (de API van de opnameserver en de API van de Dante Controller) gebeurt door de virtual_iq_server.

Het is ook mogelijk om de opnameserver en de virtual_iq_server op eenzelfde pc uit te voeren. Op dezelfde pc kan zelfs ook een client applicatie uitgevoerd worden. In dat geval wordt er op die specifieke pc geen gebruik gemaakt van een Dante AVIO USB.

De clients zijn initieel pc’s, maar vermits het over een webapplicatie gaat, zou deze ook kunnen uitgevoerd worden op bijvoorbeeld een tablet.

De verschillende onderdelen van de opstelling zullen verder in detail toegelicht worden.

3 INTERPRETER APP

3.1 Functionaliteit Interpreter App

De Interpreter App heeft verschillende knoppen, namelijk:

- FLOOR:
  Door op deze knop te drukken, luistert de tolk naar het FLOOR signaal.
- 6 knoppen, genummerd van 1 t.e.m. 6:
  Door op een van deze knoppen de drukken, krijgt de tolk een andere tolk te horen. Het nummer van de knop stemt hier overeen met het boothnummer van de andere tolk.
- Mic on/off:
  Met deze knop kan de tolk zijn microfoon omschakelen tussen aan en uit. Indien de microfoon uitstaat kunnen andere gebruikers het geluidsignaal niet horen.

Tijdens een “Activity” kan de tolk de knoppen niet besturen. De microfoon staat dan altijd aan en de tolk luistert naar het FLOOR signaal. Na het einde van de Activity worden de oude instellingen terug van toepassing.

De Interpreter App GUI bestond reeds, met uitzondering van de “LOG OUT” knop. De logica om de knoppen te laten werken (server-side) moest echter nog geïmplementeerd worden.

De client applicatie werd ontwikkeld door de co-promotor in Visual Studio Code door middel van Vue. De opzet van deze masterproef was om de achterliggende logica te implementeren.

Het zijn specifiek de rood omcirkelde knoppen die geïmplementeerd werden.

Figuur 3.1: Reeds bestaande GUI voor tolken (Met toegevoegde LOG OUT knop)

4 LOGINSCHERM

4.1 Functionaliteit Loginscherm

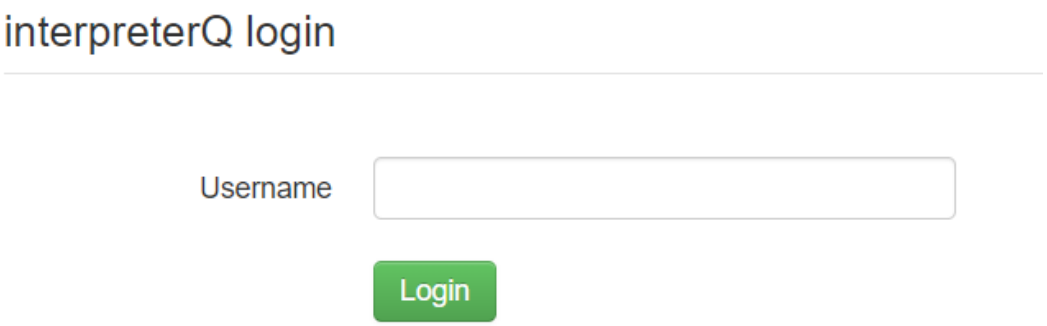
IP adressen die geconfigureerd werden als Interpreter App, krijgen eerst een loginscherm te zien. Het loginscherm heeft één tekstvak waar de student zijn naam kan ingeven. Er is geen veld voor het wachtwoord. Verder is er ook een login knop.

De server verifieert het volgende:

- Niemand anders mag ingelogd zijn met exact dezelfde naam.
- De gekozen naam moet ook een geldige bestandsnaam zijn.

Indien aan beide voorwaarden voldaan werd, wordt de gebruiker doorverwezen naar de Interpreter App, in het andere geval krijgt hij een error pagina te zien.

Het loginscherm is gebaseerd op een reeds bestaand ontwerp met MIT licentie. [1]



Figuur 4.1: Het loginscherm

4.2 Implementatie Loginscherm

Het inloggen en uitloggen gebeurt door middel van een POST request. In de configuratie van de server wordt per IP adres bijgehouden welke accountnaam is ingelogd.

4.2.1 Login

In dat geval wordt een POST request verzonden met als URL path “/login”. Het eigenlijke post bericht bevat de naam. Zie Figuur 4.2.

```
username=ExampleStudentName
```

Figuur 4.2: Voorbeeld van een POST request

4.2.2 Logout

In dat geval wordt een POST request verzonden met als URL path “/logout”. Het bericht zelf is leeg.

5 DELEGATE APP

5.1 Functionaliteit Delegate App

Een delegate is niet in staat om zelf te spreken. Indien er een microfoon aanwezig is, wordt deze niet gebruikt.

Een delegate kan wel vrij luisteren naar al de verschillende tolken en de floor.

5.1.1 Channel selector:

Er zijn twee channel selectors bij een Delegate App, een voor het linkeroor en één voor het rechteroor. De werking van beide channel selectors is identiek. Ze zijn enerzijds opgebouwd uit een dropdown lijst, alsook twee “cycle” knoppen.

5.1.1.1 De dropdown lijst

In de dropdown lijst kan gekozen worden voor het FLOOR signaal, of voor één van de tolken. Bij elke tolk staat er enerzijds het boothnummer en anderzijds de naam van de ingelogde student.

0: floor
1: Naam student1
2: Naam student2

Figuur 5.1: Mogelijke inhoud van een dropdown lijst

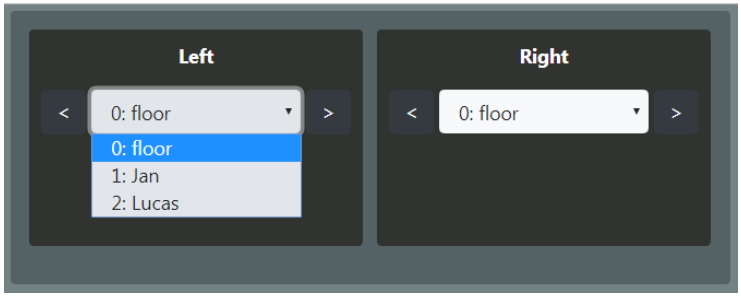
Een nieuwe optie selecteren in de dropdown lijst stuurt automatisch een commando naar de server.

De lijst wordt realtime geüpdatet wanneer een nieuwe student inlogt of uitlogt.

5.1.1.2 Cycle knop

Per dropdown lijst zijn er ook twee cycle knoppen. Eén om het geselecteerde kanaal te incrementeren en een om het geselecteerde kanaal te decrementeren.

Na een debounce (500ms) wordt het signaal naar de server verzonden.



Figuur 5.2: Nieuw geïmplementeerde GUI voor Delegate App

5.2 Implementatie Delegate App

De webpagina werd ontwikkeld in HTML en CSS. De achterliggende functionaliteit werd ontwikkeld in JavaScript. (In de file functions.js)

Voor de ontwikkeling van deze applicatie werd gebruik gemaakt van Visual Studio Code.

Voor de cycle knoppen in de user interface werd gebruik gemaakt van de “**onclick**” property om op deze manier de bijhorende functie aan te roepen.

Voor de dropdown lijst werd gebruik gemaakt van de “**onchange**” property.

Voor de realtime communicatie werd gebruik gemaakt van een WebSocket. Een WebSocket is een bidirectioneel communicatieprotocol, dat werkt bovenop TCP. Het werd gestandaardiseerd in RFC6455.

Er werd een functie “sendMessage(msg)” gedefinieerd. Deze controleert eerst of de WebSocket nog in “connected” state is. Is dat niet het geval, dan wordt eerst een connectie opgezet. Vervolgens wordt het bericht verzonden naar de server.

Het eerste wat zal gebeuren bij het opstarten van de applicatie, is het opvragen van de ingelogde accounts via het “get_accountlist” commando.

Daarnaast zijn er ook berichten die de client kan ontvangen van de server.

JavaScript is “*dynamically typed*”. Daardoor kan een via WebSocket ontvangen JSON bericht, rechtstreeks geparsed worden naar een object, door middel van JSON.parse(msg). Hiervoor hoeft geen klasse gedefinieerd te worden.

Het ontvangen JSON bericht heeft twee properties, namelijk het “MessageType” en de “MessageContent”.

Op basis van het “MessageType”, wordt de juiste functie aangeroepen.

Er is slechts één MessageType dat van toepassing is voor een Delegate App, namelijk een “AccountList”.

5.2.1 Verwerking AccountList

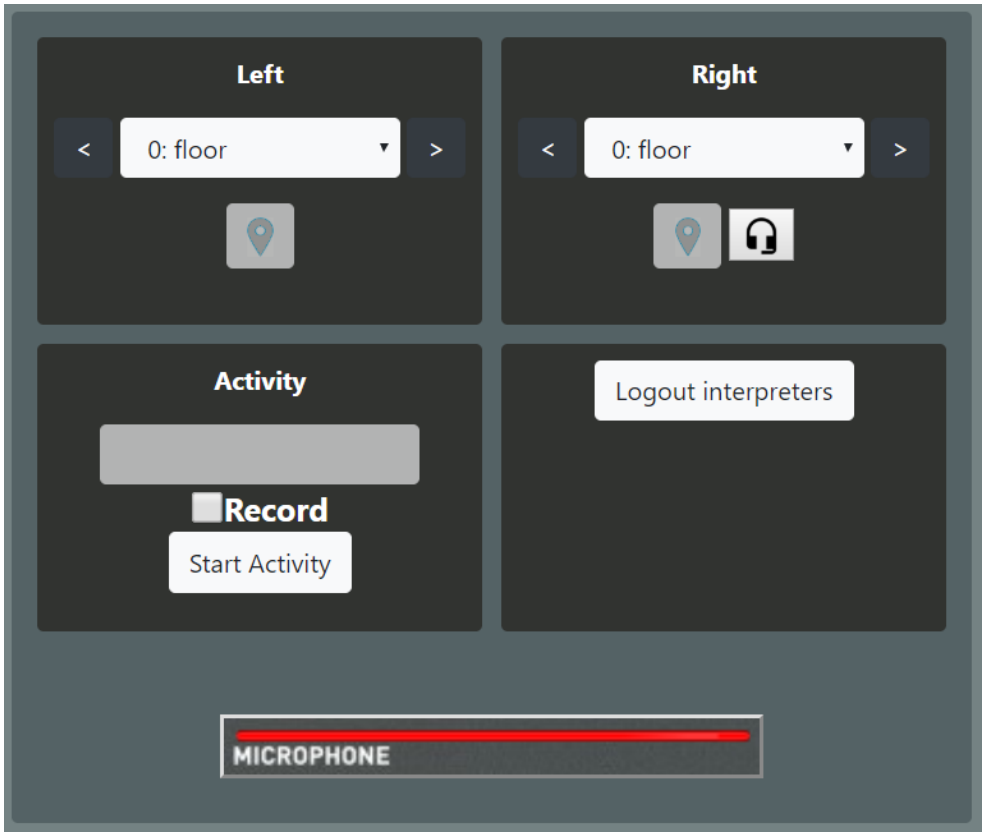
```
{
  "MessageType": "AccountList",
  "MessageContent": [
    "floor",
    "Jan",
    "Lucas"
  ]
}
```

Figuur 5.3: Een AccountList bericht

Indien de MessageType property van het ontvangen bericht een “AccountList” is, bevat de “MessageContent” property een array met de verschillende gebruikersnamen (Zie Figuur 5.3). Beide dropdown lijsten wordt geüpdatet op basis van deze array.

6 CHAIRMAN APP

6.1 Functionaliteit Chairman App



Figuur 6.1: Nieuw geïmplementeerde GUI voor Chairman App

De Chairman App bevat dezelfde functionaliteit als de Delegate App, uitgebreid met nog een aantal extra functionaliteiten:

6.1.1 Activity toggle knop

Een “Activity” is een opgave die door de leerkracht gestart kan worden.

Tijdens een “Activity” gebeuren er volgende zaken:

- Alle Interpreter Apps luisteren naar de FLOOR.
- Alle interpreter microfoons worden aangeschakeld.
- De knoppen van de Interpreter Apps worden uitgezet. (Nog te implementeren)

Verder is er ook een checkbox “Record”. Door deze checkbox aan te klikken, kan de gebruiker in een textbox een naam ingeven voor een opname.

In het geval er reeds een “Activity” bezig is, is de tekst op de knop “Stop Activity”, in het andere geval “Start Activity.”

Bij het aanklikken van “Start Activity” zal, indien de “Record” checkbox werd aangeklikt, ook tegelijk een opname starten met ingegeven naam. Hierbij geldt dat er nog geen opname mag bestaan met dezelfde naam en dat de naam ook een geldige bestandsnaam dient te zijn.

Indien de opname succesvol gestart werd zal hiervan een melding worden weergegeven.

6.1.2 Channel selector

De channel selector van de Chairman App heeft dezelfde functionaliteit als deze van de Delegate App, uitgebreid met de opties “toevoegen markers” en “intercom”.

6.1.2.1 Toevoegen markers

Een marker kan toegevoegd worden op een bepaald tijdstip, bij een bepaalde opname.

Het toevoegen van markers is alleen mogelijk tijdens een gecentraliseerde opname. De lijst van de markers wordt opgeslagen in een aparte metadata file, in dezelfde map als de geluidsbestanden.

Er zijn 2 soorten markers:

- Een globale marker

  Wanneer in de betreffende channel selector een FLOOR signaal is geselecteerd, kleurt het symbool op de knop blauw. In dat geval wordt een globale marker toegevoegd.

  Globale markers zijn nuttig wanneer de leraar de vertaling wil checken van bijvoorbeeld een specifieke terminologie of getal.

- Een student-specifieke marker

  Wanneer in de betreffende channel selector een tolk (student) werd geselecteerd, kleurt het symbool op de knop oranje. In dat geval wordt een student-specifieke marker toegevoegd.

  Dit is nuttig wanneer de leerkracht een fout hoort in bijvoorbeeld uitspraak en hier later op wil terugkomen.

Indien er geen opname bezig is, kan geen marker toegevoegd worden en kleurt de knop grijs.

6.1.2.2 Intercom

Door de intercom functie kan de chairman een gesprek hebben met een individuele student (tolk) zonder dat dit gesprek hoorbaar is voor andere studenten. Dit is nuttig om live feedback te geven.

In de huidige implementatie gebeurt dit gesprek in het rechteroor. De intercom knop is dus ook een toevoeging aan de rechtse channel selector.

6.1.3 Logout interpreters

Door op deze knop te drukken, loggen alle ingelogde tolken uit. Ze krijgen dan automatisch terug het loginscherm te zien. Deze optie is nuttig bij het beëindigen van de les.

6.2 Implementatie Chairman App

Vermits een deel van de functionaliteit overeenkomt met deze van de Delegate App, is een deel van de code ook gemeenschappelijk.

De code voor de channel selectors, de cycle knoppen en de code voor communicatie is gemeenschappelijk met de Delegate App (In de file functions.js).

Daarnaast werd ook een nieuwe file toegevoegd (chairmanfunctions.js) met functies specifiek van toepassing op de Chairman App.

6.2.1 Verwerking inkomende berichten

De reeds bestaande "processMessage" functie werd uitgebreid met een "processMessage_chairman" functie om binnenkomende berichten te verwerken die enkel van toepassing zijn op chairmans.

De bijkomende berichten die van toepassing zijn op chairmans (en niet op delegates) zijn "RecordingStatus" en "ActivityStatusMessage".

6.2.1.1 Verwerking RecordingStatus

```
{
  "MessageType": "RecordingStatus",
  "MessageContent": {
    "StatusType": "Started",
    "MessageContent": "Recording \"RecordigName\" <br> Started at: 3/05/2019 16:59:05"
  }
}
```

Figuur 6.2: Voorbeeld van een RecordingStatus bericht

Bij een “RecordingStatus” zijn er twee mogelijkheden:

- Er is een opname gestart: StatusType =“Started”.
- Er is een opname gestopt: StatusType=”Stopped”.

In het bericht wordt ook nog extra info meegegeven (Zie Figuur 6.2).

In het geval een "RecordingStatus" bericht werd ontvangen, zal dit bericht verwerkt worden door de functie "handleRecordingStatus".

“handleRecordingStatus” roept op zijn beurt twee functies aan:

- “updateMarkerButtons”

  Deze functie zal de kleur van beide marker knoppen aanpassen. De kleur van de marker is afhankelijk van of er al dan niet een opname bezig is, alsook van het geselecteerde kanaal in de channel selector.

- Update van recordingStatusText

  De tekst ingebed in het bericht, zal worden weergegeven op het scherm, onder de “Start Activity” knop.

6.2.1.2 Verwerking ActivityStatusMessage

```
{
  "MessageType": "ActivityStatusMessage",
  "MessageContent": {
    "StatusType": "Started"
  }
}
```

Figuur 6.3: Voorbeeld van een ActivityStatusMessage

In het geval een “ActivityStatusMessage” werd ontvangen, zal de functie “handleActivityStatusMessage” dit bericht verwerken. Het enige wat zal wijzigen is de tekst op de “Start Activity”/”Stop Activity” knop.

7 OPNAMESERVER

7.1 Functionaliteit opnameserver

In de huidige opstelling is er één pc voor elke opname. Een verbetering is om alle geluidsfragmenten afkomstig van de microfoons, te verzamelen op één opnameserver en daar op te slaan.

Enerzijds moeten de microfoons (bijvoorbeeld de Lingua Interpreter desk), in realtime communiceren met de opnameserver. Hiervoor wordt Dante gebruikt.

Anderzijds moet deze opnameserver ook communiceren met de virtual_iq_server. Hiervoor wordt een Windows Communication Foundation (WCF) Application Programming Interface (API) gebruikt.

Hiervoor werd een nieuwe applicatie ontwikkeld in C#, gebruik makende van .NET en Visual studio. De implementatie hiervan wordt verder ook toegelicht.

De opnameserver leest verschillende opnames in via Dante Virtual Soundcard. Verder is er ook een WCF API ter beschikking om opnames te starten/stoppen. In de WCF API kan ook info worden weergegeven over welke kanalen moeten worden opgenomen. (De ClassroomSetup).

7.1.1 Toevoegen Markers

De leraar moet ook de mogelijkheid hebben om zogenaamde “markers” toe te voegen op een bepaald tijdstip. Dit is nuttig indien er een fout werd gemaakt in uitspraak/grammatica/… waar de leraar later wil op terugkomen.

Het is zowel mogelijk om een globale marker toe te voegen (voor alle kanalen), alsook een marker voor een specifiek kanaal.

De vraag stelt zich dan op welke manier deze markers worden opgeslagen. Een eerste mogelijkheid was om deze op te slaan in dezelfde file als de opname. Een tweede mogelijkheid was om een aparte JSON file aan te maken waarin al de markers worden opgeslagen.

In overleg met de co-promotor werd de tweede mogelijkheid gekozen. Enerzijds omdat deze aanpak eenvoudiger te implementeren was, zowel voor het wegschrijven als het inlezen van de markers. Anderzijds omdat deze aanpak werkt voor alle bestandstypen.

Er is één JSON file die alle markers van de verschillende kanalen bevat.

7.1.2 Opslag op schijf

Alle bestanden worden opgeslagen in een map

“C:\Televic Education\RecordingName\AccountName.mp3”

In dezelfde map bevindt zich ook een bestand “metadata.json”. Hierin wordt onder meer de originele ClassroomSetup opgeslagen, alsook een lijst van de markers. Dit wordt verder toegelicht in onderdeel 7.2.3.4.

7.2 Application Programming Interface (API) opnameserver

7.2.1 Mogelijke implementaties API

Zowel de server (=opnameserver) als de client (=virtual_iq_server) worden ontwikkeld in .NET, door middel van Visual studio.

De communicatie moet bidirectioneel zijn, zodanig dat de server ook status updates kan sturen.

7.2.1.1 TCP/IP in combinatie met eigen protocol/JSON

Een eerste mogelijke implementatie, is om een applicatie te schrijven die rechtstreeks TCP segmenten uitwisselt. Binnenin dit segment wordt dan een JSON file meegestuurd met enerzijds de naam van de methode en anderzijds de parameters.

Het voordeel van deze oplossing is dat deze eenvoudig is en lichtgewicht.

Deze oplossing heeft echter als nadeel dat een grote hoeveelheid logica zelf moet geschreven worden, namelijk:

- het opzetten van de TCP Listener;
- het aanroepen van de juiste methoden, door middel van bijvoorbeeld een switch structuur.

7.2.1.2 Web API (REST)

Een tweede mogelijkheid is om een web API te maken.

Deze kan ook voldoen aan de REST principes. Indien wordt gekozen voor een REST API is deze echter wel per definitie stateless, waardoor communicatie van server naar client bemoeilijkt wordt.

7.2.1.3 Windows Communication Foundation (WCF)

Een derde mogelijkheid is om gebruik te maken van Windows Communication Foundation (WCF), een framework voor service-georiënteerde applicaties, dat ter beschikking wordt gesteld in het .NET framework.

De werking is als volgt:

- Er is op de server een serviceklasse, welke een interface implementeert.
- Op de client is een object ter beschikking, dat dezelfde interface implementeert en waarop dezelfde methoden aangeroepen kunnen worden. De communicatie gebeurt daarbij achter de schermen. Dit is ook een van de voordelen van WCF. Het opzetten van een connectie brengt echter wel overhead met zich mee.

In principe moeten zowel de client als server gebruik maken van het .NET framework.

7.2.1.4 Keuze protocol

Uiteindelijk werd gekozen voor Windows Communication Foundation (WCF), wegens volgende redenen:

- Er is een object ter beschikking waarop al de methoden aangeroepen kunnen worden. Dit maakt de implementatie eenvoudiger.
- Vermits zowel de client als server in .NET zullen ontwikkeld worden, is er geen probleem wat ondersteuning betreft.
- Gebruik maken van WCF zorgt voor een grotere overhead, in vergelijking met TCP. Dit is echter niet problematisch. Doordat we gebruik maken van een lokale opstelling is de tijdsvertraging beperkt.

7.2.2 Werking Application Programming Interface (API)

Bij de WCF API is er sprake van bidirectionele communicatie.

De opnameserver stelt namelijk een interface ter beschikking van de client (=de virtual_iq_server). Verder stuurt de opnameserver ook callbacks naar de client.

7.2.2.1 Client ➔ server

```
public interface IRecorderService
{
    [OperationContract(IsOneWay = true)]
    void startRecording(string name);

    [OperationContract(IsOneWay = true)]
    void stopRecording();

    [OperationContract(IsOneWay = true)]
    void AddMarker(string accountName, long timestamp);

    [OperationContract]
    bool IsRecording();

    [OperationContract(IsOneWay = true)]
    void setSetup(string classroomSetupJSON);

    [OperationContract]
    string getSetup();
}
```

Figuur 7.1: De API van de opnameserver

De API biedt volgende functionaliteit aan:

- De client moet een opname kunnen starten met een bepaalde naam, alsook een opname stoppen.
- De client moet een marker kunnen toevoegen bij een bepaalde student, op een bepaald tijdstip. Deze markers zullen worden opgeslagen in een metadata file.
- De client moet kunnen opvragen of er momenteel een opname bezig is.
- De client moet de setup kunnen wijzigen, alsook de huidige setup opvragen. De datastructuren gebruikt voor het weergeven van de setup, worden gespecifieerd in de MetaDataFormat library (zie onderdeel 7.2.3) die gedeeld wordt tussen client en server.

7.2.2.2 Server➔ Client

Het belangrijkste doel van deze API is om alle verbonden clients status updates te geven in verband met de opname.

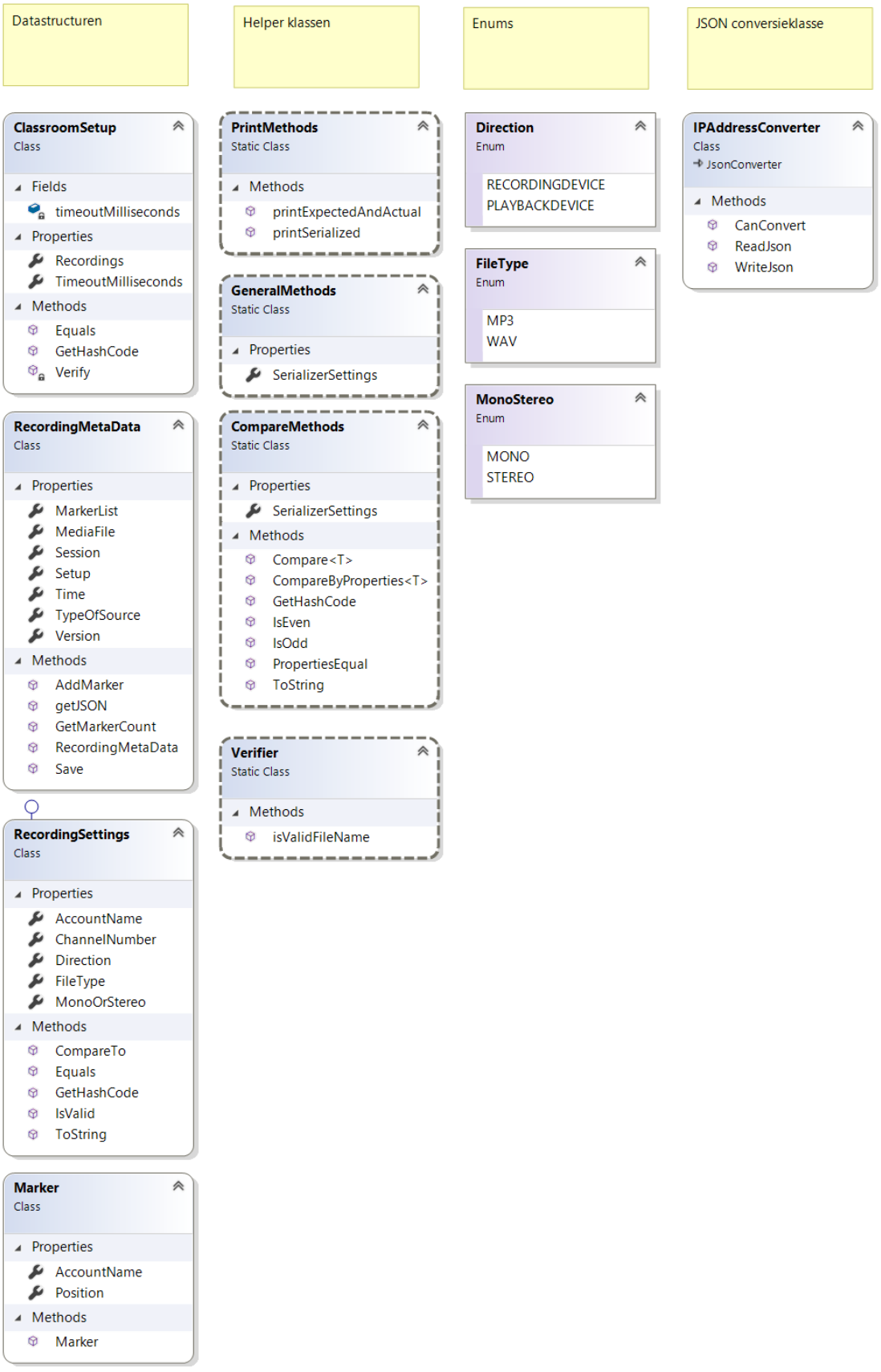
```
public interface IRecorderServiceCallback
{
    [OperationContract(IsOneWay = true)]
    void RecordingStarted(string name, long startTime);
    [OperationContract(IsOneWay = true)]
    void RecordingStopped(string name);
    [OperationContract(IsOneWay = true)]
    void RecordingAvailable(string name);
    [OperationContract(IsOneWay = true)]
    void SetUpChanged(string classroomSetupJSON);
    [OperationContract(IsOneWay = true)]
    void Error(string errorMessage);
}
```

Figuur 7.2: De callback API van de opnameserver

De methoden uit de callback API zijn de volgende:

- RecordingStarted: Een opname is succesvol gestart.
- RecordingStopped: Een opname is gestopt.
- RecordingAvailable: De opnamebestanden werden succesvol opgeslagen op de harde schijf.
- SetUpChanged: Er is een client die de setup gewijzigd heeft.
- Error: Er is een probleem met de opname. Bijvoorbeeld bij het starten van een opname met een reeds bestaande naam.

7.2.3 De MetaDataFormat library



Figuur 7.3: Klassendiagram van MetaDataFormat library

De MetaDataFormat library is een C# library die gedefinieerd werd om de communicatie tussen de opnameserver en de client (=de virtual_iq_server) te bevorderen. De MetaDataFormat library bevat onder meer datastructuren, die nodig zijn bij communicatie met de API.

Verder bevat de library ook Helper klassen. De klassen “PrintMethods” en “CompareMethods” worden gebruikt bij unit tests. (Zie onderdeel 9.3, Klassen ter ondersteuning van Unit Tests, pagina 68).

Vervolgens zijn er ook een aantal klassen die instellingen bevatten voor het converteren naar JSON en terug. (Zie onderdeel 7.2.3.5, De “IPAddressConverter” klasse, pagina 27).

Verder zijn er ook een aantal enums. De werking hiervan wordt verder toegelicht in het gedeelte over de “RecordingSettings” klasse.

7.2.3.1 De “RecordingSettings” klasse

De RecordingSettings klasse bevat info over één specifieke opname, namelijk:

- Direction: Hierbij zijn twee mogelijke waarden:
  - RECORDINGDEVICE: De server neemt een opnameapparaat op (meest gebruikelijk).
  - PLAYBACKDEVICE: In dit geval neemt de server zijn eigen afspeelapparaat op. Dit is vooral nuttig indien de server machine ook wordt gebruikt om een client app op uit te voeren.
- Channelnumber
  - Het kanaalnummer van het betreffende RECORDINGDEVICE of PLAYBACKDEVICE. Dit kan gaan van 1 t.e.m. 16, wegens gebruik van Dante Virtual Soundcard.
- MonoOrStereo
  - Stereo is enkel mogelijk indien “Channelnumber” oneven is. Stel dat Channelnumber=5, dan zullen channel 5 & 6 gecombineerd worden tot 1 stereo opname.
- FileType: Twee mogelijke waarden:
  - WAV
  - MP3
- AccountName
  - Dit is de naam van de student, overeenstemmend met de audio die binnenkomt via dit kanaal. De AccountName zal ook gebruikt worden als naam voor het opnamebestand.

7.2.3.2 De “ClassroomSetup” klasse, met gewenste setup

Bij het aanroepen van de API, moet eerst en vooral een timeout worden ingesteld. Dit is de tijd (gemeten ten opzichte van het starten van de opname) dat het duurt voordat de opname automatisch beëindigd wordt. Deze functionaliteit is nuttig indien de leraar de opname vergeet stop te zetten. (In de huidige versie van de API kan de timeout reeds ingesteld worden, maar de functionaliteit moet nog geïmplementeerd worden)

Vervolgens moet er een lijst worden meegegeven, van verschillende RecordingSettings objecten, met daarin info over de verschillende kanalen.

Eerst maakt de client (=virtual_iq_server) een object aan van “ClassroomSetup”. Dit object wordt daarna omgezet naar JSON, gebruik makende van NewtonSoft. Deze JSON file wordt als parameter meegegeven bij het aanroepen van de API. De server (=de opname pc) zet deze JSON file terug om naar een “ClassroomSetup” object en leest de data hiervan uit. [2]

Het gebruik van de klasse ClassroomSetup op zowel de client als server is mogelijk doordat deze gebruik maken van éénzelfde library, namelijk “MetaDataFormat”.

Figuur 7.4: Geserialiseerde voorstelling van een ClassroomSetup object

7.2.3.3 De “Marker” klasse

De marker klasse stelt een marker voor. Enerzijds is er een “AccountName”. Dit is de naam van de student waarop de marker van toepassing is. (Bij een globale marker is de naam floor).

Verder wordt ook de tijd opgeslagen, gemeten ten opzichte van het starten van de opname, met als eenheid “ticks”. (10.000 ticks=1ms)

7.2.3.4 De “RecordingMetaData” klasse

Het is deze klasse die zal gebruikt worden om via NetwonSoft te serialiseren naar metadata.json. De klasse heeft volgende properties:

- **Version**

Er wordt enerzijds een “Version” bijgehouden. In de huidige implementatie is dit het getal “0”. Bij toekomstige updates kan dit getal verhoogd worden. Aan de hand van dit versienummer kan de uitlezer dan weten op welke manier het bestand geformatteerd werd. Op deze manier wordt backwards compatiblity eenvoudiger te implementeren.

- **Time**

Dit is het moment waarop de opname gestart werd.

- **TypeOfSource**

In de huidige implementatie is deze waarde altijd “Centralised recording Setup”.

- **Session**

Hierin wordt de naam van de opname opgeslagen, die werd meegegeven door de chairman.

- **MediaFile**

In de huidige implementatie wordt dit veld nog niet gebruikt. In toekomstige implementaties kan hier een verwijzing komen naar bijvoorbeeld een webcam opname.

- **MarkerList**

Dit is een lijst van verschillende Marker objecten, voor alle opnames, bijgehouden in chronologische volgorde.

- **Setup**

De ClassroomSetup die werd meegegeven bij het aanroepen van de API.

```
{
  "Version": 0,
  "Time": "2019-03-22T10:55:58.5727576+01:00",
  "TypeOfSource": "Centralised recording setup",
  "Session": "DemoDay",
  "MediaFile": null,
  "MarkerList": [
    {
      "Position": 121227499,
      "AccountName": "floor"
    },
    {
      "Position": 180896260,
      "AccountName": "Jan"
    },
    {
      "Position": 232653944,
      "AccountName": "Lucas"
    }
  ],
  "Setup": {
    "TimeoutMilliseconds": 86400000,
    "Recordings": [
      {
        "Direction": "RECORDINGDEVICE",
        "ChannelNumber": 3,
        "MonoOrStereo": "MONO",
        "FileType": "MP3",
        "AccountName": "Jan"
      },
      {
        "Direction": "RECORDINGDEVICE",
        "ChannelNumber": 4,
        "MonoOrStereo": "MONO",
        "FileType": "MP3",
        "AccountName": "Lucas"
      },
      {
        "Direction": "PLAYBACKDEVICE",
        "ChannelNumber": 1,
        "MonoOrStereo": "STEREO",
        "FileType": "MP3",
        "AccountName": "floor"
      }
    ]
  }
}
```


Figuur 7.5: Voorbeeld van een metadata.json bestand

7.2.3.5 De “IPAddressConverter” klasse

NewtonSoft is standaard niet in staat om IP adressen te converteren naar JSON.

Het is nodig om een klasse te maken die erft van de abstracte klasse “JsonConverter”. JsonConverter bepaalt de manier waarop het IPadress geserialiseerd en gedeserialiseerd wordt.

Hiervoor werd gebruikt gemaakt van een reeds bestaande klasse: IPAddressConverter [3]

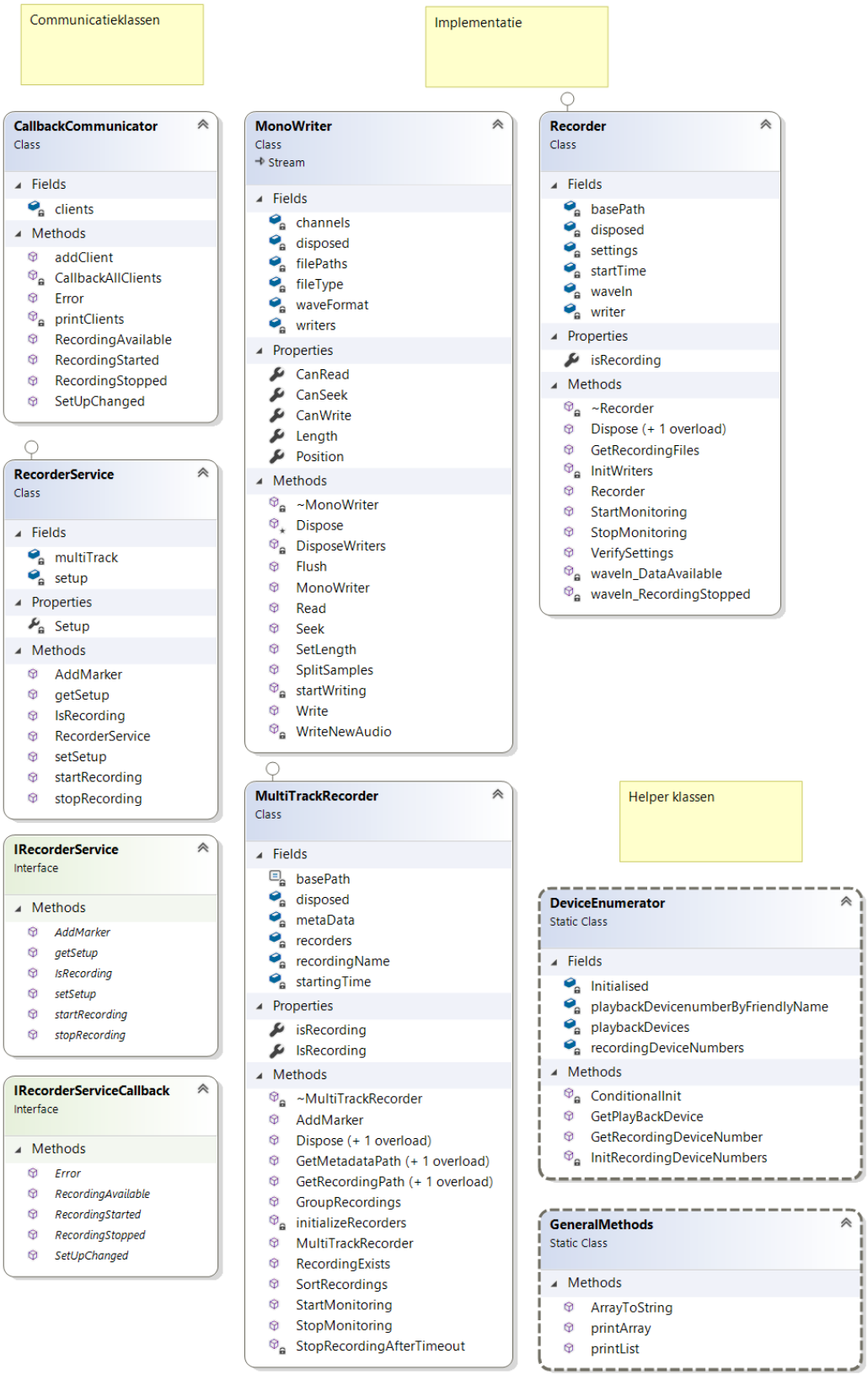
Om JsonConverter te implementeren moeten drie methoden overschreven worden, te zien in Figuur 7.6.

```
public override bool CanConvert(Type objectType)
public override void WriteJson(JsonWriter writer, object value,
JsonSerializer serializer)
public override object ReadJson(JsonReader reader, Type objectType,
object existingValue, JsonSerializer serializer)
```

Figuur 7.6: Te overschrijven methoden van JsonConverter

7.3 Implementatie van opnameserver

7.3.1 Klassendiagram “RecorderService”



Figuur 7.7: Klassendiagram van de opnameserver

De taak van de opnameserver bestaat erin om audiostreams binnen te lezen en vervolgens op te slaan in de juiste audio files.

In wat volgt zal de werking van de opnameserver als geheel uitgelegd worden, zonder gedetailleerd in te gaan op de werking van elke klasse afzonderlijk.

7.3.2 Lezen van data

Zoals al eerder vermeld, komen de audiostreams binnen op de pc door middel van Dante Virtual Soundcard. Het lijkt alsof de pc een aantal extra microfoons en luidsprekers heeft. De Virtual Soundcard groepeert de 16 inputkanalen en 16 outputkanalen tot 8 stereo geluidskaarten. Elke geluidskaart heeft dan 2 inputkanalen (=virtueel opnameapparaat) en 2 outputkanalen (=virtueel afspeelapparaat).

Om deze audiostreams in te lezen werd gebruik gemaakt van een C# programma.

De opnameserver is zowel in staat om opnameapparaten alsook op afspeelapparaten op te nemen. Voor het inlezen van beide audiostreams werd gebruik gemaakt van de NAudio bibliotheek.

Om de opnameapparaten op te nemen werd gebruik gemaakt van de NAudio klasse “WaveInEvent”. Om de afspeelapparaten op te nemen werd gebruik gemaakt van de NAudio klasse “WasapiLoopbackCapture”.

Beide klassen hebben gemeenschappelijk dat ze de NAudio interface “IWaveIn” implementeren.

7.3.2.1 IWaveIn interface

IWaveIn heeft events voor wanneer:

- Nieuwe data aanwezig is (DataAvailable).
- De opname gestopt is (RecordingStopped).

[4]

De nieuwe data komt binnen als samples in functie van de tijd. De data komt binnen in stereo. De samples zijn dus afwisselend die van het linkse kanaal en rechtse kanaal.

Er zijn 2 kanalen, de samplefrequentie (voor elk kanaal) is 48000Hz en er worden 16 bits gebruikt om een sample voor te stellen.

Er werd een eventhandler toegevoegd (waveIn_DataAvailable). Deze zal de data wegschrijven naar de harde schijf. De manier waarop dit wegschrijven gebeurt wordt in de volgende paragraaf toegelicht.

7.3.2.2 Opnemen van opnameapparaat

Deze bibliotheek geeft de mogelijkheid om zowel een “WaveIn” als WaveInEvent” object aan te maken. Deze klassen geven toegang tot de low-level audio input devices op Windows.

WaveIn biedt echter enkel ondersteuning voor GUI applicaties en niet voor console applicaties of WCF services. Dit omdat WaveIn gebruik maakt van de Windows message loop, die enkel beschikbaar is bij GUI applicaties

Vandaar dat werd gekozen om de streams binnen te lezen door middel van een “WaveInEvent” object, die ook ondersteuning biedt voor console applicaties & WCF services.

7.3.2.3 Opnemen van afspeelapparaat

Voor het opnemen van het afspeelapparaat werd gebruik gemaakt van de klasse WasapiLoopbackCapture.

Ook nuttig om te weten is dat WasapiLoopbackCapture enkel samples afgeeft wanneer er ook een programma is dat audio afspeelt op deze geluidskaart. Indien dat niet het geval is, worden er geen samples afgegeven door het DataAvailable event. Er worden dus ook geen nulwaarden afgegeven.

Een mogelijke toekomstige toevoeging is om een programma toe te voegen dat altijd samples met een waarde van 0 afspeelt naar de geluidskaart.

7.3.3 Schrijven van Data

Om data weg te schrijven kan gekozen worden tussen verschillende klassen. Deze klassen hebben allemaal gemeenschappelijk dat ze de “Stream” interface implementeren. [5]

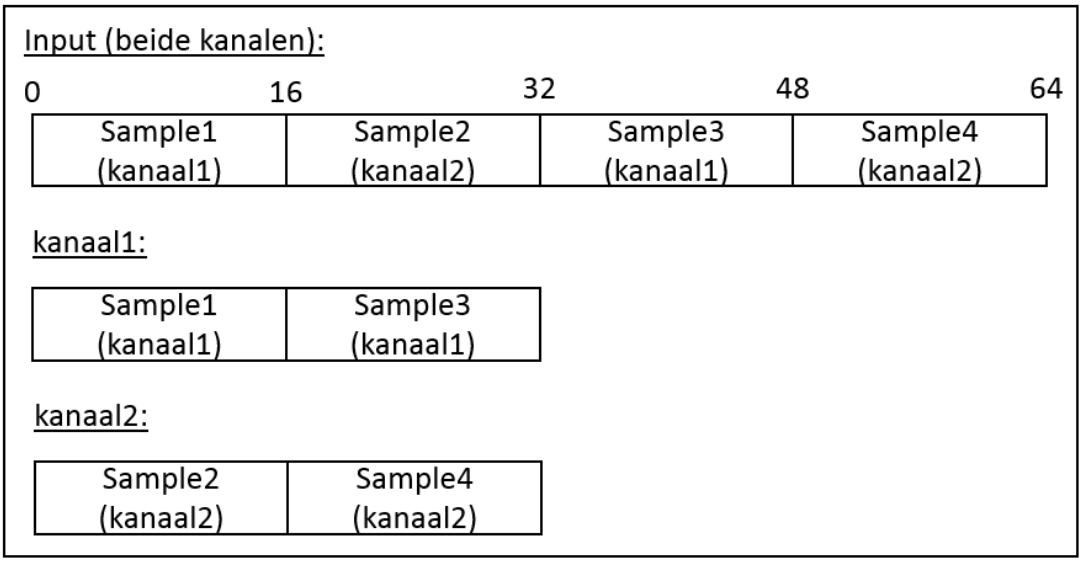
De bibliotheek “NAudio.Lame” biedt volgende klassen aan:

- LameMP3FileWriter (Voor mp3 files)
- WaveFileWriter (Voor wav files)

[6]

Voor stereo opnames volstaat het om, in het geval er nieuwe data aanwezig is, de nieuwe samples rechtstreeks weg te schrijven naar één van bovenstaande streams door middel van de “write” methode.

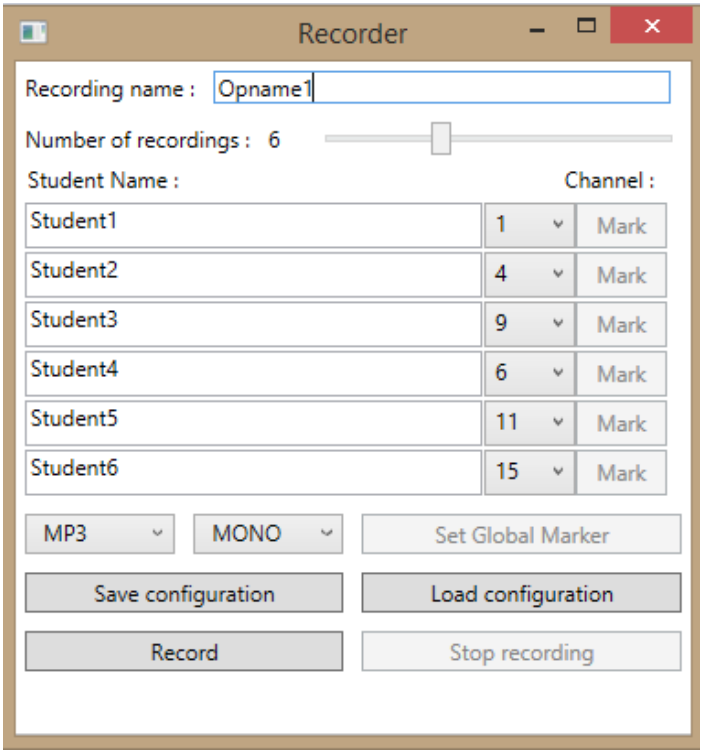
In de opstelling nemen de microfoons echter in mono op, wat wil zeggen dat deze terug gesplitst dienen te worden. Daarom werd de MonoWriter klasse gedefinieerd, die ook de Stream klasse implementeert. Hierop kan ook de write methode aangeroepen worden. Achter de schermen splitst deze klasse eerst de stereo samples op en maakt vervolgens twee nieuwe mono streams aan om de waarden naar de harde schijf weg te schrijven.



Figuur 7.8: Het splitsen van de samples (1 sample=16 bits)

8 DE GRAFISCHE GEBRUIKERSINTERFACE (GUI) VOOR HET DEBUGGEN

8.1 Functionaliteit grafische gebruikersinterface



Figuur 8.1: Grafische gebruikersinterface pc van de leraar (wanneer de opname niet bezig is)

Eerst werd gestart met de ontwikkeling van de opnameserver. Er werd een API ontwikkeld, maar er was nog geen software om deze API ook aan te spreken. Vandaar werd ook een grafische gebruikersinterface (GUI) ontwikkeld (zie Figuur 8.1:), om tijdens het ontwikkelproces te gebruiken.

Vanaf het moment dat de virtual_iq_server ontwikkeld werd, was de grafische gebruikersinterface (GUI) niet meer nodig. De definitieve API van de opnameserver is ook verschillend van degene die gebruikt werd tijdens de ontwikkeling van de GUI en is in de huidige versie dus ook niet meer aanspreekbaar door de GUI.

Deze voorlopige GUI gaf eerst en vooral de mogelijkheid om een naam voor de opname te kiezen. Vervolgens werd het aantal opnames dat zal plaatsvinden bepaald. De GUI herschaalde zich automatisch op basis van dit aantal. Vervolgens werd de naam van de studenten ingegeven en hun bijhorende kanalen.

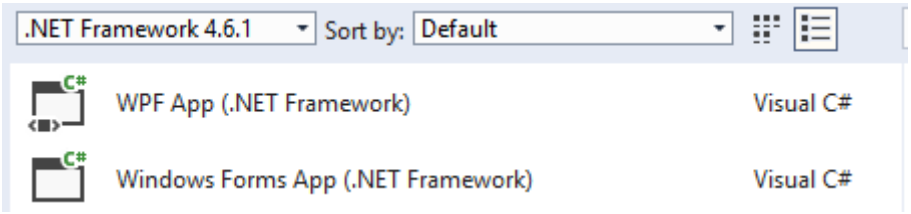
Het was ook mogelijk om te kiezen voor het bestandstype (MP3 of WAV) en voor een MONO of STEREO opname.

Door op de “Save configuration” knop te drukken, werd de opstelling van de klas opgeslagen in JSON formaat. Een eerder opgeslagen configuratie kon uiteraard ook terug geladen worden.

8.2 Implementatie grafische gebruikersinterface

8.2.1 Keuze GUI framework

Voor het maken van een GUI in het .NET framework zijn er twee opties, namelijk een WPF App en een Windows Forms App.



Figuur 8.2: Types gebruikersinterface in dotNET. Schermafbeelding uit Visual Studio

8.2.1.1 Windows Presentation foundation (WPF) App

WPF is op vlak van werking vergelijkbaar met HTML. In een XAML file kan een hiërarchische structuur van elementen gedefinieerd worden. Het is ook mogelijk om tijdens de uitvoer van het programma elementen toe te voegen, waardoor de gebruikersinterface zich automatisch zal herschalen.

8.2.1.2 Windows Forms App

Windows Forms App is een eerder statische vorm van GUI. In Visual studio krijgen componenten een vaste plaats door deze te verslepen (drag and drop). Visual studio genereert automatisch de juiste code.

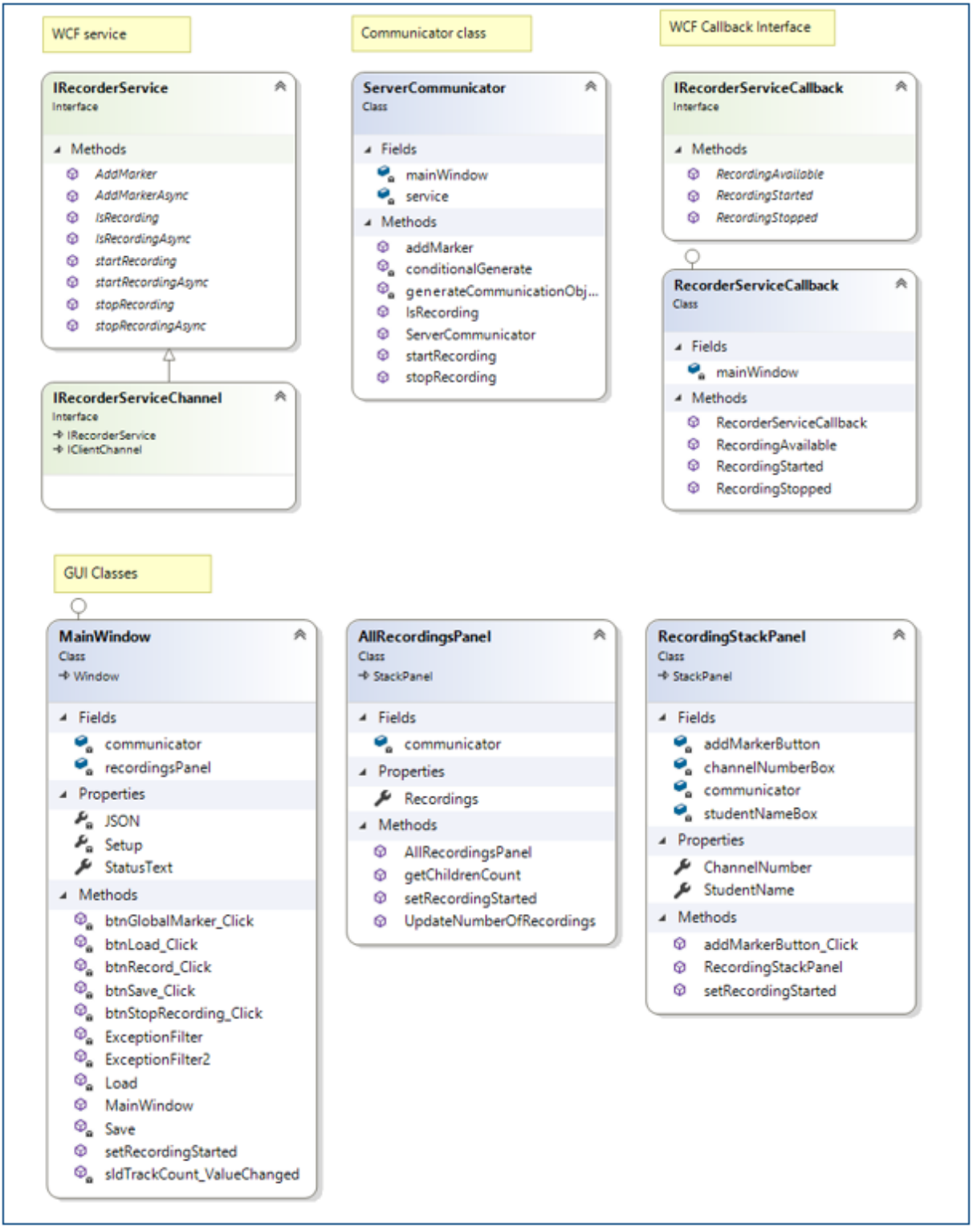
Voor het maken van de teacher applicatie werd gekozen voor WPF (Windows Presentation Foundation), om de volgende redenen:

- De GUI is eenvoudig te wijzigen tijdens uitvoer van programma, deze wordt automatisch herschaald.
- De uitlijning van de elementen kan gemakkelijker consistent gemaakt worden.

Het grootste deel van de GUI werd opgesteld door gebruik te maken van een XAML file.

Vermits het aantal opnames zelf te kiezen is en dus variabel, wordt het overzicht van de opnames tijdens de uitvoer van het programma (runtime) toegevoegd.

8.2.2 Klassendiagram



Figuur 8.3: Het klassendiagram van de GUI

8.2.2.1 De klasse “RecordingStackPanel”.

Elke rij komt overeen met één opname. Hiertoe werd een nieuwe klasse gedefinieerd genaamd RecordingStackPanel, welke een subklasse is van StackPanel.

De elementen zijn de volgende:

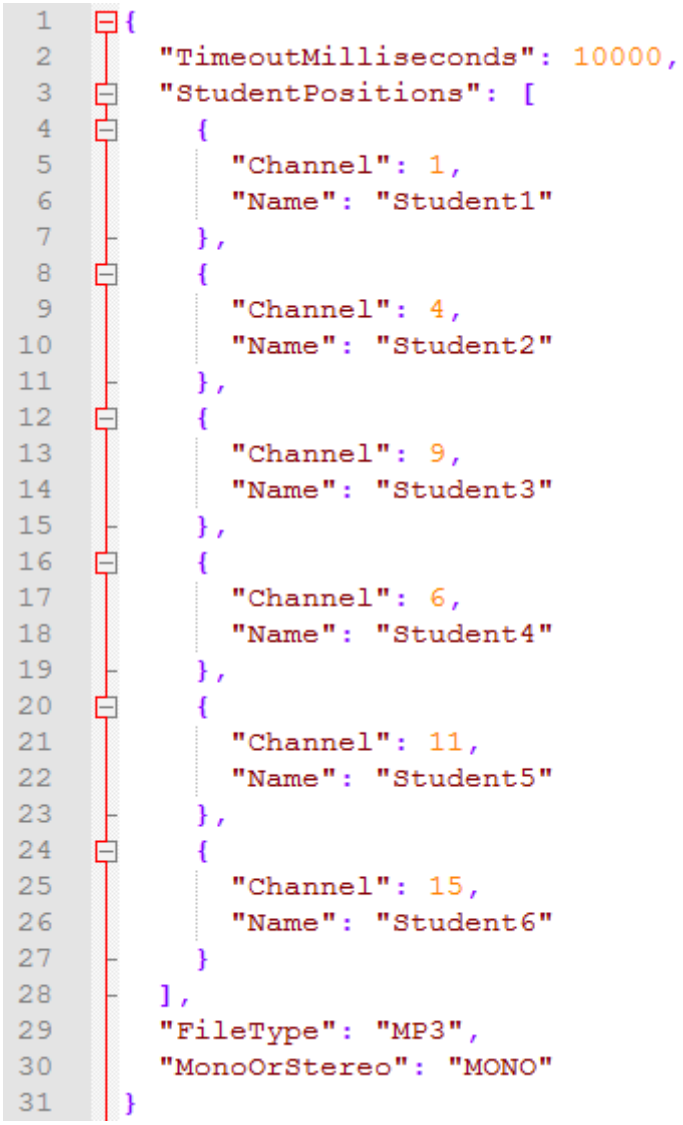
- Een TextBox, om de naam van de student te kiezen.
- Een ComboBox waarbij het nummer van het bijhorende kanaal kan geselecteerd worden (gaande van 0 t.e.m. 15)
- Een Button, enkel actief tijdens de opname, waarmee een marker kan toegevoegd worden op het bijhorende kanaal.

Uiteraard biedt RecordingStackPanel ook de mogelijkheid om bovenstaande waarden op te vragen door middel van Properties.

Doordat RecordingStackPanel een subklasse is van StackPanel, kan deze tijdens de uitvoer van het programma toegevoegd/verwijderd worden als Child van AllRecordingsPanel.

[7]

8.2.2.2 De klasse “ClassroomSetup” (eerste versie).



```
{
  "TimeoutMilliseconds": 10000,
  "StudentPositions": [
    {
      "Channel": 1,
      "Name": "Student1"
    },
    {
      "Channel": 4,
      "Name": "Student2"
    },
    {
      "Channel": 9,
      "Name": "Student3"
    },
    {
      "Channel": 6,
      "Name": "Student4"
    },
    {
      "Channel": 11,
      "Name": "Student5"
    },
    {
      "Channel": 15,
      "Name": "Student6"
    }
  ],
  "FileType": "MP3",
  "MonoOrStereo": "MONO"
}
```

Figuur 8.4: Een voorbeeld van een JSON file (gebruikt in de oude versie van de API), met de configuratie van de klas

Indien de gebruiker op de “Record” knop drukt, zal eerst gecontroleerd worden of de ingegeven waarden correct zijn.

De controle op incorrecte waarden gebeurt bij het aanmaken van een “ClassroomSetup” object. Het aanmaken van zo’n object met ongeldige waarden zorgt voor een Exceptie.

(Dit is ArgumentOutOfRangeException of FormatException, afhankelijk van wat de fout is.)

Het bericht van deze exceptie wordt opgevangen en weergegeven op de GUI.

Indien de waarde wel correct is probeert de client de server te contacteren.

9 VIRTUAL_IQ_SERVER

9.1 Functionaliteit Virtual_iq_server

Binnen elke opstelling is er exact één “virtual_iq_server.” De belangrijkste functionaliteiten hiervan zijn:

- bijhouden van configuratie;
- fungeren als webserver
    - Login screen
    - Interpreter App
    - Delegate App
    - Chairman App;
- aansturen van Dante Controller;
- aansturen van de Recording API.

9.1.1 Bijhouden van configuratie

De configuratie bestaat uit een aantal verschillende “entries”. Iedere entry komt overeen met één pc in het subnet. Elke entry bestaat voor een deel uit statische configuratie en een deel dynamische configuratie.

De statische configuratie beschrijft de opstelling zelf.

De dynamische configuratie houdt bij wie er op dat moment aangemeld is en de instelling op dat moment.

Het is de bedoeling dat de statische configuratie persistent wordt bijgehouden (in een JSON file) en dat de dynamische configuratie enkel in RAM wordt bijgehouden. (Opmerking: In de huidige implementatie bevat de JSON file echter ook nog dynamische configuratie.)

9.1.2 Fungeren als webserver

TCP poort 2000 van de virtual_iq_server fungeert als webserver. Eventueel kan ook voor een andere poort geopteerd worden, zoals poort 80.

De verschillende pc’s kunnen met deze webserver verbinden. Het IP adres van de pc wordt dan opgezocht in de configuratie:

- Indien het IP adres niet aanwezig is in de configuratie, wordt de gebruiker doorverwezen naar een foutpagina, met instructies om de configuratie aan te passen.
- Indien het IP adres wel aanwezig is in de configuratie, wordt het bijhorende type opgezocht in de configuratie. Afhankelijk van dit type wordt de juiste webpagina toegestuurd:
    - Indien het IP adres ingesteld werd als delegate, wordt de Delegate App weergegeven.
    - Indien het IP adres ingesteld werd als chairman, wordt de Chairman App weergegeven.
    - Indien het IP adres ingesteld werd als interpreter, wordt eerst een loginscherm weergegeven.

9.1.3 Aansturen van Dante Controller

Er wordt gebruik gemaakt van de API die de Dante controller ter beschikking stelt, om zo de correcte streams op te zetten.

Een AdapterChannelCombination is een kanaal van een bepaalde Dante Adapter.

Het opzetten van een stream gebeurt tussen twee verschillende AdapterChannelCombinations.

Via een inputkanaal kan slechts naar één outputkanaal geluisterd worden. Eenzelfde outputkanaal kan echter wel door meerderen tegelijk beluisterd worden.

De audiostreams hebben 4 verschillende functionaliteiten, namelijk (in volgorde van prioriteit):

1) RecordingSubscriptions (audiostreams naar de opnameserver)
2) IntercomSubscriptions
3) ActivitySubscriptions (audiostreams naar het FLOOR signaal tijdens een klassikale opgave)
4) RelaySubscriptions (vrij te kiezen audiostreams tussen de verschillende deelnemers)

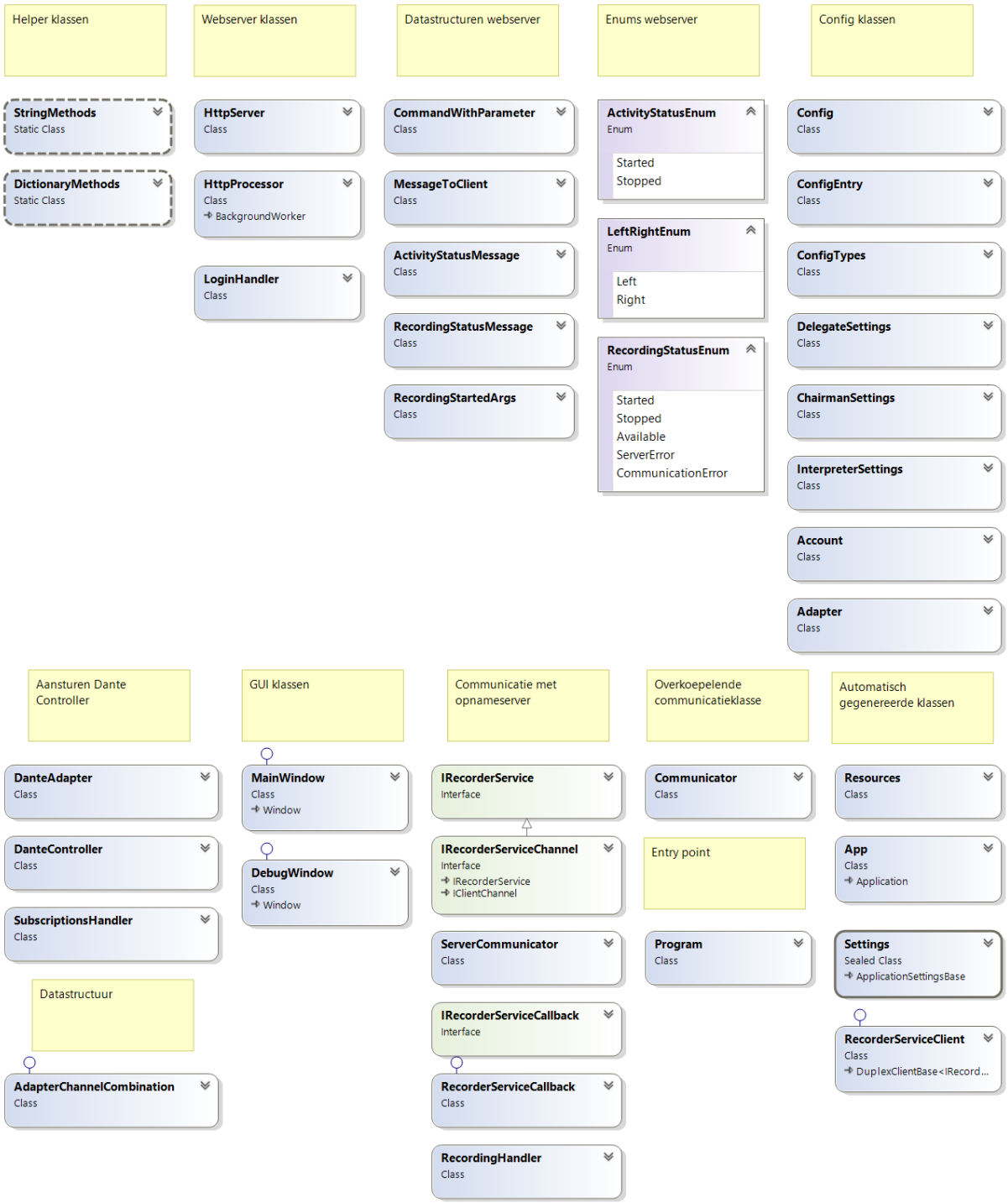
Het is mogelijk dat er conflicten zijn tussen deze verschillende soorten streams, in het geval eenzelfde inputkanaal meerdere tegenstrijdige mappings heeft. Dit is bijvoorbeeld het geval wanneer een student via zijn inputkanaal naar een andere student wil luisteren, maar er op dat moment ook een “Activity” bezig is. In dat geval krijgt de “ActivitySubscription” voorrang.

9.1.4 Aansturen van de recording API

Nadat de subscriptions naar de Virtual Soundcard van de opnameserver gelegd zijn, moet ook de API van de opnameserver aangeroepen worden om de opname te starten. Dit zal pas gebeuren na het bevel van een Chairman App. De respons van de callback API wordt ook teruggestuurd naar de Chairman App.

9.2 Implementatie Virtual_iq_server

9.2.1 Klassendiagram



Figuur 9.1: Klassendiagram van virtual_iq_server

In Figuur 9.1 is een overzicht te zien van de klassen van de virtual_iq_server.

Om het overzichtelijk te houden werden enkel de namen van de klassen opgenomen. De functionaliteit van de verschillende klassen zal in wat volgt verder toegelicht worden.

De klassen HttpProcessor, HttpServer, DanteController, DanteAdapter, Communicator en MainWindow bestonden reeds. De klassen DanteAdapter en MainWindow werden in de huidige implementatie niet gebruikt. Tijdens de masterproef werden de andere bestaande klassen nog verder uitgewerkt. De rest van de klassen uit het klassendiagram werden volledig ontwikkeld tijdens deze masterproef. (Met uitzondering van de automatische gegenereerde klassen).

9.2.2 Werking Dictionary

Een Dictionary is een veel gebruikte datastructuur in de virtual_iq_server. Zo worden onder meer Dictionaries gebruikt om de gewenste subscriptions bij te houden. Daarom is het belangrijk om eerst uit te leggen wat Dictionaries zijn.

9.2.2.1 Functionaliteit Dictionary

Een C# Dictionary is vergelijkbaar met een “Map” uit Java. Een Dictionary is een datastructuur die mappings bijhoudt tussen een “Key” en een “Value”. Op basis van de Key, kan de gebruiker de bijhorende Value opvragen.

Specifiek in het programma wordt gebruik gemaakt van de Dictionary te zien in Figuur 9.2.

```
Dictionary<AdapterChannelCombination, AdapterChannelCombination>
```

Figuur 9.2: De gebruikte dictionary voor het bijhouden van subscriptions

De Key stelt de subscriber voor en de Value is het kanaal waarop geabonneerd wordt.

Een bepaalde Key kan slechts eenmaal voorkomen in een Dictionary. Dit is geen probleem vermits een subscriber slechts op één kanaal geabonneerd kan zijn.

9.2.2.2 Implementatie en gebruik van een Dictionary

Een Dictionary datastructuur bestaat uit een aantal KeyValuePairs.

Om een object te kunnen gebruiken als Key in een Dictionary, moeten er twee methoden overschreven worden (zie Figuur 9.3).

```
public virtual int GetHashCode ();
public virtual bool Equals (object obj);
```

Figuur 9.3: De GetHashCode en Equals methode van de "Object" klasse

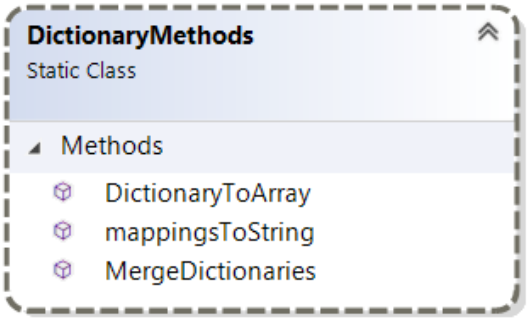
Wanneer twee objecten gelijk zijn, moet de Equals methode “true” teruggeven. Voor beide objecten moet GetHashCode() ook hetzelfde resultaat teruggeven.

Wanneer twee objecten verschillend zijn, moet de Equals methode “false” teruggeven. Om de efficiëntie van Dictionaries te bevorderen, is het wenselijk dat de hashcode van deze objecten ook verschillend is. Dit is echter niet altijd mogelijk om te implementeren, vermits er vaak een oneindig aantal object states zijn, terwijl er slechts een eindig aantal hashcodes zijn. In dat geval is het mogelijk dat twee verschillende objecten toch een gelijke hash hebben.

In een Dictionary worden de verschillende KeyValuePairs namelijk gesorteerd op basis van de hashcode van de Key. Het kan voorkomen dat er meerdere Keys zijn die verschillend zijn (volgens de Equals methode), maar die toch een gelijke hashcode hebben. In dat geval worden deze KeyValuePairs bijgehouden in een aparte datastructuur, genaamd een “bucket”.

9.2.3 Helper klassen

9.2.3.1 DictionaryMethods



Figuur 9.4: De Dictionary klasse van de virtual_iq_server

De statische helper klasse DictionaryMethods bevat verschillende methoden met betrekking tot Dictionaries. De DictionaryToArray methode en MergeDictionaries methode, worden verder toegelicht.

- **DictionaryToArray**

```
public static string[] DictionaryToArray(Dictionary<int, string> input)
```

Figuur 9.5: De DictionaryToArray methode

In deze methode wordt een Dictionary geconverteerd naar array. De “int” Key stelt hier de index voor die het object in de array zal krijgen. De hoogste index van de array komt overeen met de hoogste Key uit de Dictionary. Vermits de index van een array vanaf 0 start, is de lengte van de array is dus de hoogste Key, plus één. Key numbers die werden overgeslagen in de Dictionary, krijgen een “null” waarde in de array.

Het spreekt voor zich dat de Keys uit de Dictionary geen negatieve waarden mogen bevatten. In dat geval wordt een ArgumentException geworpen.

- **“MergeDictionaries”: samenvoegen van Dictionaries**

In de klasse “DictionaryMethods” werd de methode MergeDictionaries gedefinieerd (zie Figuur 9.6).

```
public static Dictionary<AdapterChannelCombination,AdapterChannelCombination>
MergeDictionaries(List<Dictionary<AdapterChannelCombination,
AdapterChannelCombination>> dictionaries);
```

Figuur 9.6: De DictionaryMethods.MergeDictionaries methode

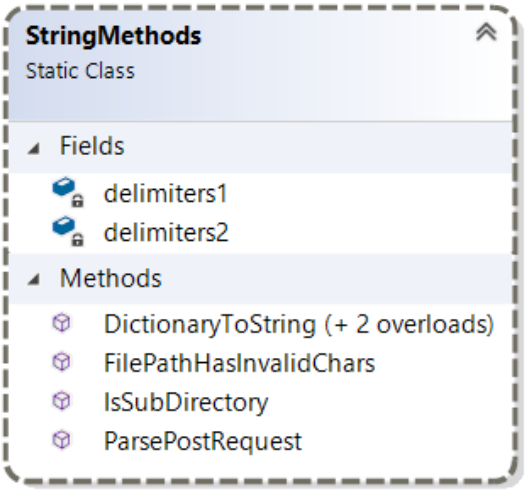
Deze methode combineert een lijst van verschillende Dictionaries tot één Dictionary.

Wanneer een bepaalde Key in meerdere Dictionaries voorkomt, krijgt de Dictionary die het eerst voorkomt in de lijst prioriteit.

Er wordt over alle Dictionaires in de lijst geïtereerd en het samenvoegen gebeurt door middel van Lambda Expressions.

Deze methode wordt gebruikt bij het samenvoegen van de verschillende subscriptions volgens prioriteit.

9.2.3.2 StringMethods



Figuur 9.7: De StringMethods klasse

In de StringMethods klasse zijn er meerdere methoden die betrekking hebben op strings.

```
public static Dictionary<string, string> ParsePostRequest(string request);
public static bool FilePathHasInvalidChars(string path);
public static bool IsSubDirectory(string subdirectory, string parent);
```

Figuur 9.8: Enkele method headers van de StringMethods klasse

- **DictionaryToString**

De methode DictionaryToString wordt in de huidige implementatie niet meer gebruikt.

- **FilePathHasInvalidChars**

Deze methode geeft een false indien er karakters aanwezig zijn die niet mogen voorkomen in een pad naar een bestand. Hiervoor werd gebruikt gemaakt van een reeds bestaande methode. [8]

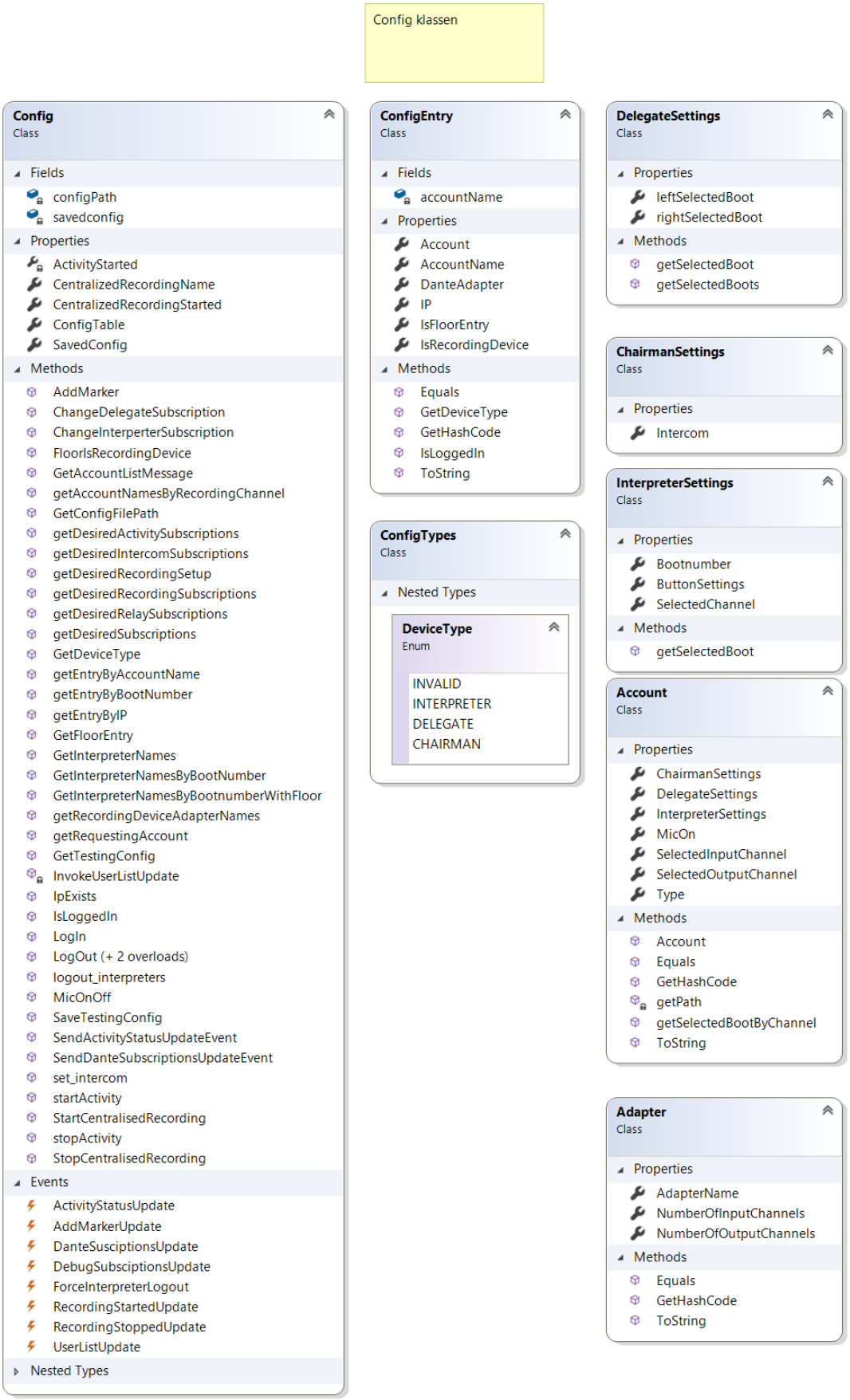
- **IsSubDirectory**

Deze methode test of een bepaalde map een subdirectory is van een andere map. Ook hiervoor werd gebruik gemaakt van een reeds bestaand codevoorbeeld. [9]

- **ParsePostRequest**

In deze methode zal een HTTP post request ontvangen van een form, omgezet worden naar een Dictionary<string,string>. Meer uitleg hierover in onderdeel 9.2.7.3, Parsen van POST request body, pagina 54.

9.2.4 Bijhouden van configuratie



Figuur 9.9: De klassen voor het bijhouden van de configuratie

9.2.4.1 De Config Entries

Een “ConfigEntry” komt overeen met één pc in het netwerk. Een ConfigEntry bestaat voor een deel uit statische configuratie en een deel uit dynamische configuratie.
Voor het bijhouden van de Config entries zijn er twee klassen verantwoordelijk:

- De "ConfigEntry" klasse,
- De "Account" klasse die ook voorkomt als property van "ConfigEntry".

Tabel 9-1: De statische configuratie

Type	Naam	Beschrijving
IPAddress	IP	Het (statisch) IP adres van de pc.
bool	IsRecordingDevice	“true” indien er een opnameserver wordt uitgevoerd op deze pc.
bool	IsFloorEntry	“true” indien deze pc het floor signaal levert.
Adapter	DanteAdapter	De naam van de bijhorende Dante Adapter van de pc.
DeviceType (Enum)	Type	Het type (Interpreter, Chairman, Delegate).

Tabel 9-2: De dynamische configuratie

Type	Naam	Beschrijving
bool	MicOn	Deze waarde houdt bij of de microfoon al dan niet aanstaat.
int	SelectedOutputChannel	Het geselecteerde outputkanaal. Dit is vooral nuttig voor chairmans. Zij kunnen kiezen tussen volgende outputkanalen: • Het geluid van de microfoon • Het pc outputgeluid • Een audiospeler met aparte geluidskaart
InterpreterSettings	InterpreterSettings	Bevat volgende info: • Het boothnummer van deze Interpreter App • Het gekozen kanaal (FLOOR, of kanaal 1 t.e.m. 6) • De naam van de ingelogde account.
DelegateSettings	DelegateSettings	Het gekozen kanaal voor zowel het linkeroor als het rechteroor
ChairmanSettings	ChairmanSettings	Uitbreiding van de DelegateSettings: • Er wordt bijgehouden of er momenteel een intercom actief is.

9.2.4.2 De Config Klasse

De Config klasse houdt een lijst bij van alle ConfigEntries.

Er zijn methoden om de configuratie te wijzigen. Telkens de configuratie gewijzigd wordt, worden er events gegenereerd waarop gesubscribed kan worden.

- **Opvragen van Entries**

Het opvragen van de entries gebeurt door middel van Language-Integrated Query (LINQ). Enerzijds kan een entry opgevraagd worden op basis van een IP adres.

```
public ConfigEntry getEntryByIP(IPAddress IP)
{
    var query =
        from entry in ConfigTable
        where entry.IP.Equals(IP)
        select entry;
    List<ConfigEntry> list = query.ToList();
    if (list.Any())
    {
        return list[0];
    } else
    {
        return null;
    }
}
```

Figuur 9.10: Opvragen van entry aan de hand van IP, door middel van LINQ

Anderzijds kan ook een entry opgevraagd worden overeenstemmend met een bepaald boothnummer. De implementatie hiervan is gelijkaardig als het opvragen van een entry aan de hand van IP adres.

LINQ biedt ook alternatieve en kortere syntax aan.

```
return ConfigTable.Where(x => x.IP == IP).FirstOrDefault();
```

Figuur 9.11: Kortere LINQ syntax

- **Genereren van Events**

De Config klasse houdt enerzijds een lijst bij van alle Config entries. Verder worden ook events gegenereerd indien de configuratie gewijzigd werd.

Er zijn meerdere soorten events die gegenereerd kunnen worden door de Config klasse. Een overzicht wordt weergegeven in Tabel 9-3.

Er zijn in totaal 3 klassen die werden ontwikkeld voor de afhandeling van deze events, namelijk:

- LoginHandler: Verantwoordelijk voor communicatie met de webapplicaties.
- SubscriptionsHandler: Verantwoordelijk voor aansturen Dante Controller API.
- RecordingHandler: Verantwoordelijk voor communicatie met de opnameserver.

Verder is ook het Debug Window. Dit window geeft een overzicht van alle audiosubscriptions volgens type (zie onderdeel 9.2.8.4, Het Debug Window, pagina 63).

Tabel 9-3: Overzicht van de gegenereerde events in de Config klasse van de virtual_iq_server

Naam event	Voorwaarde	Parameters	Gevolg
UserListUpdate (afgehandeld door LoginHandler)	Een tolk logt in of uit.	Een “MessageToClient”. Deze bevat een array met de namen van alle ingelogde tolken.	De lijst zal via WebSocket worden doorgestuurd naar alle verbonden clients. De Chairman Apps en Delegate Apps zullen hun dropdown lijst updaten.
DanteSubscriptionsUpdate (afgehandeld door SubscriptionsHandler)	De audiosubscriptions moeten gewijzigd worden, bijvoorbeeld wanneer iemand kiest om naar een ander audiosignaal te luisteren.	Een Dictionary met alle gewenste audiosubscriptions (desiredSubscriptions).	De gewenste subscriptions zullen vergeleken worden met reeds aanwezige subscriptions (currentSubscriptions). Aan de hand van de verschillen zal de API aangeroepen worden.
DebugSubscriptionsUpdate (weergegeven in Debug Window)	idem	Gelijkaardig aan DanteSusciptionsUpdate, maar de subscriptions worden opgesplitst volgens type. Dit onderscheid is niet nodig voor de werking, maar wel nuttig tijdens het debuggen.	De subscriptions worden weergegeven in een grafische gebruikersinterface (GUI) op de virtual_iq_server.
RecordingStartedUpdate (afgehandeld door RecordingHandler)	Een chairman start een Activity met bijhorende recording.	De naam van de recording, alsook de “ClassroomSetup”. Deze klasse “ClassroomSetup” bevat een lijst van alle gewenste opnames.	De API van de opnameserver zal tweemaal worden aangeroepen: eenmaal om de setup door te geven en eenmaal om de opname te starten.
RecordingStoppedUpdate (afgehandeld door RecordingHandler)	Een chairman zet de Activity stop.	Geen parameters.	De API van de opnameserver zal worden aangeroepen om de opname te stoppen.

Tabel 9-3: Overzicht van de gegenereerde events in de Config klasse van de virtual_iq_server (vervolg)

Naam event	Voorwaarde	Parameters	Gevolg
AddMarkerUpdate (afgehandeld door RecordingHandler)	Een chairman voegt een marker toe tijdens een opname	De naam van de student waarbij de marker wordt toegevoegd.	De API van de opnameserver zal worden aangeroepen om de marker toe te voegen.
ActivityStatusUpdate (afgehandeld door LoginHandler)	Een Activity werd gestart of gestopt door een chairman	Een “ActivityStatusMessage”.	De ActivityStatus zal via WebSocket worden doorgegeven aan al de verbonden clients. De chairman apps zullen al dan niet hun knop togglen.
ForceInterpreterLogout (afgehandeld door LoginHandler)	Een chairman klikt op “Logout Interpreters”.	Geen parameters	Er zal een via WebSocket een “ForceInterpreterLogout” bericht gestuurd worden naar de clients. Alle tolken krijgen terug het loginscherm te zien.

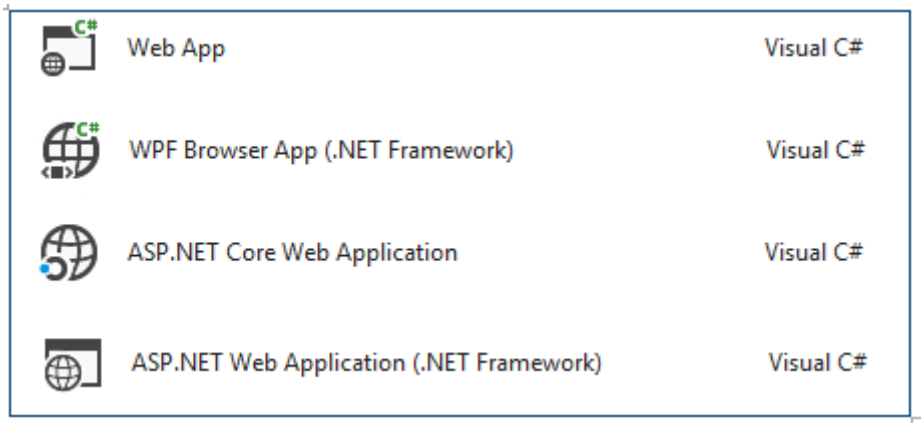
9.2.5 Keuze webserver

9.2.5.1 Vereisten webserver

1) De webserver moet de gebruikers doorverwijzen naar de correcte webpagina afhankelijk van hun geconfigureerd type (Interpreter/Delegate/Chairman).
2) Er dient een login procedure te zijn voor tolken.
3) De webserver moet in staat zijn om statische bestanden (.html) aan te bieden.

Om coherentie te bewaren met de andere software die reeds ontwikkeld werd, wordt ook hier gebruik gemaakt van C#, .NET en Visual studio.

Binnen .NET worden verschillende soorten webservers ondersteund. Deze zullen nu verder bestudeerd worden. Daarnaast bestaat er ook de "*TCC HttpServer*", ontwikkeld door Televic Education.



Figuur 9.12: Webservers (in Visual Studio)

9.2.5.2 ASP.NET

ASP.NET is een framework voor de ontwikkeling van webapplicaties en services.

Het is gebaseerd op .NET en JavaScript.

- **ASP .NET Core**

  Bij een .NET Core Web Application, is er standaard een map aanwezig "wwwroot". Al de documenten in deze map kunnen opgevraagd worden door middel van GET request.

  Daarnaast bied ASP.NET nog een aantal extra functionaliteiten aan. Twee daarvan worden hier verder toegelicht:

  - **API**

    Aan de hand van de URL kunnen gegevens opgevraagd worden. Het antwoord is bijvoorbeeld een JSON file, maar de implementatie kan zelf gekozen worden.

    Er is ondersteuning voor GET, POST, PUT & DELETE.

  - **Web Application**

    Er is een map "Pages" aanwezig waarin .cshtml files voorkomen. Razor genereert aan de hand hiervan de uiteindelijke HTML.

- **ASP .NET Web application (.NET framework)**

Dit is vergelijkbaar met een ASP.NET Core Web Application, maar biedt ook ondersteuning voor het .NET framework. Dit voegt extra functionaliteit toe, maar betekent ook dat de applicatie niet door alle besturingssystemen zal ondersteund worden.

Het .NET framework stelt een klassenbibliotheek ter beschikking. Deze klassen bieden onder meer ondersteuning voor geheugenbeheer, security, netwerkbeheer, …

Belangrijk is (voor onze toepassing) de ondersteuning voor WCF.

Bovendien zijn er API's ter beschikking voor communicatie met het Windows operating systeem.

[10] [11]

9.2.5.3 TCC HttpServer

Binnen Televic Education werd een webserver ontwikkeld, namelijk de "*TCC HttpServer*".

In deze webserver wordt gebruik gemaakt van de *"TcpListener"* klasse van het .NET framework om de requests te ontvangen.

Voor het afhandelen van de requests werd een nieuwe interne klasse gedefinieerd, namelijk de HttpProcessor. Deze erft van BackgroundWorker.

9.2.5.4 Conclusie

Er werd gekozen voor de *TCC HttpServer.*

Het belangrijkste voordeel van de TCC HttpServer was de flexibiliteit van deze oplossing. De broncode van deze server was ter beschikking en deze kon dus ook aangepast worden aan de noden van deze specifieke applicatie.

9.2.6 Werking HTTP

In wat volgt wordt kort de werking van Hypertext Transfer Protocol (HTTP) toegelicht, dit is belangrijk om de werking van de server te begrijpen.

HTTP is een protocol voor communicatie tussen clients en servers.

Het is request-response gebaseerd. De client stuurt een request en de server een response. In dit geval is de webbrowser de client.

9.2.6.1 Request

Binnen HTTP zijn er meerdere soorten requests, namelijk:

GET, POST, PUT, HEAD, DELETE, PATCH, OPTIONS

Voor deze toepassing is enkel GET en POST van belang.

- GET

  Een GET request is bedoeld om een specifieke resource op te vragen. De query wordt verzonden in de URL. Bijvoorbeeld:

  /loginscreen/index.html

Een GET request mag niet gebruikt worden om data te wijzigen. Verder wordt een GET request ook opgeslagen in de browsergeschiedenis. Een GET request mag daarom geen gevoelige data bevatten.

- POST

  Een POST request is bedoeld om data te verzenden naar een server. Naast de URL is er ook een “body”, die het bericht zelf bevat.

Vermits Login en Logout stateful zijn en de status van de server wijzigen, wordt daarvoor gebruik gemaakt van POST. Voor al de andere requests wordt gebruik gemaakt van een GET request.

9.2.6.2 Response

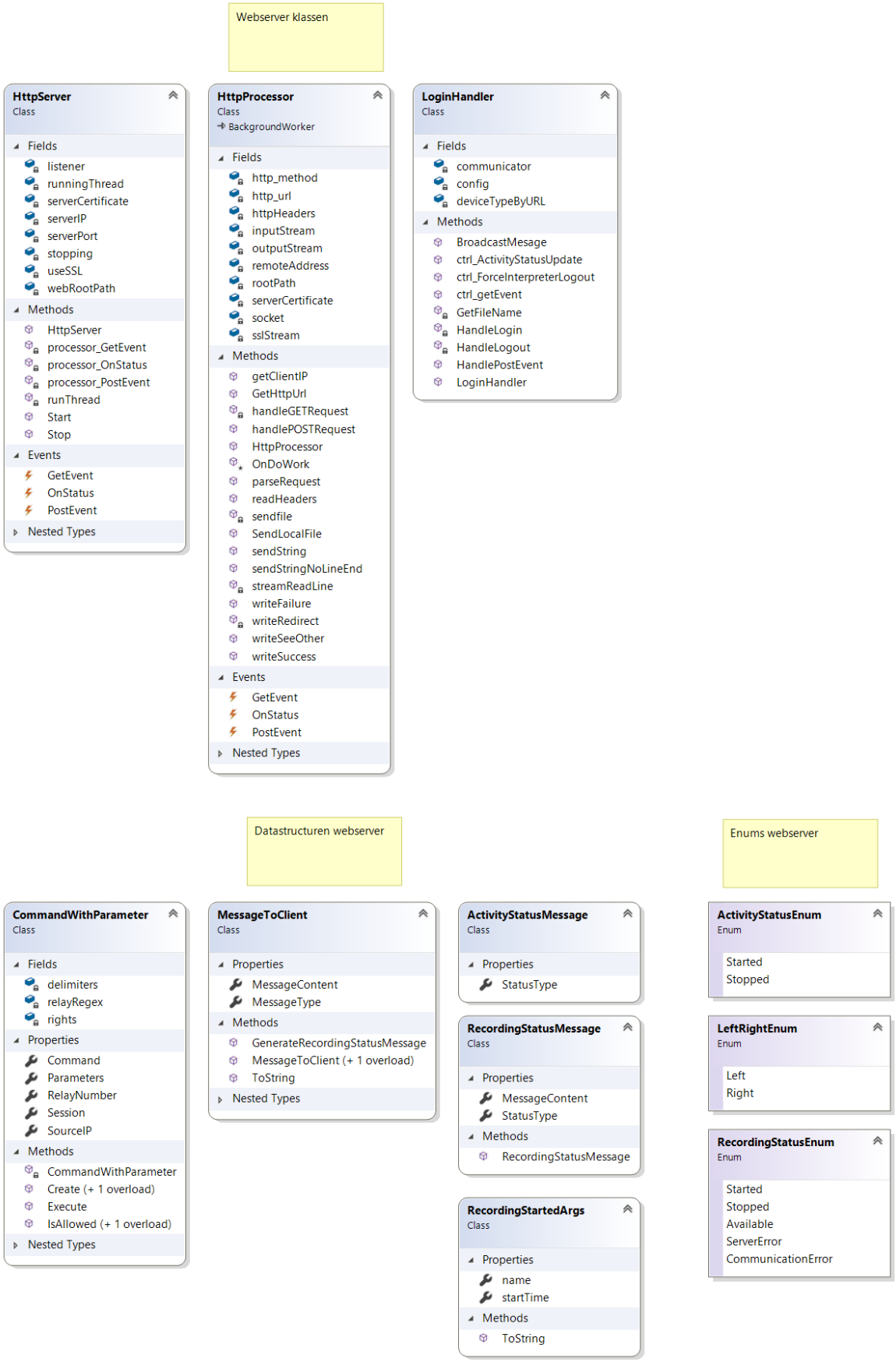
Elk antwoord dat gestuurd wordt heeft een driecijferige statuscode. De antwoorden zijn gegroepeerd volgens vijf verschillende klassen:

- 1xx: Information responses
- 2xx: Successful responses
- 3xx: Redirection messages
- 4xx: Client error responses
- 5xx: Server error responses

Voor onze toepassing wordt er gebruik gemaakt van 3 responscodes, namelijk:

- **200 OK**: De request is geslaagd;
- **303 See Other**: Doorverwijzing naar een andere URI;
- **404 Not found:** De opgevraagde resource bestaat niet.

9.2.6.3 Klassendiagram webserver

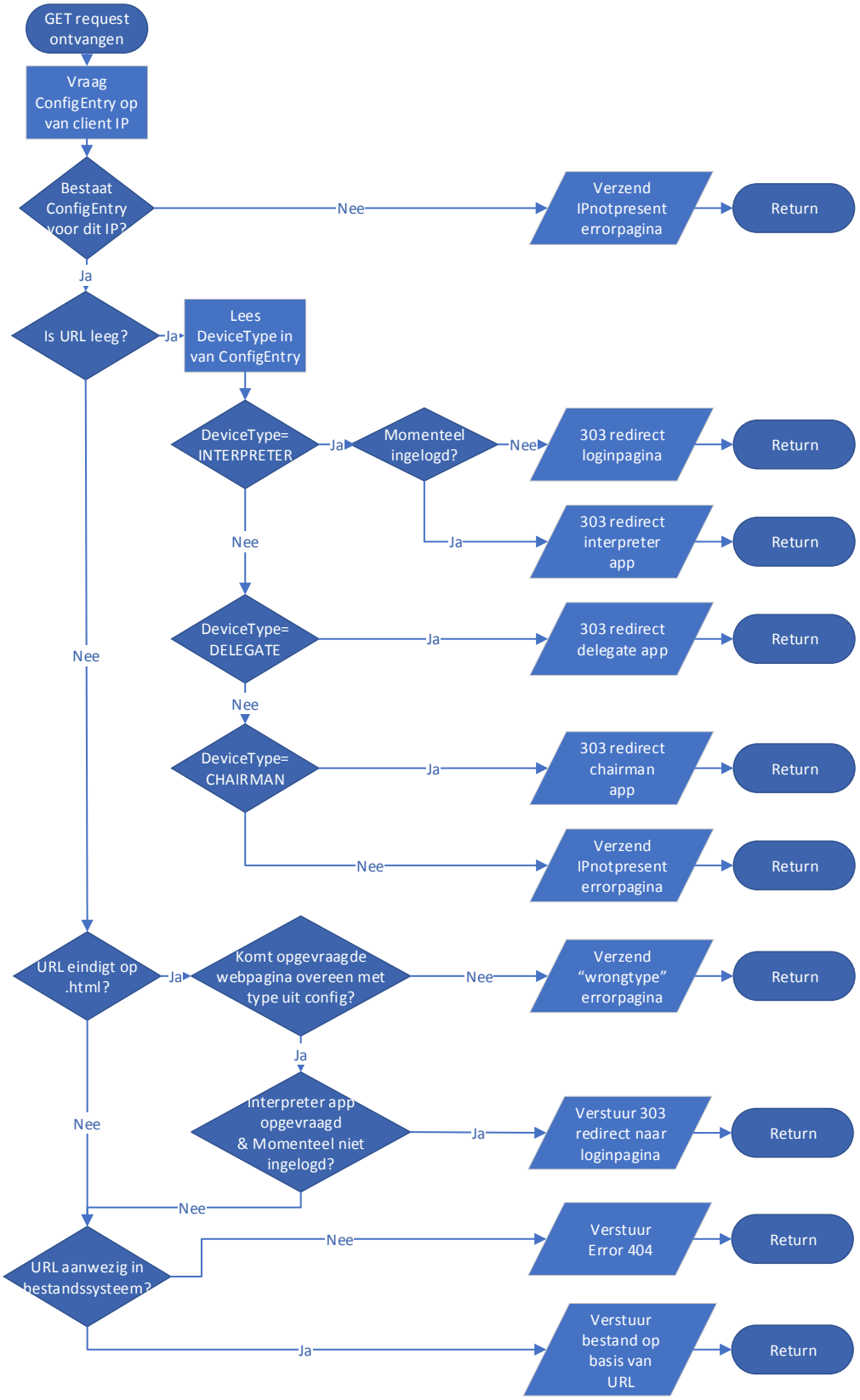


Figuur 9.13: Klassendiagram van het webserver gedeelte van de virtual_iq_server

9.2.7 Implementatie webserver

In de klasse “LoginHandler worden onder meer GET en POST requests afgehandeld.

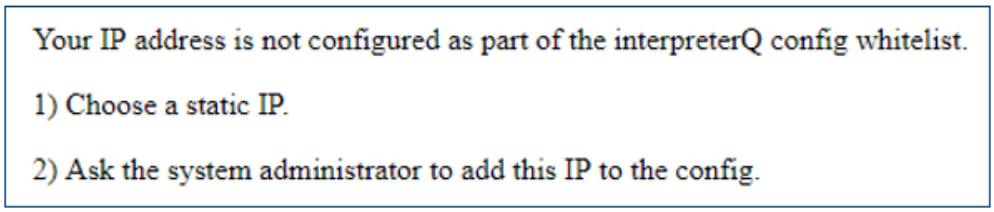
9.2.7.1 Afhandeling GET request



Figuur 9.14: Flowchart afhandeling van een GET request

Hierbij volgt wat toelichting bij de flowchart.
Zoals reeds eerder vermeld, komt iedere ConfigEntry overeen met een bepaald IP adres van een pc uit de opstelling.

Het eerste dat gebeurt is het opvragen van de ConfigEntry van de client die de request stuurt. Indien deze ConfigEntry niet aanwezig is, wordt een errorpagina gestuurd met instructies om het IP adres aan de configuratie toe te voegen (Zie Figuur 9.15:).

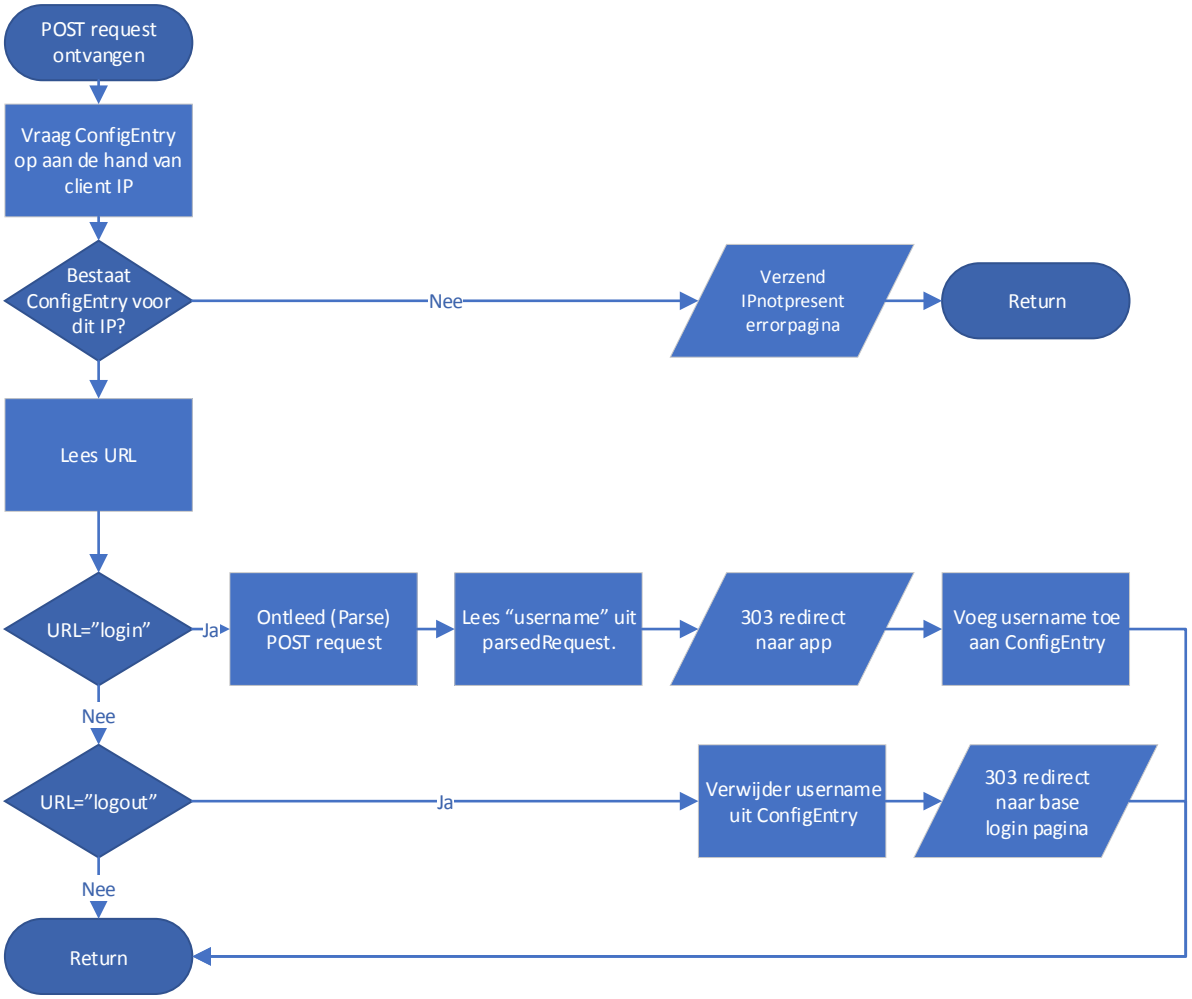


Figuur 9.15: De IPnotpresent errorpagina

Indien het IP adres wel aanwezig is, zal eerst de URL uitgelezen worden.

- Indien de URL leeg is, wordt via een 303 (See Other) statuscode doorverwezen naar de juiste webapplicatie afhankelijk van het DeviceType. Op deze manier hoeft de gebruiker geen URL in te geven, wat de opstelling gebruiksvriendelijker maakt.
- Indien de URL eindigt op .html, wordt eerst geverifieerd of de gebruiker het recht heeft om deze webpagina op te vragen. Zo mag een Interpreter niet de webpagina opvragen voor bijvoorbeeld een Delegate App. In dat laatste geval zal een error pagina worden weergegeven. Indien een webpagina van een Interpreter app wordt opgevraagd, wordt ook altijd geverifieerd of de tolk op dat moment ingelogd is. Is dat niet het geval, dan volgt een doorverwijzing naar de loginpagina.
- In het geval de URL wel aanwezig is, wordt een bestand van het lokale bestandssysteem verzonden. De bestanden bevinden zich in de map wwwroot.

9.2.7.2 Afhandeling POST request



Figuur 9.16: Flowchart afhandeling POST request

Een POST request heeft enerzijds een URL en anderzijds een “body” die het eigenlijke bericht bevat.

De manier waarop een afwezig IP adres wordt afgehandeld gebeurt op dezelfde manier als bij een GET request.

Indien het IP adres wel aanwezig is, gebeurt afhankelijk van de URL het volgende:

- URL=”login”. In dat geval wordt eerst het POST request ontleed, vervolgens wordt de “username” hieruit uitgelezen. Dan volgt de doorsturing naar het juiste type app afhankelijk van het DeviceType. In de praktijk is dit altijd de Interpreter App, vermits in de huidige implementatie enkel interpreters dienen in te loggen. De username zal ook worden toegevoegd aan de ConfigEntry.
- URL=”logout”. In dat geval wordt de username verwijderd naar “/”, wat overeenkomt met de loginpagina.

Indien de URL van het POST request niet gekend is, wordt er geen respons gestuurd, ook geen 404. In normale omstandigheden zal dit echter niet voorkomen, aangezien de gebruiker niet zelf manueel de POST request samenstelt.

9.2.7.3 Parsen van POST request body

Een POST request verzonden door een HTML form heeft een vaststaande structuur, namelijk een lijst van KeyValuePairs.

De verschillende KeyValuePairs worden onderling van elkaar gescheiden door een ampersand (&). Het onderscheid tussen Key en Value wordt gemaakt door middel van een gelijkheidsteken (=).

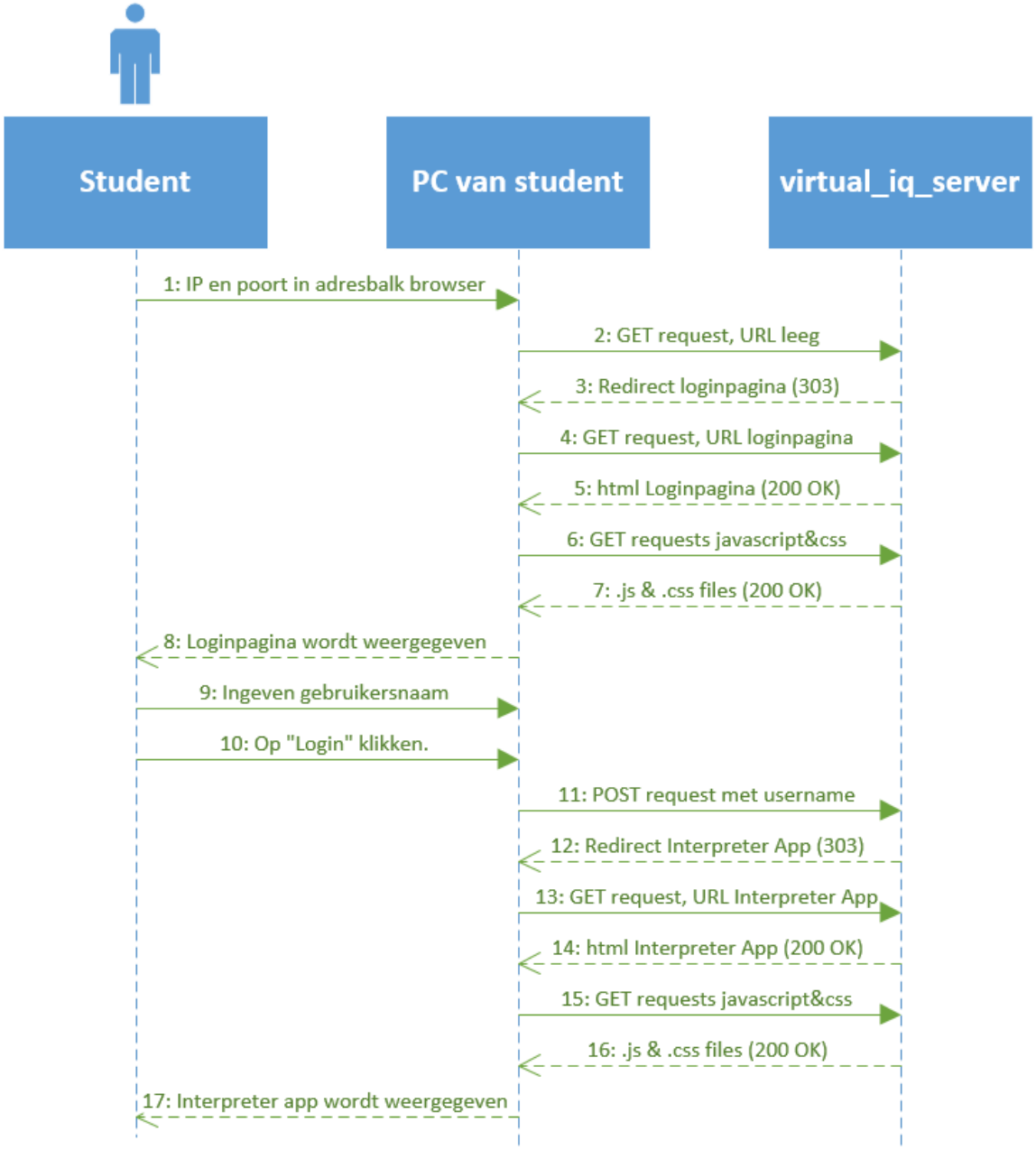
Voorbeelden:

- *key1=value1&key2=value2*
- *username=name*

Er werd een methode “ParsePostRequest” gedefinieerd. Deze methode zet het body van een POST request om naar een Dictionary<string,string>.

Uit deze Dictionary wordt dan de username opgevraagd.

9.2.7.4 Voorbeeld: login van een tolk

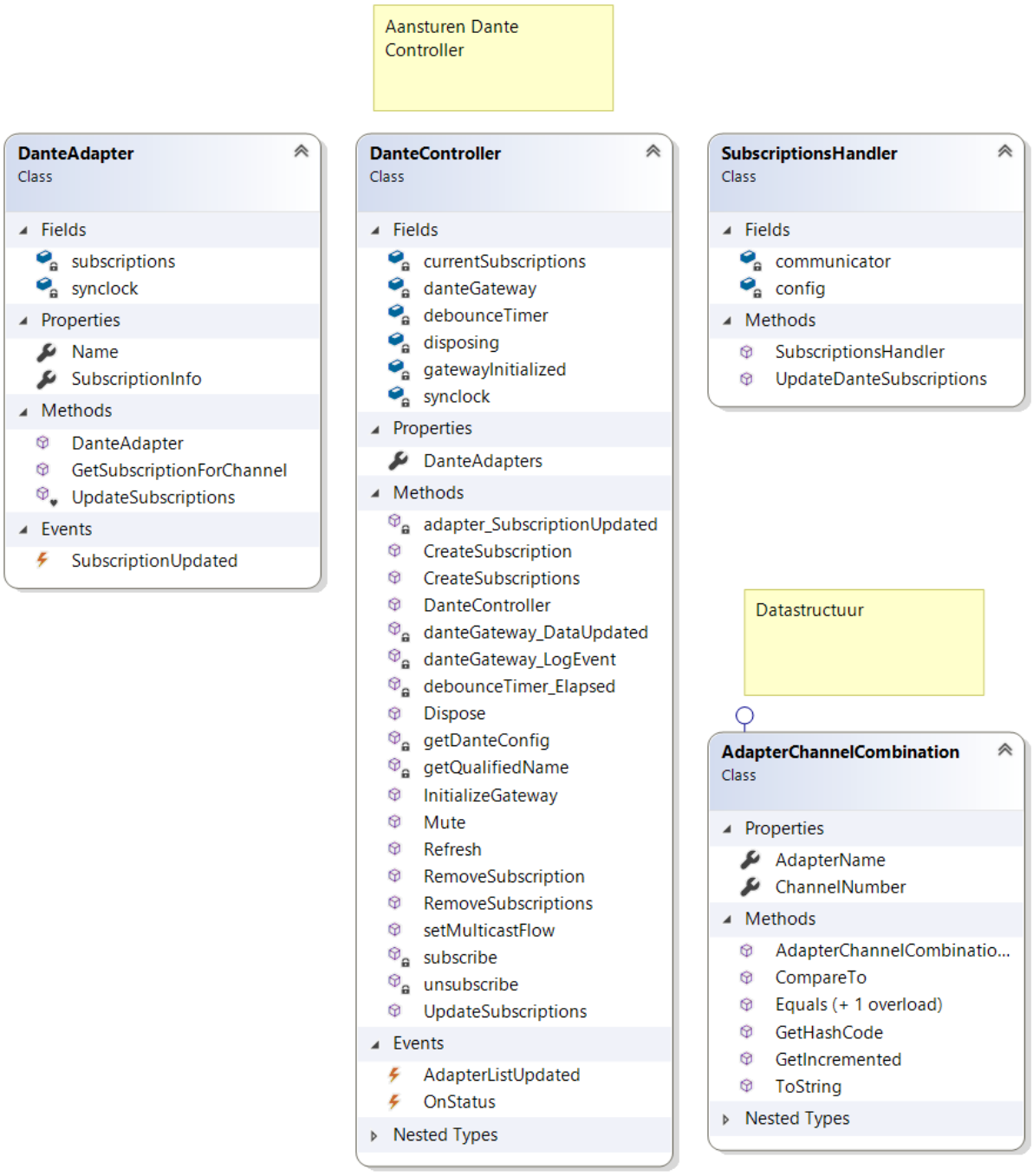


Figuur 9.17: Sequence diagram Login Interpreter

Een mogelijke use case van de webserver is die van een student (=interpreter/tolk).

Een student verbindt met de server, zonder URL in te geven. De gebruiker krijgt een 303 redirect naar het login scherm. De browser zal het login scherm en de bijhorende bestanden opvragen.
De student geeft zijn naam in en klikt op login. De browser van de student stuurt dan een POST request terug met URL=”login” en in de body de ingegeven gebruikersnaam. Dan volgt een 303 redirect naar de Interpreter App webpagina. Alle bijhorende bestanden worden opnieuw opgevraagd en de Interpreter App wordt weergegeven op het scherm.

9.2.8 Aansturen van Dante Controller



Figuur 9.18: Klassen van virtual_iq_server verantwoordelijk voor aansturen Dante Controller

Voor de Dante Controller is er een API ter beschikking om deze aan te sturen.

Door Televic Education werden reeds twee Dynamic-link libraries (DLL's) ontwikkeld om te communiceren met deze API. Namelijk DanteBridge.dll & DanteControl.dll. Deze DLL files werden niet opgenomen in het klassendiagram.

Verder werd ook door Televic Education een klasse DanteController ontwikkeld, die ook deel uitmaakt van dit project. Deze klasse maakt intern gebruik van de DanteGateway klasse van de DanteControl.dll.

Tijdens de masterproef werden ook nog een aantal functionaliteiten toegevoegd aan de DanteController klasse.

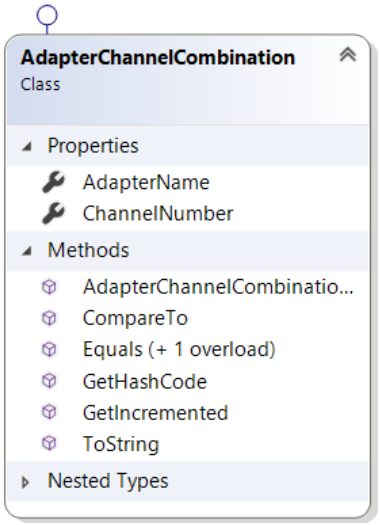
Verder is er ook de DanteAdapter klasse. Dit is een datastructuur, die in de huidige implementatie niet gebruikt wordt. Voor de volledigheid werd deze toch toegevoegd aan het klassendiagram.

De klasse SubscriptionsHandler werd tijdens de masterproef ontwikkeld. Dit is een klasse die events ontvangt van de config klasse en vervolgens de DanteController aanroept.

Tenslotte is er de tijdens de masterproef ontwikkelde klasse “AdapterChannelCombination”.

In wat volgt zullen alle klassen, met uitzondering van “DanteAdapter” verder toegelicht worden.

9.2.8.1 De klasse “AdapterChannelCombination”



Figuur 9.19: De klasse “AdapterChannelCombination”

De klasse AdapterChannelCombination is een datasctructuur die een kanaal van een bepaalde Dante Adapter voorstelt.

Er zijn twee properties, namelijk de “AdapterName” en “ChannelNumber”.

Daarnaast werden er ook nog een aantal bijkomende methoden gedefinieerd, onder meer om bruikbaar te zijn als Key in een Dictionary:

- De Equals methode geeft “true” als resultaat indien de “AdapterName” en “ChannelNumber” gelijk zijn.
- De ToString methode geeft de AdapterName en het ChannelNumber weer.
- De GetHashCode methode geeft als resultaat de hashcode van de string representatie van het object.

Verder werd ook de interface IComparable<T> geïmplementeerd. T werd hier gekozen als een ander AdapterChannelCombination object.

Om aan deze interface te voldoen moet de “CompareTo” methode geïmplementeerd worden. Zie Figuur 9.20.

```
public int CompareTo (T other);
```

Value	Meaning
Less than zero	This instance precedes `other` in the sort order.
Zero	This instance occurs in the same position in the sort order as `other`.
Greater than zero	This instance follows `other` in the sort order.

Figuur 9.20: De CompareTo methode van de IComparable<T> interface. Bron: [12]

Het voordeel van de IComparable<T> interface te implementeren is dat de klasse ook kan dienen als Key in een SortedDictionary.

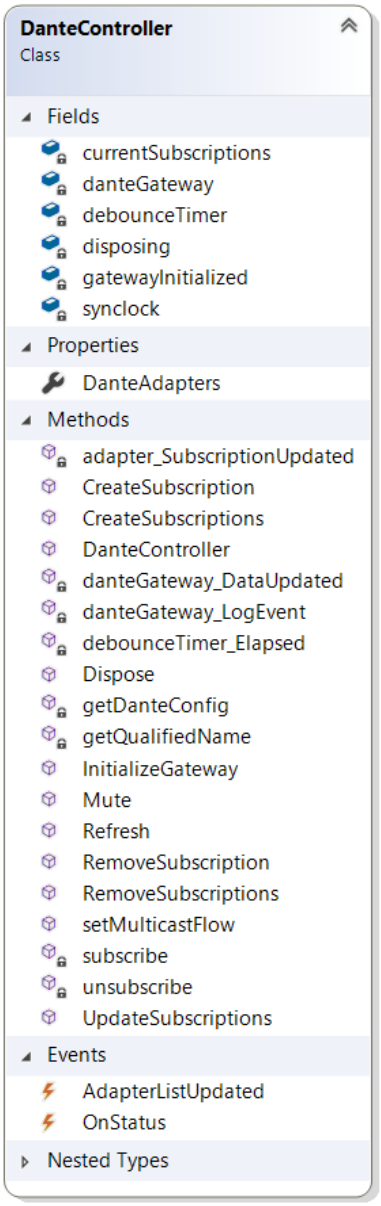
De implementatie van de CompareTo methode (Zie Figuur 9.21) specifiek in deze klasse werkt als volgt:

- Eerst wordt de DeviceName vergeleken, gebruik makende van de string.CompareTo methode.
- Indien de DeviceName gelijk is, dan wordt het ChannelNumber vergeleken, ook gebruik makende van de CompareTo methode.

```
public int CompareTo(AdapterChannelCombination other)
```

Figuur 9.21: De CompareTo methode van AdapterChannelCombination

9.2.8.2 De “DanteController” klasse



Figuur 9.22: De “DanteController” klasse

De belangrijkste functionaliteiten van de DanteController klasse zijn om subscriptions te configureren en te verwijderen en om multicast te configureren.

9.2.8.2.1 Updaten van de subscriptions

Telkens er een aanpassing aan de configuratie gebeurt, worden de gewenste subscriptions (desiredSubscriptions) opnieuw berekend.

In het programma wordt ook een overzicht bijgehouden van de subscriptions die reeds ingesteld werden (currentSubscriptions).

Beide zijn datastructuren van de vorm:

Dictionary<AdapterChannelCombination, AdapterChannelCombination>

Deze twee Dictionaries zullen vergeleken worden en op basis hiervan zullen 3 groepen gegenereerd worden:

- newSubscriptions: dit zijn volledig nieuwe subscriptions. Voorheen luisterde de subscriber naar geen enkel audiosignaal. De Dante Controller API zal aangeroepen worden om deze subscriptions aan te maken.
- changedSubscriptions: dit zijn gewijzigde subscriptions. Voorheen luisterde de subscriber naar een ander audiosignaal. De Dante Controller API zal aangeroepen worden op dezelfde manier als dat gebeurt bij een nieuwe subscription. Vermits een kanaal slechts naar een ander kanaal kan luisteren, zal de Dante Controller impliciet ook de vorige subscription verwijderen.
- toRemoveList: voorheen luisterde deze subscriber naar een audiosignaal, maar nu luistert deze naar geen enkel audiosignaal meer.

De Dante Controller geeft ook een statuscode mee of de subscription al dan niet geslaagd is. Indien de subscription geslaagd is, zullen ook de “currentSubscriptions” geüpdatet worden.

9.2.8.2.2 Multicast (setMulticastFlow)

- **Probleemstelling**

Volgens de handleiding van Dante Controller is een "flow" gedefinieerd als een stream van audiopakketten tussen twee devices. Een flow kan tot 4 audiokanalen tegelijk bevatten. [13]

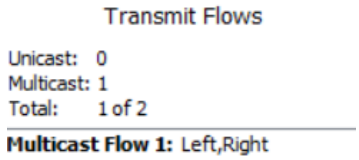
Het aantal flows dat tegelijkertijd ontvangen en verzonden kan worden is beperkt.

Bij Dante AVIO kunnen slechts 2 flows tegelijkertijd verzonden worden en 2 flows ontvangen worden. Bij Dante Virtual Soundcard is er de mogelijkheid tot 16 flows in elke richting.

- **Dante AVIO**

Doordat er 2 output flows zijn kan, bij unicast, eenzelfde tolk slechts door 2 verschillende gebruikers tegelijk beluisterd worden, wat niet wenselijk is.

De oplossing voor dit probleem is gebruik maken van multicast streaming voor de Dante AVIO Adapters. Dit is mogelijk doordat alle gebruikers verbonden zijn met eenzelfde subnet.



Figuur 9.23: Multicast flow van een AVIO adapter, schermafbeelding uit Dante Controller

Telkens de setup wijzigt wordt een "AdapterListUpdated" event gegenereerd in de DanteController klasse van de virtual_iq_server.

Telkens er een AdapterListUpdated event gegenereerd werd, wordt de methode "setMulticastFlow" van DanteController aangeroepen. Deze laatstgenoemde methode itereert over alle Adapters.

Er wordt van uitgegaan dat het om een Dante AVIO adapter gaat, indien aan de volgende voorwaarden voldaan is:

- De Adapter heeft twee inputkanalen.
- De Adapter heeft twee outputkanalen.

In dat geval wordt een multicast flow gecreëerd door middel van de CreateMulticastFlow methode (zie Figuur 9.24).

```
public bool CreateMulticastFlow(string device, uint slotCount, uint
txChannelID)
```

Figuur 9.24: De CreateMulticastFlow methode van de DanteGateway klasse

De methode stelt een aantal opeenvolgende kanalen in voor multicast. Het nummer van het eerste kanaal wordt gespecifieerd in txChannelID en het aantal opeenvolgende kanalen wordt gespecifieerd in slotCount. In dit geval is “txChannelID”=1 en “slotCount”=2 wat wil zeggen dat een multicast flow gecreëerd wordt voor kanalen 1 & 2 (linkeroor & rechteroor).

- **Dante Virtual Soundcard**

De Dante Virtual Soundcard heeft een groter aantal outputflows, maar is nog steeds beperkt. Daarom werd ook geprobeerd om multicast te configureren voor de Virtual Soundcard. De Dante Virtual Soundcard heeft echter 16 outputkanalen. Er werd tijdens het onderzoek geen manier gevonden om zoveel multicast streams tegelijk op te zetten, gebruik makende van de Dante Controller API.

Een mogelijke toekomstige aanpassing is om wel multicast toe te passen voor het “FLOOR” signaal.

9.2.8.2.3 De methode “getQualifiedName”

Een subscription aanleggen in de Dante Controller API gebeurt op basis van de "qualified name".

De qualified name heeft het formaat "channel@deviceName".

- Bij een Dante AVIO adapter is het channel "Left" of "Right".
- Bij een Virtual Soundcard gaat het channel van 01 tot 16. Bij ééncijferige kanaalnummers moet er altijd een nul voorafgaan.

Voorbeelden van geldige qualified names:

- Left@AVIOUSB-UNICOS-student1
- 05@TLV-TRA-RES9

```
private string getQualifiedName(string name, int index)
```

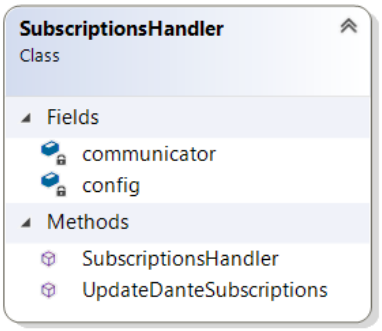
Figuur 9.25: Method header van getQualifiedName

De methode “getQualifiedName” converteert op basis van de deviceName en de index (=channelnumber) naar de beschreven notatie.

Indien de deviceName voorafgaat door “AVIOUSB-“ wordt ervan uitgegaan dat het om een Dante AVIO adapter gaat en wordt de “Left”/”Right” notatie gebruikt. In andere gevallen wordt een tweecijferig kanaalnummer gebruikt.

De methode werd reeds ontwikkeld door Televic Education, maar moest tijdens de masterproef nog aangepast worden om gebruik te maken van een tweecijferig kanaalnummer.

9.2.8.3 De “SubscriptionsHandler” klasse



Figuur 9.26: De “SubscriptionsHandler” klasse

```
public SubscriptionsHandler(Communicator communicator, Config config)

public void UpdateDanteSubscriptions(Dictionary<AdapterChannelCombination,
AdapterChannelCombination> desiredSubscriptions)
```

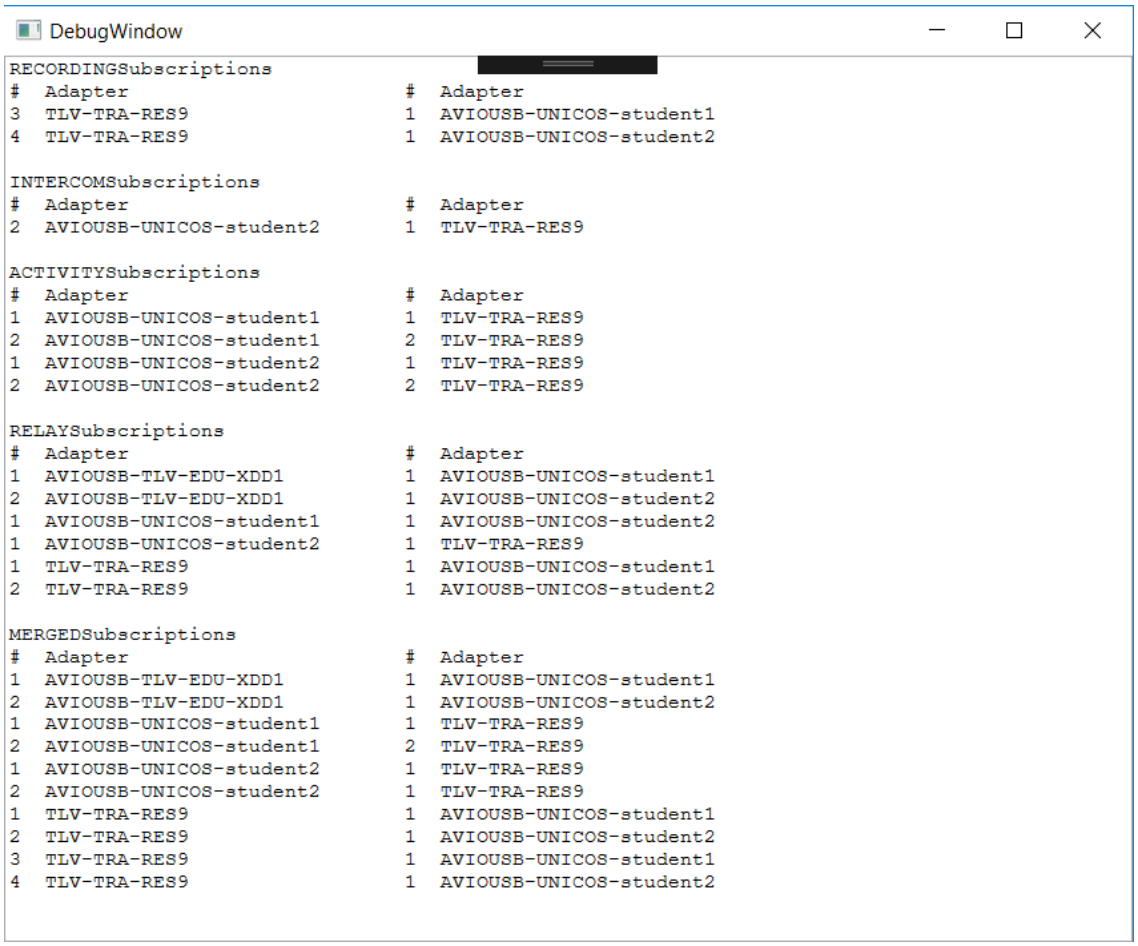
Figuur 9.27: Methoden van de SubscriptionsHandler klasse

De subscriptionshandler klasse.is verantwoordelijk voor de afhandeling van events. Bij het aanmaken van een SubscriptionsHandler, zullen events uit de config klasse gebruikt worden om door te sturen naar de Communicator klasse.

Ter info: de Communicator klasse is een door Televic Education ontwikkelde overkoepelende communicatieklasse, met als belangrijkste verantwoordelijkheid:

- het hosten van een HttpServer;
- het hosten van een WebSocketServer;
- het aansturen van de Dante Controller.

9.2.8.4 Het Debug Window



Figuur 9.28: Debug Window, met overzicht van subscriptions volgens type

In het Debug Window, te zien in Figuur 9.28 staan de subscriptions vermeld volgens type: RecordingSubscriptions, IntercomSubscriptions, ActivitySubscriptions en RelaySubscriptions.

Onderaan staat een overzicht van de samengevoegde subscriptions.

Zoals reeds eerder vermeld zijn Subscriptions altijd van één AdapterChannelCombination naar een andere AdapterChannelCombination. In de linkerkolom zien we de subsribers, in de rechterkolom de kanalen waarop geabonneerd werd. In beide gevallen wordt eerst het ChannelNumber en vervolgens de AdapterName vermeld.

Deze GUI was voornamelijk nuttig tijdens het ontwikkelen van de applicatie.

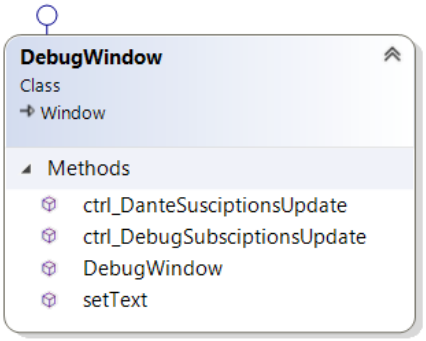
In Figuur 9.28 wordt ook de prioriteit gedemonstreerd bij het samenvoegen van subscriptions.

Zo is kanaal 2 van “AVIOUSB-UNICOS-student2” zowel geabonneerd via intercom, als via een Activity.

Intercom Subscription: 2 AVIOUSB-UNICOS-student2 ➔ 1 TLV-TRA-RES9

Relay Subscription: 2 AVIOUSB-UNICOS-student2 ➔ 2 TLV-TRA-RES9

Bij de samengevoegde subscriptions (MERGEDsubscriptions) zien we dat enkel de intercom subscription overblijft, aangezien deze een hogere prioriteit heeft.



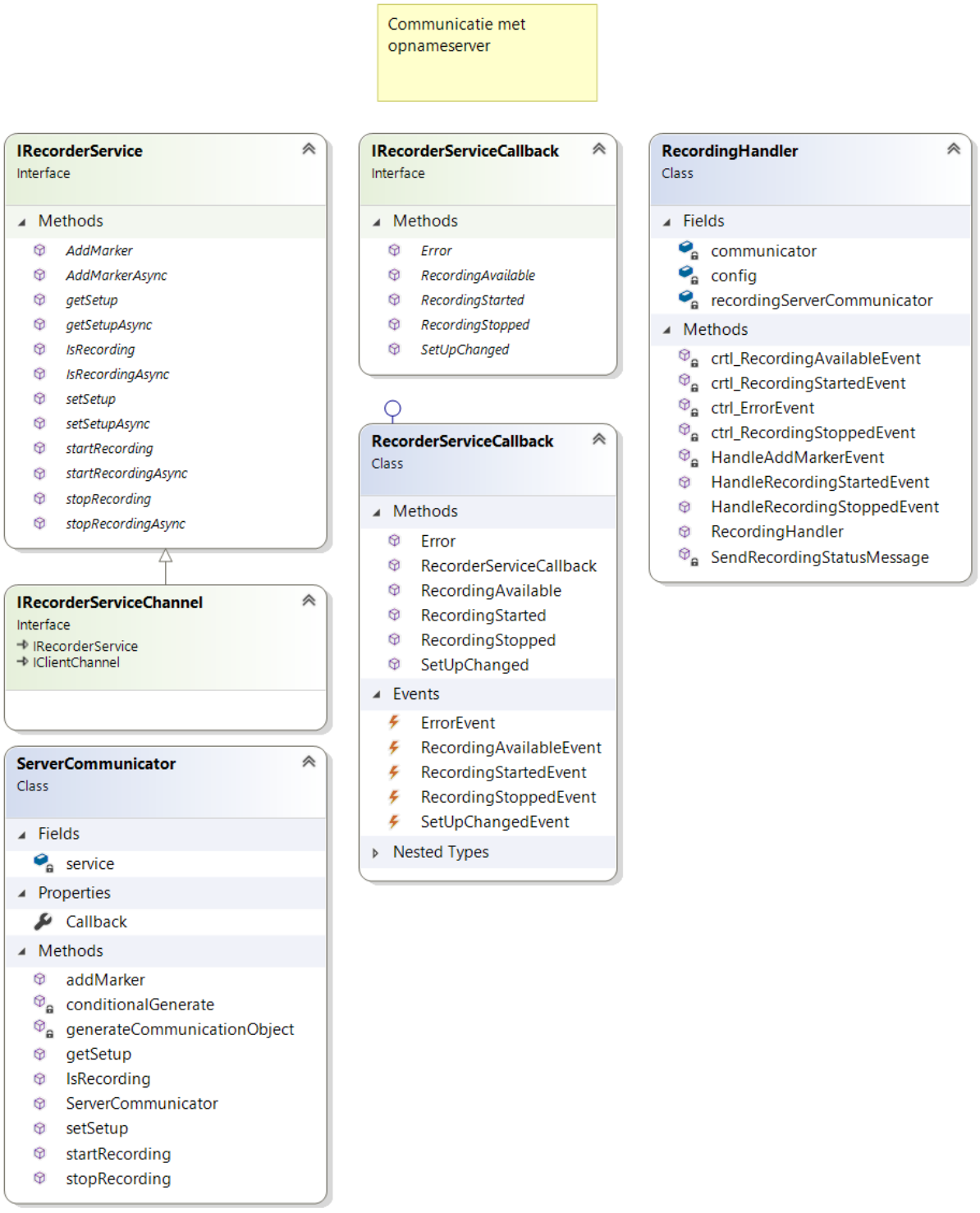
Figuur 9.29: Diagram van de DebugWindow klasse

Het Debug Window wordt telkens opgestart bij het opstarten van de virtual_iq_server.

De methode crtl_DebugSubscriptionsUpdate (zie Figuur 9.29) is geabonneerd op het event “DebugSubscriptionsUpdate” (zie Tabel 9-3 op p45).

De GUI wijzigt dus telkens wanneer de subscriptions wijzigen.

9.2.9 Aansturen van de recording API



Figuur 9.30: Klassen van virtual_iq_server verantwoordelijk voor aansturen opnameserver

Voor de communicatie met de opnameserver wordt gebruik gemaakt van Windows Communication Foundation (WCF).

De werking van de klassen wordt toegelicht in wat volgt.

Om gebruik te kunnen maken van de WCF service, moet de service eerst uitgevoerd worden op de server.

Om in Visual Studio gebruik te kunnen maken van de service, moet rechts geklikt worden op het betreffende project dat als client zal fungeren en vervolgens moet de optie “Add” ➔ “Service Reference” gekozen worden.

Vanaf dan worden er een aantal klassen toegevoegd aan het project die gebruikt kunnen worden om de API mee aan te sturen.

De twee interfaces “IRecorderService” en “IRecorderServiceCallback” zijn gelijk aan de interfaces die ter beschikking werden gesteld door de server. Ook werd de interface “IRecorderServiceChannel” toegevoegd.

Daarnaast werd ook de klasse “RecorderServiceReference.RecorderServiceClient” toegevoegd (zie het volgende onderdeel). Deze klasse is echter niet vermeld in het klassendiagram.

De klassen “ServerCommunicator” “RecorderServiceCallback” en “RecordingHandler” werden manueel geïmplementeerd.

9.2.9.1 RecorderServiceReference.RecorderServiceClient

Doordat gebruik gemaakt wordt van WCF, wordt op de client (de virtual_iq_server) een proxy object ter beschikking gesteld.

Dit proxy object is van het type “RecorderServiceReference.RecorderServiceClient” dat de IRecorderService interface implementeert. We kunnen dus alle methoden van de API aanroepen op dit object. Bij het aanroepen van een methode zorgt WCF achter de schermen voor de communicatie met de opnameserver.

Verder implementeert het object de “CommunicationObject” interface.

Daardoor heeft het object verschillende mogelijke CommunicationStates (System.ServiceModel.CommunicationState). Deze states worden weergegeven in de “State” property. [14]

De mogelijke states zijn:

- Closed
- Closing
- Created
- Faulted
- Opened
- Opening

In de “Opened” state kan communicatie plaatsvinden.

Bij de communicatie met de server kunnen er verschillende zaken mislopen. Zo is het bijvoorbeeld mogelijk dat de server niet aanstaat, of dat er problemen zijn met de netwerkverbinding.

In dat geval bevindt het proxy object zich in de “Faulted” state.

Wanneer de communicatie mislukt bij het aanroepen van een methode, wordt een CommunicationException geworpen. (System.ServiceModel.CommunicationException).

9.2.9.2 ServerCommunicator

Omdat er verschillende zaken kunnen mislopen bij de communicatie met de WCF API, werd een nieuwe klasse ServerCommunicator gedefinieerd. Deze klasse heeft een RecorderServiceClient object als member, alsook een property van het type “RecorderServiceCallback”.

Deze klasse heeft dezelfde methoden als de IRecorderService interface, met het verschil dat void methoden nu een “bool” als resultaat teruggeven.

Bij eender welke publieke methode die wordt aangeroepen op het object, wordt eerst de private methode “conditionalGenerate” aangeroepen.

Dan wordt geprobeerd om met een try/catch structuur de betreffende methode van RecorderServiceClient klasse aan te roepen. In het geval een CommunicationException optreedt, geeft de methode een “false” terug als resultaat, of een null, afhankelijk van het terugkeertype.

Volgende methoden werden geïmplementeerd:

- **conditionalGenerate**

Er werd een methode conditionalGenerate gedefinieerd.

```
private void conditionalGenerate()
```

Figuur 9.31: De "conditionalGenerate" methode

Indien het proxy object zich in de “Faulted” state bevindt, wordt de methode “generateCommunicationObject” aangeroepen.

Indien dat niet het geval is, dan doet deze methode niets. Vandaar de naam conditionalGenerate.

- **generateCommunicationObject**

```
private void generateCommunicationObject()
```

Figuur 9.32: De "generateCommunicationObject" methode

Deze methode zorgt voor het aanmaken van een CommunicationObject en daarmee ook het opzetten van een nieuwe connectie met de WCF service.

Indien er nog een CommunicationObject aanwezig is, wordt eerst deze verbinding verbroken. Vervolgens zal er een nieuw “RecorderServiceCallback” object aangemaakt worden.

Hierna wordt een nieuw proxy object aangemaakt (en daarmee dus ook een nieuwe connectie). Indien er Callbacks ontvangen worden via de connectie, zal de RecorderServiceCallback klasse aangeroepen worden.

9.2.9.3 De RecorderServiceCallback klasse

De RecorderServiceCallback klasse specifieert wat er moet gebeuren indien een callback werd ontvangen van de server.

Er werd gekozen om telkens er een callback methode wordt aangeroepen, een bijhorend Event te genereren waarop gesubscribed kan worden.

9.2.9.4 RecordingHandler

Bij het aanmaken van een RecordingHandler klasse, moeten drie referenties meegegeven worden (Zie Figuur 9.33).

```
public RecordingHandler(Config config, ServerCommunicator
recordingServerCommunicator, Communicator communicator)
```

Figuur 9.33: De constructor van RecordingHandler

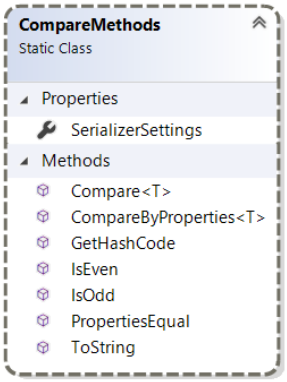
De belangrijkste verantwoordelijkheid van deze klasse is subscriben op verschillende events en de juiste acties ondernemen wanneer een event plaatsvindt.

De “ServerCommunicator” klasse heeft als property een “RecorderServiceCallback”. Er wordt gesubscribed op de verschillende events van deze klasse. Bij het ontvangen van bijvoorbeeld een “RecordingStartedEvent”, wordt via de Communicator klasse een “RecordingStatus” bericht verstuurd naar de verschillende webapplicaties door middel van WebSockets.

Omgekeerd wordt ook gesubscribed op bijvoorbeeld events van de Config klasse, zoals het “AddMarkerUpdate” event. Deze events worden doorgestuurd via de ServerCommunicator.

9.3 Klassen ter ondersteuning van Unit Tests

9.3.1 De “CompareMethods” klasse



Figuur 9.34: De CompareMethods klasse van MetaDataFormat

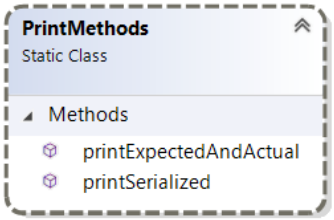
De CompareMethods klasse is ontwikkeld om gebruikt te worden als ondersteuning van testen.

Om te testen moet vergeleken worden of objecten aan elkaar gelijk zijn. (De Equals methode). Telkens manueel code schrijven om de properties te vergelijken zorgt voor veel boilerplate code. Daarom werd de PropertiesEqual methode ontwikkeld. Die vraagt door middel van reflection alle properties op en vergelijkt deze dan.

```
public static bool PropertiesEqual(object obj1, object obj2);
```

Figuur 9.35: De PropertiesEqual methode

9.3.2 De “PrintMethods” klasse



Figuur 9.36: PrintMethods klasse van MetaDataFormat

Enerzijds is er de printSerialized methode. Deze methode print een geserialiseerde voorstelling naar de console. Geserialiseerd betekent dat alle properties worden weergegeven in JSON formaat.

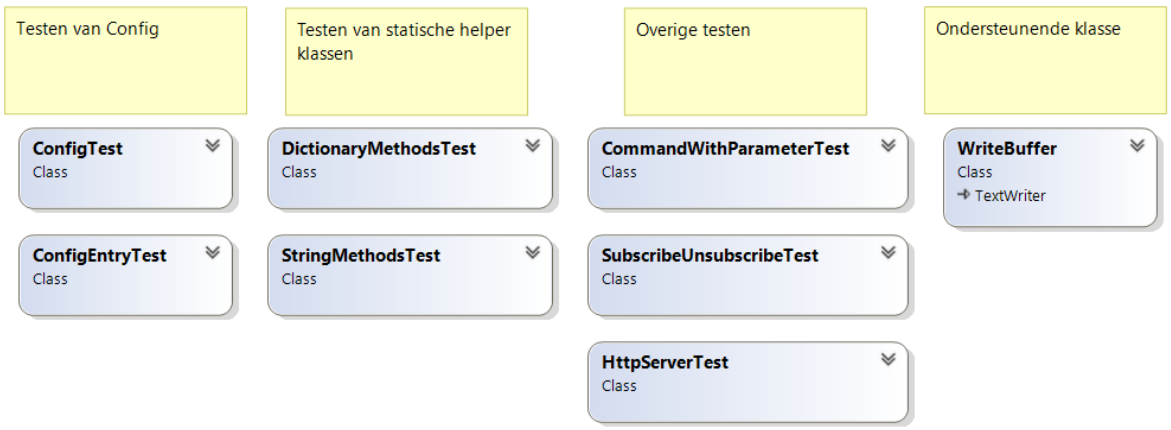
Bij een test is het de bedoeling dat we twee objecten kunnen vergelijken.

Daarom werd ook de methode printExpectedAndActual(expected,actual) gedefinieerd, die zowel de verwachte als reëele waarde uitprint, die dan vervolgens vergeleken kunnen worden. Dit is een aanvulling op de “Assert” statements die voorkomen in testen.

9.4 Unit tests: virtual_iq_server_test

De testen werden, net zoals de code zelf, ontwikkeld met behulp van Visual Studio.

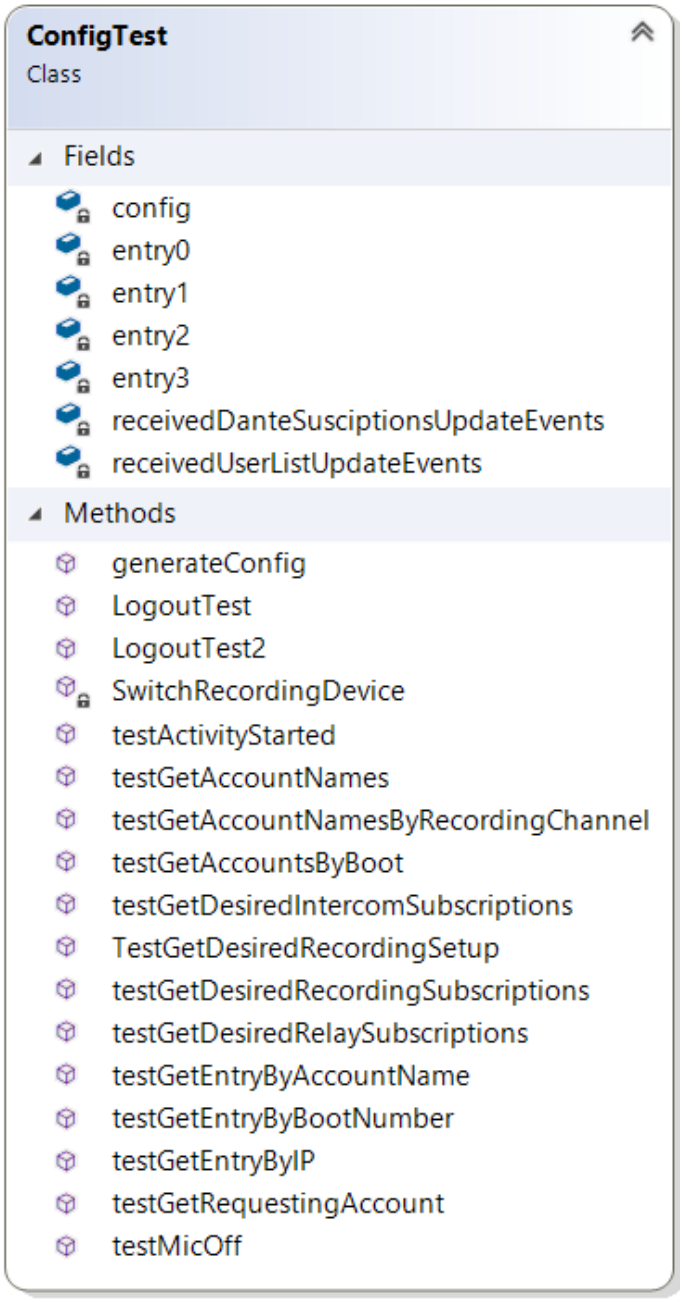
Voor het virtual_iq_server project werd een nieuw unit test project gemaakt, namelijk virtual_iq_server_test. In Figuur 9.37 is een overzicht te zien van alle testklassen van dit project.



Figuur 9.37: Klassendiagram van virtual_iq_server_test project

De meeste testen werden ontwikkeld voor de Config klasse (ConfigTest). Deze zullen in wat volgt verder uitgelegd worden.

9.4.1 Testen van de Config klasse



Figuur 9.38: Diagram van de ConfigTest klasse

Voor de Config klasse werden er het meeste testen ontwikkeld.

De use case die hiervoor gebruikt werd was de opstelling bij Televic.

Tabel 9-4: geeft een lijst met verschillende entries weer. Iedere entry komt overeen met 1 pc in het netwerk. Een entry heeft een IP adres, een naam en een type. Dat type is ofwel delegate, ofwel chairman, ofwel interpreter.

Elke test gaat uit van deze reële opstelling die bij Televic gebruikt wordt.

Tabel 9-4: De testopstelling bij Televic Education, deze waarden werden gebruikt bij de unit tests

Entry	IP	Accountnaam	Type	Booth number	Dante Adapter
0	192.168.0.220	account0	chairman	0	TLV-TRA-RES9
1	192.168.0.221	account1	delegate		AVIOUSB-UNICOS-student1
2	192.168.0.222	account2	interpreter	1	AVIOUSB-UNICOS-student2
3	192.168.0.223	account3	interpreter	2	AVIOUSB-UNICOS-student3

Entry0 is ook FLOOR en recordingdevice.

Er is bij de testen één TestInitialize. Voor de uitvoering van elke test wordt eerst de TestInitialize uitgevoerd. In de TestInitialize genereren we telkens opnieuw een Config klasse met de setup uit Tabel 9-4.

9.4.1.1 Ondersteunende methode: SwitchRecordingDevice

SwitchRecordingDevice is geen testmethode, maar wordt wel gebruikt ter ondersteuning van andere testmethoden. Door deze methode aan te roepen wordt entry1 ingesteld als opnameapparaat in plaats van entry0.

9.4.1.2 Overzicht testen

De testmethoden van de Config klasse kunnen in 4 grote groepen opgedeeld worden:

- Testen van audiosubscriptions
- Testen van LINQ queries
- Testen van Logout functionaliteit
- Testen van opnamekanalen

9.4.1.3 Testen van audiosubscriptions

Zoals reeds eerder vermeld zijn er 4 verschillende types subscriptions, namelijk RecordingSubscriptions, IntercomSubscriptions, ActivitySubscriptions en RelaySubscriptions. Deze subscriptions worden in de Config klasse telkens gegenereerd door afzonderlijke methoden. Dit stelt ons in staat om deze methoden ook afzonderlijk te testen.

Over het algemeen werkt het testen van audiosubscriptions als volgt:

- Eerst wordt manueel een Dictionary gedefinieerd met daarin de gewenste subscriptions.
- Vervolgens wordt de implementatie aangeroepen.
- Tenslotte worden beide Dictionaries vergeleken door middel van CollectionsAssert.AreEqual().

De Dictionary is van het type:

Dictionary<AdapterChannelCombination,AdapterChannelCombination>

- **testGetDesiredRecordingSubscriptions**

In deze test worden twee mogelijkheden getest.

In het geval de FLOOR en het recordingdevice eenzelfde apparaat zijn (entry0), moet kanaal 3 en 4 van “TLV-TRA-RES9” gebruikt worden voor de opname van beide tolken.

Daarna wordt SwitchRecordingDevice() aangeroepen.

Nu zullen kanaal 3 en 4 van “AVIOUSB-UNICOS-student1” (entry1) gebruikt worden voor de opname van de FLOOR. Kanaal 5 en 6 worden gebruikt voor de tolken.

De gewenste audiosubscriptions worden in beide gevallen in een Dictionary gedefinieerd en vergeleken met de reële waarde.

- **testGetDesiredRelaySubscriptions**

Een relay betekent dat een gebruiker naar een tolk naar keuze luistert. Eerst wordt een overzicht gemaakt van alle gewenste relay subscriptions.

De gewenste subscriptions worden dan vergeleken met de actuele, door middel van CollectionAssert.

- **testActivityStarted**

Tijdens een “Activity” moet iedere tolk naar het FLOOR signaal luisteren. Deze subscriptions worden hier getest.

- **testGetDesiredIntercomSubscriptions**

Intercom betekent een bidirectioneel gesprek tussen een leerling en een leraar.

De chairman (=leraar) luistert reeds in zijn rechteroor naar de betreffende student via een relay subscription. Wanneer intercom ook wordt aangezet, komt er een intercomsubscription bij van de microfoon van de leraar naar het rechteroor van de student. Op die manier ontstaat een bidirectioneel gesprek.

In deze test wordt deze laatste subscription geverifieerd.

- **testMicOff**

Gebruikers zijn in staat om hun microfoon aan of uit te zetten.

In deze test wordt in de config de microfoon van een tolk (account2) uitgezet.

Vervolgens wordt geverifieerd dat ook alle subscriptions naar deze tolk opgeheven zijn.

Er wordt een overzicht gemaakt van de gewenste relay subscriptions, maar dan zonder de subscriptions naar de tolk die zijn microfoon heeft uitgezet.

9.4.1.4 Testen van LINQ queries

- **testGetEntryByBootNumber**

Door middel van GetEntryByBootNumber kunnen we een entry opvragen van een tolk, aan de hand van zijn boothnummer. We vragen de entry op met boothnummer 2 en dat hoort entry3 te zijn, die we in de TestInitialize gedefinieerd hebben.

- **testGetEntryByAccountName**

Testen van het opvragen van de entry op basis van AccountName.

- **testGetAccountNames**

In de methode “GetInterpreterNames” wordt een array teruggegeven met de tolken, met oplopend boothnummer. In “testGetAccountNames” wordt geverifieerd of deze functionaliteit ook werkt. In dit geval is de array, oplopend volgens boothnummer:

{"floor","account2","account3" }

Merk op dat de chairmans en delegates geen boothnummer hebben en dus ook niet voorkomen in het overzicht.

- **testGetAccountsByBoot**

Hier wordt de methode “GetInterpreterNamesByBootnumberWithFloor” getest.

Deze methode is gelijkaardig aan “GetInterpreterNames”, met het verschil dat er in “GetInterpreterNamesByBootnumberWithFloor” een Dictionary wordt teruggegeven in plaats van een array.

9.4.1.5 Testen van Logout functionaliteit

- **LogoutTest**

Hier wordt de Logout functie toegepast. IP-adres “192.168.0.222”, overeenkomstig met “account2” wordt uitgelogd. Hierdoor wordt een UserListUpdateEvent event gegenereerd, met als parameter een lijst van ingelogde gebruikers. Er wordt in de test geverifieerd dat “account2” ook uit de lijst verdwenen is.

- **LogoutTest2**

Hier wordt een andere account uitgelogd, dit keer op basis van “AccountName”. Ook hier worden de parameters van het event bijgehouden en geverifieerd.

9.4.1.6 Testen van opnamekanalen

- **testGetAccountNamesByRecordingChannel**

Bij de Dante Virtual Soundcard zijn er 16 inputkanalen. Kanalen 1 en 2 worden gebruikt voor relaying. Alles vanaf kanaal 3 wordt gebruikt voor opnames.

De methode Config.GetAccountNamesByRecordingChannel() geeft een Dictionary<int,string> terug. Deze Dictionary bevat voor elk binnenkomend audiokanaal, de naam van de overeenkomstige tolk.

De correcte werking van de “GetAccountNamesByRecordingChannel” methode wordt geverifieerd in deze testmethode.

De correcte Dictionary is namelijk:

3: Eerste tolk (“account2”)

4: Tweede tolk (“account3”)

Vervolgens wordt de instructie SwitchRecordingDevice() gegeven. Daardoor zal op kanaal 3 en 4 het FLOOR signaal worden opgenomen (in stereo). Dus kanaal 5 en 6 worden nu gebruikt voor de tolken.

In dat geval is de correcte Dictionary:

5: Eerste tolk (“account2”)

6: Tweede tolk (“account3”)

- **TestGetDesiredRecordingSetup**

Er wordt een HashSet gedefinieerd met alle gewenste opnames.

Dit zijn er drie, namelijk de eerder vermelde opname van de tolk, alsook het “FLOOR” signaal.

Deze verwachte HashSet van opnames wordt vergeleken met de door de Config gegenereerde RecordingSetup.

9.5 Penetration testing

In de webserver werden verschillende veiligheidslekken geïdentificeerd en gedicht, waarvan de belangrijkste Path Traversal.

9.5.1 Path Traversal

9.5.1.1 Betekenis

Binnen een path wil tweemaal een punt (..) zeggen: de bovenliggende map.

Het pad “C:\webfiles\..” komt zo bijvoorbeeld overeen met “C:\”.

In de originele versie van de TCC webserver werd gebruik gemaakt van de Path.Combine methode van het .NET framework. [15]

Hierbij wordt de wwwroot folder gecombineerd met de URL ingegeven door de gebruiker.

Door een “..” in te geven in de URL, kon de gebruiker ook bestanden opvragen buiten de “wwwroot” uit bovenliggende mappen.

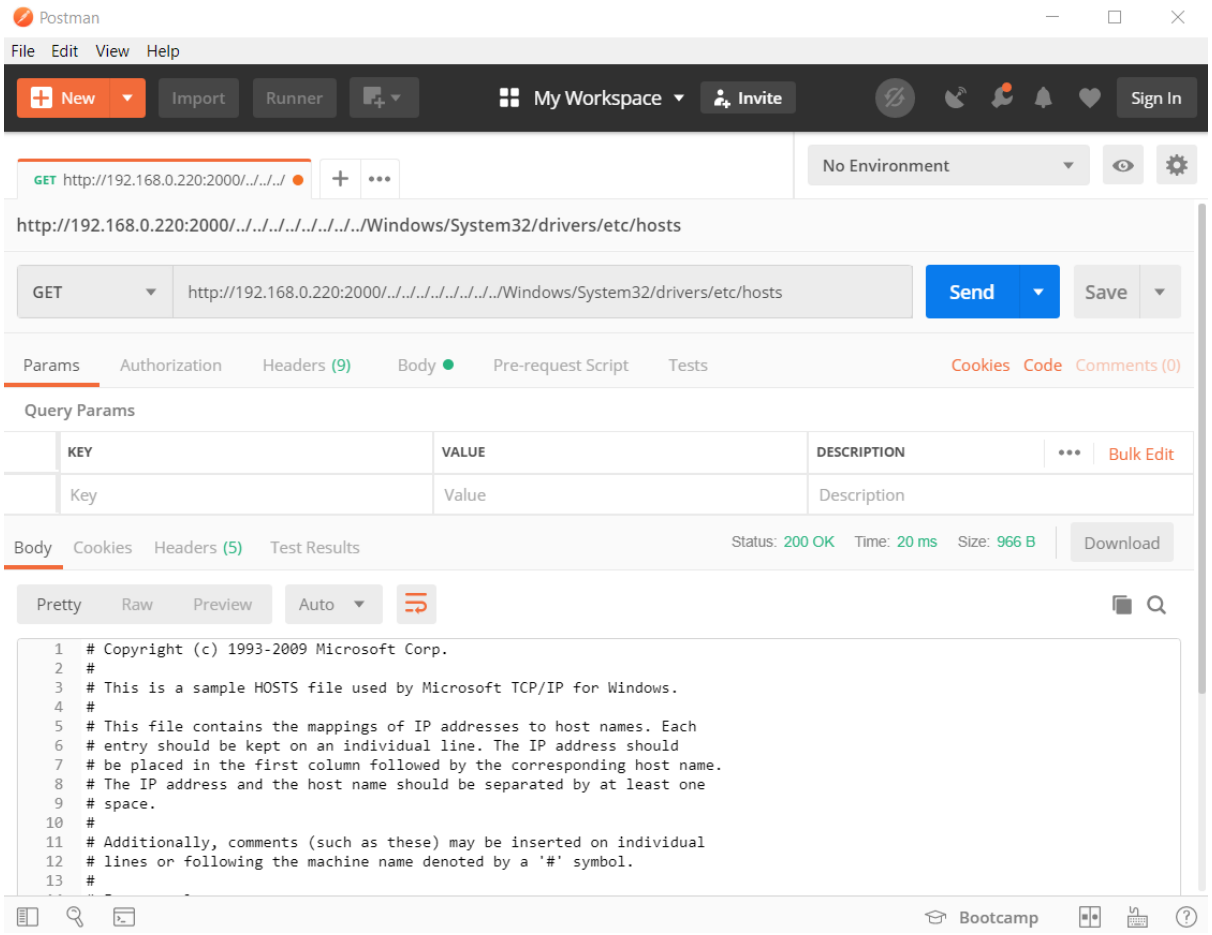
Door voldoende “..” te cominbineren, kon de gebruiker naar de C:\ schijf gaan en op deze manier eender welk bestand opvragen van de harde schijf.

De voorwaarde om gebruik te kunnen maken van Path Traversal is uiteraard dat de gebruiker de exacte locatie van het bestand op de schijf kent.

9.5.1.2 Identificatie

Dit veiligheidslek kan niet getest worden in de webbrowser. Webbrowsers verwijderen namelijk de ".." uit de URL voordat ze deze verzenden naar de server.

Daarom werd gebruik gemaakt van het programma Postman. (Zie Figuur 9.39).



Figuur 9.39: Opvragen van systeembestand door middel van path traversal

9.5.1.3 Oplossing

Dit veiligheidslek werd gedicht.

1. Eerst worden ongeldige karakters verwijderd uit de URL.
2. Path.Combine() wordt toegepast.
3. Vervolgens wordt geverifieerd op het resultaat nog een subdirectory is van de wwwroot map.

Hiervoor werd gebruik gemaakt van een reeds bestaand codevoorbeeld. [9]

9.5.2 Verkeerd geformatteerd commando

```
public static CommandWithParameter Create(string message)
```

Figuur 9.40: De Create methode van CommandWithParameter

Binnenkomende websocket commando's worden geparsed naar een CommandWithParameter object, door middel van de statische "Create" factory methode.

In voorgaande versies van het programma konden verkeerd geformatteerde commando's aanleiding geven tot een unhandled expection en dus het stilvallen van de virtual_iq_server.

Daarom werd de volledige interne logica van deze methode binnen een try/catch structuur geplaatst, die alle soorten exceptions opvangt.

Eender welke fout bij het parsen van het commando zal ervoor zorgen dat de Create methode een null aflevert. Het gevolg is dat verkeerd geformatteerde commando's worden genegeerd.

Merk op: Bij normaal gebruik worden de commando's altijd juist geformatteerd, vermits ze verzonden worden door een zelf ontwikkelde webapplicatie. Het probleem stelde zich pas wanneer manueel een websocket werd geopend en verkeerd geformatteerde commando's werden gestuurd.

9.5.3 Cross-site scripting

Cross-site scripting (XSS) is een mogelijk veiligheidslek dat optreedt wanneer gebruikers zelf in staat zijn om gegevens te posten, die later op een webpagina worden weergegeven.

Een gebruiker kan als input ook HTML tags ingeven, waaronder de <script> tag, met als gevolg dat er JavaScript code kan worden ingevoerd die zal worden uitgevoerd wanneer andere gebruikers de webpagina laden.

Vermits tolken via het loginscherm een naam dienen in te geven, die wordt weergeven in de dropdown lijst van de delegates en chairmans, is deze applicatie ook potentieel kwetsbaar voor XSS.

Daarom werd gekozen om in de webapplicaties een "escapeHtml" functie toe te voegen. Deze zorgt ervoor dat eventuele HTML tags worden weergegeven als letterlijke tekst. [16]

10 REALTIME COMMUNICATIE TUSSEN VIRTUAL_IQ_SERVER EN WEBAPPLICATIES

10.1 Keuze van protocol

10.1.1 Vereisten

Als er een commando werd gegeven, dan moet de server ook een update geven dat dit commando werkelijk uitgevoerd is. Daarnaast kunnen ook status updates gegeven worden zoals een lijst van de gebruikers.

Oftewel kan gekozen worden voor een event gebaseerde aanpak, oftewel kan gekozen worden voor een periodiek pollen.

Daarnaast kan ook gekozen worden voor long polling, wat wil zeggen dat pas antwoord wordt gestuurd wanneer de data ter beschikking is.

De verschillende technologieën die overwogen werden, worden hieronder kort toegelicht.

10.1.2 SignalR

SignalR is een bibliotheek die ter beschikking wordt gesteld in ASP.NET. Die biedt de mogelijkheid tot bidirectionele communicatie. De webserver kan content “pushen” naar alle clients, vanaf het moment dat deze content ter beschikking is. Er zijn onder meer API’s aanwezig voor connection management.

De communicatie kan (intern) op verschillende manieren, namelijk door middel van:

- WebSockets
  Een WebSocket is een bidirectioneel communicatieprotocol, dat werkt bovenop TCP. Het werd gestandaardiseerd in RFC6455.
  WebSockets worden ondersteund door al de 6 meest gebruikte browsers.
- Server-Sent Events
  Dit stelt de client in staat om updates te ontvangen van de Server. [17]
- Long Polling
  Dit wil zeggen dat de client een request stuurt en dat de server wacht met antwoorden tot deze data ter beschikking is. [18]

SignalR kiest automatisch een type connectie.

[19] [20]

10.1.3 Asynchronous JavaScript And XML (Ajax)

Ajax is de afkorting voor “Asynchronous JavaScript And XML.”

Het geeft de mogelijkheid om asynchroon webpagina’s te herladen, door achter de schermen data te versturen.

Het heeft verschillende functionaliteiten waaronder:

- Een webpagina updaten, zonder deze te herladen.
- Data opvragen en ontvangen, na het laden van een webpagina.

Bij Ajax is het echter altijd de client die een vraag initieert en de server die hier een antwoord op geeft. [21]

10.1.4 Polling

Vanaf het moment dat de client een commando geeft, wordt er in de respons ook de link naar een statuspagina gegeven. De client vraagt periodiek de waarde van deze pagina op.

Dit is dan bijvoorbeeld een RSS feed of een Atom Feed.

10.1.4.1 Really Simple Syndication (RSS)

Een RSS is een tekstbestand, waarin nieuwsupdates kunnen worden weergegeven. Het tekstbestand wordt opgeslagen onder de vorm van XML.

10.1.4.2 Atom

Atom is een nieuwere versie die een aantal problemen van RSS probeert op te lossen. Onder andere de onduidelijkheden in de standaard en de verschillende versies. Ook biedt Atom extra functionaliteiten. [22]

10.1.5 Conclusie

Voor de realtime communicatie wordt gebruikt gemaakt van WebSockets.

De communicatie is bidirectioneel en WebSockets zijn eenvoudig in gebruik.

Een mogelijk nadeel is wel dat oudere browsers WebSockets niet ondersteunen. Maar dit is slechts een kleine minderheid. [23]

10.2 Soorten berichten

Eerst worden de mogelijke berichten van de server naar de client toegelicht, vervolgens de berichten van de client naar de server.

10.2.1 Server ➔ Client

De berichten van de server naar de client worden verzonden in JSON formaat.

In de virtual_iq_server wordt gebruik gemaakt van de klasse “MessageToClient”, met properties “MessageType” en “MessageContent”. De soort info die in de MessageContent wordt opgeslagen, is afhankelijk van het MessageType. Via Newtonsoft zal het bericht geserialiseerd worden naar JSON en verzonden naar de client.

De server broadcast alle status updates naar alle verbonden WebSockets. Enkel de webapplicaties die een bepaald type bericht nodig hebben, maken er gebruik van.

De verschillende status updates worden hier verder toegelicht.

10.2.1.1 MessageType

Figuur 10.1 geeft een overzicht van de verschillende MessageTypes.

Meer info in Tabel 10-1.

```
public enum MessageTypeEnum { Invalid, AccountList, ForceInterpreterLogout,
RecordingStatus, ActivityStatusMessage};
```

Figuur 10.1: De verschillende messageTypes

Tabel 10-1: De verschillende types WebSocket berichten van virtual_iq_server naar webapplicaties

MessageType	Voorwaarde	MessageContent	Gevolg
Invalid	Standaard waarde van de enum. Dit MessageType wordt in normale omstandigheden niet verzonden.	Niet van toepassing	Niet van toepassing
AccountList	Er logt een tolk in of uit.	Een array met de namen van alle ingelogde tolken, in volgorde van boothnummer. Element 0 is de FLOOR.	Deze accountlist wordt in de Chairman App en Delegate App gebruikt om de dropdown lijst te updaten.
ForceInterpreterLogout	Een chairman klikt op de knop “Logout Interpreters.”	Leeg	Alle tolken loggen uit en krijgen terug het login scherm te zien.
RecordingStatus	De opnameserver stuurt een statusupdate.	“Started”, “Stopped”, “Available”, “ServerError”, “CommunicationError”.	De Chairman App zal het bericht verwerken.
ActivityStatusMessage	Een chairman heeft de opdracht gegeven om een Activity te starten of te stoppen.	“Started” of “Stopped”.	De tekst op de knop in de Chairman App zal gewijzigd worden.

```
{
  "MessageType": "AccountList",
  "MessageContent": [
    "floor",
    null,
    "Lucas"
  ]
}

{
  "MessageType": "ForceInterpreterLogout",
  "MessageContent": null
}

{
  "MessageType": "ActivityStatusMessage",
  "MessageContent": {
    "StatusType": "Started"
  }
}

{
  "MessageType": "ActivityStatusMessage",
  "MessageContent": {
    "StatusType": "Stopped"
  }
}

{
  "MessageType": "RecordingStatus",
  "MessageContent": {
    "StatusType": "CommunicationError",
    "MessageContent": "Could not connect to recording server."
  }
}

{
  "MessageType": "RecordingStatus",
  "MessageContent": {
    "StatusType": "Started",
    "MessageContent": "Recording \"RecordigName\" <br> Started at: 3/05/2019 16:59:05"
  }
}

{
  "MessageType": "RecordingStatus",
  "MessageContent": {
    "StatusType": "Stopped",
    "MessageContent": "Recording stopped:RecordigName"
  }
}
```


Figuur 10.2: Een overzicht van berichten verstuurd door de virtual_iq_server

10.2.2 Client ➔ Server

```
public enum CommandType { invalid, relay, mic_on, mic_off, start_activity,
stop_activity, start_intercom, stop_intercom, start_recording,stop_recording,
get_accountlist,relay_left,relay_right,add_marker, logout_interpreters };
```

Figuur 10.3: De commando's die de webapplicaties (=client) naar de virtual_iq_server kunnen sturen

Een client (webapplicatie) is in staat om commando's te versturen via WebSocket naar de virtual_iq_server. Een overzicht van de verschillende commando's is te zien in Figuur 10.3.

Het commando wordt in plaintext verzonden via WebSocket, alsook eventueel de bijhorende parameter.

Als scheidingsteken tussen commando en parameter wordt er verticale balk (pipe character) gebruikt "|". Het "relay" commando is hierop een uitzondering. Daarbij wordt het boothnummer rechtstreeks geconcateneerd aan het commando.

10.2.2.1 Gebruik van commando's

Niet alle commando's kunnen door elk type applicatie verzonden worden.

In Tabel 10-2 staat een overzicht van de types commando die kunnen verzonden worden, volgens type webapplicatie.

Tabel 10-2: Toegelaten commando's volgens type webapplicatie

	Interpreter	Delegate	Chairman
Invalid			
relay	X		
mic_on	X		X
mic_off	X		X
start_activity			X
stop_activity			X
start_intercom			X
stop_intercom			X
start_recording			X
stop_recording			X
get_accountlist		X	X
relay_left		X	X
relay_right		X	X
add_marker			X
logout_interpreters			X

Bij normaal gebruik stuurt de webapplicatie enkel de commando's uit die in Tabel 10-2 vermeld staan.

Indien er manueel een websocket wordt opgezet, kan een gebruiker verkeerd geformatteerde commando's versturen of commando's waar de gebruiker niet de correcte rechten voor heeft. Deze commando's worden genegeerd.

10.2.2.2 Betekenis commando's

In Tabel 10-3 wordt een overzicht gegeven van enkele commando's.

Tabel 10-3: Overzicht van enkele commando's gestuurd door de clients

MessageType	Voorwaarde	Parameters	Gevolg
Invalid (=Default waarde enum)	Wordt niet verzonden in normale omstandigheden.	n.v.t.	Geen effect
relay	Een interpreter klikt op een van de zes genummerde knoppen, of op de “FLOOR” knop.	Het boothnummer (0 t.e.m. 6)	Er zal een audiosubscription gemaakt worden vanaf het gekozen bootnummer, naar de hoofdtelefoon van de tolk.
mic_off	Dit commando wordt verzonden wanneer een interpreter of chairman klikt op de “Mic on/off” knop klikt, op voorwaarde dat de microfoon momenteel aanstaat.	Geen	Deze interpreter/chairman kan niet meer beluisterd worden door anderen.
mic_on	Dit commando wordt verzonden wanneer een interpreter of chairman klikt op de “Mic on/off” knop klikt, op voorwaarde dat de microfoon momenteel uitstaat.	Geen	Deze interpreter/chairman kan terug beluisterd worden door anderen.
start_intercom	Een chairman klikt op de intercom knop.	Niet van toepassing	Er wordt een audiosubscription opgezet vanaf de microfoon van de chairman, naar de in de rechtste channel selector geselecteerde student. (In diens rechteroor).
start_recording	Dit commando wordt verzonden wanneer de chairman op de “Start Activity” knop klikt, op voorwaarde dat ook de “Record” checkbox aangevinkt staat.	De naam van de opname	De virtual_iq_server zal de API van de opnameserver aanroepen.
get_accountlist	Dit commando wordt verzonden bij het opstarten van een Chairman App of Delegate App.	geen	De virtual_iq_server zal een lijst terugsturen met ingelogde gebruikers. Deze zal gebruikt worden bij het samenstellen van de dropdown lijst.

Tabel 10-3: Overzicht van enkele commando's gestuurd door de clients (vervolg)

MessageType	Voorwaarde	Parameters	Gevolg
relay_left	Een chairman of delegate kiest een ander kanaal in de linkse channel selector.	Het boothnummer	Er zal een audiosubscription gemaakt worden vanaf het gekozen boothnummer, naar het linkeroor.
relay_right	Een chairman of delegate kiest een ander kanaal in de rechste channel selector.	Het boothnummer	Er zal een audiosubscription gemaakt worden vanaf het gekozen boothnummer, naar het rechteroor.
add_marker	Een chairman klikt op een van de add_marker knoppen.	“left” of “right”	Er zal een marker worden toegevoegd aan een beluisterd kanaal. Afhankelijk van de meegegeven parameter zal dit het linkerkanaal of het rechterkanaal zijn. De virtual_iq_server zal de API van opnameserver aanroepen en daar zullen de markers worden opgeslagen in de metadata file.
logout_interpreters	Een chairman klikt op de “Logout Interpreters” knop.	geen	Het “ForceInterpreterLogout” commando zal naar alle websockets gebroadcast worden, met als gevolg dat de tolken uitloggen.

11 CONCLUSIE

Deze masterproef (*Studie en ontwikkeling van een softwaregebaseerd systeem met gecentraliseerde opname voor opleiding van tolken*) heeft als resultaat meerdere webapplicaties, een webserver en een opnameserver.

In het originele masterproef voorstel opgesteld door de co-promotor (*"Virtual interpreter training system with centralized recording"),* werden **4 mogelijke doelstellingen** geformuleerd:

1) Het ontwikkelen van een gecentraliseerde opnameserver

2) Het ontwikkelen van een annotatie tool

3) Het ontwikkelen van een virtuele Lingua Interpreter Desk.

4) Het ontwikkelen van een remote control browserapplicatie.

Deze doelstellingen werden tijdens de masterproef bijgesteld. Redelijk snel werd beslist dat de tweede doelstelling geen prioriteit was, vermits er al een desktopapplicatie bestaat met gelijkaardige functionaliteit.

De eerste doelstelling, namelijk het ontwikkelen van de gecentraliseerde opnameserver, is bereikt. De server is aanspreekbaar via een API en is in staat om opnames in te lezen via Dante en deze op te slaan op de harde schijf.

De derde doelstelling, namelijk de virtuele "Lingua Interpreter Desk" werd verder geconcretiseerd tijdens het bedrijfscontact. De client applicatie werd reeds ontwikkeld door de co-promotor. Het doel was dan ook om de achterliggende back end logica te implementeren om de applicatie ook te laten werken. Enerzijds moest de werking van de relay knoppen, alsook de FLOOR knop geïmplementeerd worden. Verder moest ook de mogelijkheid toegevoegd worden om de microfoon aan- of uit te schakelen. Deze derde doelstelling is bereikt.

De vierde doelstelling, namelijk het ontwikkelen van een remote control browserapplicatie, werd ook verder geconcretiseerd tijdens het bedrijfscontact: Er moest een lijst met oefeningen (geluidsbestanden) getoond worden, met een mogelijkheid om deze opgave te starten en te stoppen, en met de mogelijkheid om tegelijkertijd een opname te starten. Verder zou er ook een "talk on the floor" knop zijn, alsook een "intercom knop. De vierde doelstelling werd nog niet expliciet uitgewerkt. Er werd wel een Chairman App ontwikkeld, waarvan de functionaliteit grotendeels overlapt met deze van de remote control applicatie.

Tijdens de masterproef werd gewerkt aan een nieuw product in ontwikkeling. Vermits deze masterproef gespreid was over een volledig academiejaar, is het ook logisch dat tijdens het verloop van het project de doestellingen bijgesteld werden, afhankelijk van de veranderende prioriteiten.

Tijdens het verloop van de masterproef werden volgende **bijkomende doelstellingen** geformuleerd, die ook bereikt werden:

- Het ontwikkelen van een server, genaamd de virtual_iq_server, die de schakel vormt tussen de verschillende elementen uit de opstelling.
    - De server houdt de configuratie van de opstelling bij voor elke pc: de positie, het IP, de AVIOadapter, de studentname en het type.
    - De server verwijst afhankelijk van het IP adres, de gebruiker door naar de juiste webpagina. (interpreter/delegate/chairman)
- Het ontwikkelen van een login procedure voor de tolken.
- Het ontwikkelen van een Delegate App, met twee channel selectors voor de verschillende kanalen.
- Het ontwikkelen van een Chairman App, die extra functionaliteit toevoegt aan de Delegate App. Deze functionaliteiten zijn: Intercom en de mogelijkheid om Activities te starten en te stoppen, met ook de mogelijkheid om tegelijkertijd een opname te starten. Tenslotte is er ook de mogelijkheid om alle tolken die op dat moment ingelogd zijn, tegelijk uit te laten loggen.

Daarnaast waren er ook een aantal optionele bijkomende doelstellingen, die niet haalbaar bleken te zijn binnen de beschikbare tijd:

- Testen op iPad/Raspberry PI
- Gebruik maken van WebRTC voor audiostreaming.

Tenslotte werden ook op eigen initiatief een aantal functionaliteiten uitgewerkt die niet expliciet in de doelstellingen stonden, maar die wel logisch leken. Vermits er bijvoorbeeld al een login functie was, leek het ook logisch om een bijhorende logout functie te maken.

Eén van de vereisten van de API van de opnameserver was de mogelijkheid om een marker toe te voegen. Daarom leek het ook logisch om deze functionaliteit te voorzien in de Chairman App, vermits deze functionaliteit anders niet aanspreekbaar was voor een gebruiker.

Tijdens de masterproef werd er uiteraard ook aandacht besteed aan de **betrouwbaarheid** van de applicatie. Grote delen van de virtual_iq_server werden bijvoorbeeld getest door middel van unit tests.

Ook werd er aandacht besteed aan de veiligheid van de applicaties. Meerdere veiligheidslekken, waaronder Cross-site scripting (XSS) werden gedicht.

Zowel de co-promotor als ikzelf hadden veel ideeën over mogelijke functionaliteiten die konden toegevoegd worden. Maar vermits het over een masterproef gaat met beperkte tijd, dienden keuzes gemaakt te worden over wat prioritair was en wat niet. Het resultaat is meerdere webapplicaties, een webserver en opnameserver.

Voor mij persoonlijk was deze masterproef een leerrijke en interessante ervaring.

Inhoudelijk heb ik veel bijgeleerd over C# en de verschillende klassen van het .NET framework. Ook heb ik leren werken met verschillende packages zoals Nuget. Ik heb ook geleerd om een groot project aan te pakken en uit te voeren.

Ik ben tevreden met het geleverde resultaat. Ik hoop dat ik met deze webapplicaties, webserver en opnameserver een nuttige bijdrage heb kunnen leveren aan interpreterQ van Televic Education.

12 FUTURE WORK

Hierbij volgt een overzicht van mogelijke extra functionaliteiten.

12.1 Web Real-Time Communication (WebRTC)

Web Real-Time Communication (WebRTC) is een open-source project, dat een API ter beschikking stelt voor realtime communicatie. WebRTC maakt onder meer peer-to-peer audiostreams mogelijk tussen de verschillende webbrowsers.

WebRTC kan op deze manier een alternatief bieden voor Dante audiostreaming.

Dit biedt volgende voordelen:

- Door geen gebruik meer te maken van de Dante AVIO hardware of proprietary software, kan de kost van de opstelling verder verlaagd worden.
- Gebruik maken van WebRTC, maakt het mogelijk om ook deel te nemen aan een opgave buiten het subnet van de onderwijsinstelling. De virtual_iq_server kan verbonden worden met het internet, om op deze manier afstandsleren mogelijk te maken.

12.2 Remote control applicatie

Televic heeft reeds een desktopapplicatie ontwikkeld voor de leraar, genaamd de Interpreter Control Center (ICC). Daarmee kan de leraar onder meer een geluidsfragment starten en stoppen.

Het idee is om een webapplicatie te maken die deze functionaliteit op afstand kan aansturen.

De leerlingen kunnen gebruik maken van deze webapplicatie waarbij ze dezelfde tools ter beschikking hebben als de leraar, om aan zelfstudie te doen.

Volgende functionaliteit zou geïmplementeerd kunnen worden:

- Een lijst van de geluidsfragmenten tonen. Elk geluidsfragment kan gestart of gestopt worden, al dan niet met gecentraliseerde opname.
- Een “talk on the floor” knop, waarmee de student zelf het FLOOR signaal levert.

Deze functionaliteit zou ook kunnen worden toegevoegd aan de Chairman App, in plaats van een aparte applicatie te ontwikkelen.

12.3 Configureren virtual_iq_server

Momenteel is het enkel mogelijk om de statische configuratie te wijzigen door manueel een JSON file aan te passen die op deze server opgeslagen staat.

Een mogelijke uitbreiding is een “config” webpagina.

Hierop wordt een lijst weergegeven met alle entries (IP adres, application type, naam van de AVIO Adapter). Op deze webpagina dient het ook mogelijk te zijn een entry toe te voegen of verwijderen.

12.4 Uitlezen markers in applicatie

In de metadata file bij de opnames wordt onder meer een lijst opgeslagen van de markers van alle opnames, in chronologische volgorde.

Door Televic Education werd reeds een desktopapplicatie ontwikkeld om de markers weer te geven. De functionaliteit om de markers uit te lezen uit de metafile file moet nog geïmplementeerd worden door de ontwikkelaar van deze desktopapplicatie bij Televic Education.

12.5 Meer opnames

12.5.1 Huidige implementatie

Binnen Dante Virtual Soundcard werd gekozen om gebruik te maken van Windows Driver Model (WDM) voor de geluidskaarten. Indien de optie WDM is aangevinkt, worden er 8 stereo geluidskaarten toegevoegd, wat 16 inputkanalen en 16 outputkanalen betekent.

De eerste twee audiokanalen worden gereserveerd voor de Chairman App, zo kan de leraar in elk oor luisteren naar een kanaal naar keuze. Op deze manier zijn er 14 kanalen ter beschikking voor gecentraliseerde opname.

Er zijn twee manieren waarop dit aantal kan worden verhoogd, namelijk door meer opnames per server, of door extra opnameservers toe te voegen.

12.5.2 Meer opnames per server

Door gebruik te maken van *Audio Stream Input/Output* (ASIO) in plaats van WDM kunnen 64 inputkanalen en 64 outputkanalen gebruikt worden.

Vermits de code van de opnameserver momenteel ontwikkeld is om gebruik te maken van WDM, zal een deel hiervan moeten herschreven worden.

Onderzoek moet uitwijzen hoeveel code herschreven moet worden. Verder is het geen zekerheid dat de netwerkkaart, CPU en harde schijf van de server performant genoeg zullen zijn.

12.5.3 Meerdere opnameservers

De eerste opnameserver neemt 14 geluiden op. Er kan geopteerd worden om extra opnameservers toe te voegen, die dan elk nog 16 extra geluiden opnemen (Vermits er op

deze extra opnameservers geen Chairman App wordt uitgevoerd, zijn er 2 extra kanalen ter beschikking.)

Indien geopteerd wordt voor deze oplossing, betekent dat de virtual_iq_server dient herschreven te worden om de API van meerdere opnameservers aan te spreken.

Het is ook mogelijk om daarnaast code te schrijven om na de opname, de opnamebestanden te centraliseren naar één server.

12.6 Extra toevoegingen aan de Interpreter App

Figuur 12.1: De interpreter apps, met de Quality Calculation rood omcirkeld

12.6.1 Quality Calculation

Een tolk kan kiezen om naar het “FLOOR” signaal te luisteren. Indien de tolk de taal van de spreker niet kent, kan de tolk ook kiezen om naar een andere tolk te luisteren. De vertaling gebeurt dan via een omweg zoals bijvoorbeeld het Engels. Door een extra tussenstap toe te voegen neemt de kwaliteit van de vertaling echter af en wordt extra tijdsvertraging toegevoegd. In het ergste geval luisteren tolken circulair naar elkaar.

Het idee is om voor elke andere tolk weer te geven hoeveel tussenstappen deze verwijderd is van het “FLOOR” signaal. Indien een tolk meerdere talen spreekt, kan deze kiezen voor de taal met het laagste aantal tussenstappen.

12.6.2 Configureerbare knoppen

In de huidige implementatie kan slechts naar de tolken met boothnummer tot en met 6 geluisterd worden.

Een mogelijke aanpassing is de volgende: er zijn nog steeds 6 knoppen maar de gebruiker kan zelf kiezen met welk boothnummer deze knop overeenkomt, aan de hand van een dropdown lijst. Dit is reeds de manier waarop het gebeurt bij de hardwaregebaseerde opstelling.

12.7 Tablet

De belangrijkste reden dat gebruik werd gemaakt van webapplicaties is de platformonafhankelijkheid. Ter vervanging van een pc zou ook een tablet gebruikt kunnen worden. Dit brengt echter meerdere uitdagingen met zich mee.

Eerst en vooral dient de Dante AVIO USB adapter aangesloten te worden op de tablet. De tablet dient dus een USB poort ter beschikking hebben. Er dient onderzocht te worden of er op Android/iOS drivers bestaan die kunnen communiceren met de Dante AVIO USB adapter.

Verder moet ook onderzocht hoe de drivers moeten doorverbonden worden:

- Het geluid van de microfoon (opnameapparaat) moet worden “afgespeeld” via de Dante AVIO USB adapter.
- Het inkomend geluid via de Dante AVIO USB adapter moet worden “afgespeeld” via de hoofdtelefoon.

Indien gebruik gemaakt wordt van WebRTC stellen deze moeilijkheden zich niet.

Behalve voor een tablet kan ook geopteerd worden voor een kleine computer zoals een Raspberry Pi.

12.8 Automatisch stoppen van opname

Het idee is dat de opname automatisch stopt, een bepaalde vaste tijd nadat de opname gestart werd. Dit is vooral nuttig indien de leraar de opname vergeet stop te zetten. In de API kan reeds de waarde “TimeoutMilliseconds” meegeven worden. De onderliggende functionaliteit dient echter nog geïmplementeerd te worden.

12.9 Upgrades aan de webserver

De webserver kan op nog enkele manieren verbeterd worden. Deze worden verder toegelicht.

12.9.1 Implementeren “HEAD”

De HTTP HEAD request methode is gelijkaardig aan de GET methode. Het belangrijkste verschil is dat de respons enkel de header bevat en niet de response body.

Om aan de RFC standaard te voldoen moet de HEAD methode ook geïmplementeerd worden. [24]

12.9.2 Gebruik maken van HTTP/2

De tweede versie van HTTP biedt een aantal extra functionaliteiten aan. De belangrijkste hiervan is “Server Push”. Dit wil zeggen dat de webserver reeds bestanden kan versturen, voordat de client deze opvraagt.

In het geval van een 303 redirect, kan de server ook onmiddellijk de correcte webpagina meesturen.

Bij het opvragen van een webpagina kunnen bijvoorbeeld ook de bijhorende afbeeldingen, CSS & JavaScript meegestuurd worden.

Op deze manier kan het aantal requests dat de client dient te sturen, sterk verminderd worden.

Referenties

[1] „Bootstrap Snippet Very simple login form using HTML,” [Online]. Available: https://bootsnipp.com/snippets/3kp.

[2] „Json.NET - Newtonsoft,” [Online]. Available: https://www.newtonsoft.com/json.

[3] B. Rogers, „Json.Net - Error getting value from 'ScopeId' on 'System.Net.IPAddress',” [Online]. Available: https://stackoverflow.com/questions/18668617/json-net-error-getting-value-from-scopeid-on-system-net-ipaddress.

[4] P. J. Mark Heath, „WaveInEvent.cs,” [Online]. Available: https://github.com/naudio/NAudio/blob/master/NAudio/Wave/WaveInputs/WaveInEvent.cs. [Geopend 21 Oktober 2018].

[5] „Stream Class,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.io.stream?view=netframework-4.7.2. [Geopend 19 September 2018].

[6] C. Murtagh, „NAudio.Lame,” [Online]. Available: https://github.com/Corey-M/NAudio.Lame.

[7] „WPF Controls,” [Online]. Available: https://www.tutorialspoint.com/wpf/wpf_controls.htm.

[8] „.Net: How do I check for illegal characters in a path?,” [Online]. Available: https://stackoverflow.com/questions/2435894/net-how-do-i-check-for-illegal-characters-in-a-path.

[9] „How to check if one path is a child of another path?,” [Online]. Available: https://stackoverflow.com/questions/8091829/how-to-check-if-one-path-is-a-child-of-another-path/23354773#23354773.

[10] „.NET Framework Guide,” 10 April 2018. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/.

[11] „What Is Windows Communication Foundation,” 30 Maart 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf.

[12] „IComparable<T>.CompareTo(T) Method,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.icomparable-1.compareto?view=netframework-4.8.

[13] D. C. User, „Dante Controller User,” [Online]. Available: https://dev.audinate.com/GA/dante-controller/userguide/pdf/latest/AUD-MAN-DanteController-4.1.x-v1.0.pdf.

[14] „CommunicationState Enum,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.communicationstate?view=netframework-4.8.

[15] „C# Path traversal warning when using Path.Combine,” [Online]. Available: https://stackoverflow.com/questions/48777564/c-sharp-path-traversal-warning-when-using-path-combine.

[16] „HtmlSpecialChars equivalent in Javascript?,” [Online]. Available: https://stackoverflow.com/questions/1787322/htmlspecialchars-equivalent-in-javascript.

[17] „HTML5 Server-Sent Events,” [Online]. Available: https://www.w3schools.com/Html/html5_serversentevents.asp.

[18] J. Hanson, „What is HTTP Long Polling?,” 1 December 2014. [Online]. Available: https://www.pubnub.com/blog/2014-12-01-http-long-polling/.

[19] „SignalR,” [Online]. Available: https://www.asp.net/signalr.

[20] „Introduction to ASP.NET Core SignalR,” 25 April 2018. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-2.2.

[21] „AJAX Introduction,” [Online]. Available: https://www.w3schools.com/xml/ajax_intro.asp.

[22] „Atom (standard),” [Online]. Available: https://pdfs.semanticscholar.org/31fb/fc022d20d128e15fc41fe43ac2013016643c.pdf.

[23] [Online]. Available: https://caniuse.com/#search=websockets.

[24] H. r. methods. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[25] X. D. Donder, „Virtual interpreter training system with centralized recording,” Televic Education, Izegem, 2018.

[26] „HttpListener Class,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.net.httplistener?view=netframework-4.7.2.

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS BRUGGE
Spoorwegstraat 12
8200 BRUGGE, België
tel. + 32 50 66 48 00
iiw.brugge@kuleuven.be
www.iiw.kuleuven.be