XX. reál- és humántudományi ERDÉLYI TUDOMÁNYOS DIÁKKÖRI KONFERENCIA (ETDK)
Kolozsvár, 2017. május 18-21.

A VRemote virtuális valóság platform játékok megjelenítésére és irányítására

Szerzők:
Hobaj Roland
Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Kocs Krisztián-István
Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:
dr. Simon Károly, egyetemi adjunktus
Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar

Mátis Szilárd-Gábor, szoftverfejlesztő, Codespring
Sebestyén Balázs, szoftverfejlesztő, Codespring
Zölde Attila, szoftverfejlesztő, Codespring

CODESPRING

Kivonat

A dolgozatban bemutatott VRemote projekt célja egy bárki számára könnyen elérhető virtuális valóság környezet megteremtése számítógépes játékokhoz. A forgalomban lévő drágább megoldásokkal és eszközökkel összehasonlítva, a VRemote esetében a virtuális valóság megtapasztalásához elegendő egy viszonylag egyszerűen beszerezhető VR szemüveg, melybe a telefonunkat helyezhetjük.

A szoftverrendszer szerver komponense a játszani kívánt játékkal azonos gépen fut, annak érdekében hogy a játék képét közvetíthesse a mobilalkalmazás felé. A szerver alkalmazáshoz elkészült cross-platform grafikus felület a szükséges hálózati beállítások elvégzésére, valamint statisztikák megjelenítésére ad lehetőséget.

Az Android kliensalkalmazás a sikeres kapcsolódást követően elvégzi a felé közvetített videó megjelenítését, és rajta keresztül történik a játék irányítása is. A gépen futó játék irányítása a szemüvegben elhelyezett telefon mozgását figyelembe véve történik.

Tartalomjegyzék

1. fejezet

Bevezető

A virtuális valóság napjainkban egyre nagyobb teret hódít különleges felhasználói élmény miatt. A jelenlegi megvalósítások használatához komoly hardware eszközök szükségesek, melyek nem kevés anyagi befektetéssel szerezhetőek be. A VRemote projekt elsődleges célja megpróbálni ennek a problémának a kiküszöbölését, egy elérhető VR megoldás nyújtásával.

A projekt fejlesztése 2016 nyarán indult, a Codespring mentorprogram keretén belül. Kezdetben a fejlesztő csapat Hompoth Szilárdból, Hobaj Rolandból valamint Kocs Krisztiánból állt. A projektet végigkísérő mentorok Mátis Szilárd-Gábor, Sebestyén Balázs és Zölde Atilla voltak valamint a témavezető tanár szerepét Simon Károly töltötte be. A fejlesztéshez szükséges infrastruktúrát a Codespring biztosította.

A 2016-2017-es egyetemi év első félévében Hompoth Szilárd távozását követően, a csoportos projekt tantárgy keretén belül a csapat kibővült további három új taggal név szerint Kálmán Noémi, Kiss-Budai Mátyás és Sinkler-Németh István. A tantárgy keretein belül elkészült egy cross-platform Electron grafikus felület, a szerveralkalmazás konfigurálásának megkönnyítéséhez.

A szerzők legfontosabb hozzájárulásai a projekthez az Androidos mobilalkalmazás főbb funkcionalításainak megvalósítása valamint a szerveralkalmazás fejlesztése, amelyek a dolgozat további részeibel kerülnek bemutatásra.

Elsőként a virtuális valóság fogalom ismertetésére kerül sor. Ezt követően a VRemote projekt struktúrája és főbb funckionalitásai vannak megjelenetítve. A fejlesztési módszerek és eszközök fejezeten belül a fejlesztő csapat által követett folyamatok és az azokban segítő eszközök lesznek ismertetve. A dolgozat fő célja a megvalósítás lépéseinek részletes bemutatása, úgy a szerveralkalmazás mint a kliensalkalmazás felől. Az alkalmazás működésének szemléltetése után a dolgozatot a levont következtetések és a hátramaradt fejlesztési lehetőségek feltárása zárja.

2. fejezet

A virtuális valóság bemutatása

A virtuális valóság (VR) az a számítógépes technológia, amely eszközök (szemüveg, kontrollerek, stb.) segítségével olyan valósághű hang- és képanyagot ad át amely egy saját környezetet teremt. A felhasználó az így teremtett 360 fokos és három dimenziós térben szimuláció segítségével jelen tud lenni, sőt, interakciókra is képes a különböző hardver és szoftver megoldásoknak köszönhetően. A VR szemüveget viselő felhasználó "körülnézhet" az adott térben, és a kontrollereknek köszönhetően képes műveletek elvégzésére is mint például kattintás, kiválasztás stb.

Már 1965-ben[1] megjelent egy olyan kutatási irány amely fő célja, a számítógép által kreált világ valóságossá tétele volt, de fejlesztése csak az 1990-es évektől vált igazán népszerűvé. Számos felhasználási területe van mint például modellezés, oktatás, játékipar stb. Napjainkban több eszköz jelent meg a piacon mint például Oculus rift, HTC Vive, PlayStation VR stb.

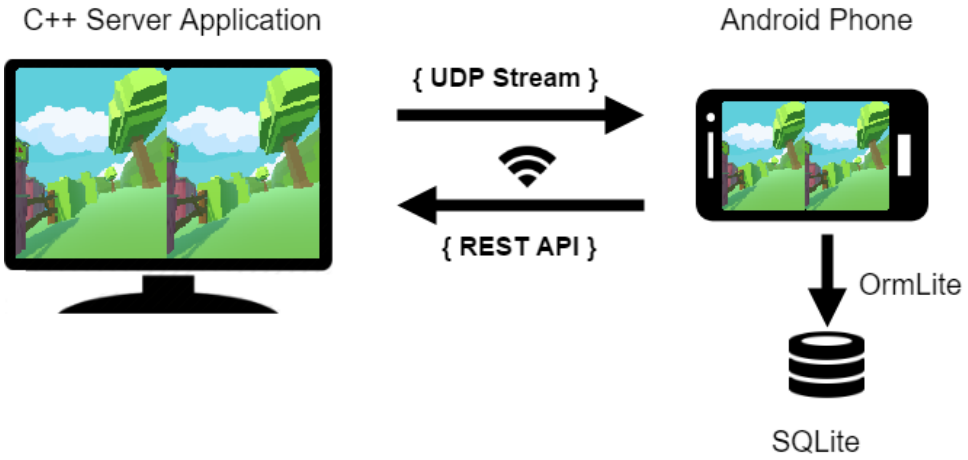
A virtuális valóság legfontosabb eszköze egy fejhez rögzíthető kijelző (head mounted display - HMD) amelyen megjelennek a megteremtett környezet elemei. A jelenleg piacon található drágább HMD eszközök esetében kábel segítségével történik a kommunikáció a környezetet megteremtő rendszer és a kijelző között. A VRemote projekt során a virtuális kép megjelenítése hálózaton keresztül történik, nagyobb mozgási szabadságot biztosítva a felhasználóknak. A Google által piacra dobott Cardboard[2] szemüvegbe behelyezhető egy okostelefon, amely kijelzőként funkcionál (HMD), ezzel nyújtván lehetőséget a virtuális valóság megtapasztalásához. A fejlesztő csapat a mobilalkalmazás tesztelésére a Google Cardboard-ot használta.

3. fejezet

A VRemote projekt

3.1. Követelmények és fontosabb funkcionalitások

A VRemote projekt követelményrendszerét a 3.1 ábra foglalja össze. Az alkalmazás két komponensből áll egy szerveralkalmazás valamint egy kliensalkalmazás. A szerveralkalmazás funkcionalitásai közé a számítógép grafikus felületének közvetítése, valamint a játék irányítása tartozik. A mobilkliens felelős a felé közvetített stream lejátszásáért valamint a játék irányításáért, melyet szenzoradatok küldésével végez. A telefon a Google által megtervezett Cardboard segítségével a felhasználó fejéhez lesz csatolva. A fejmozgatás által generált szenzoradatok a szerveralkalmazás felé lesznek továbbítva. Ezáltal a felhasználónak lehetősége van "körülnézni" a játék virtuális világában.



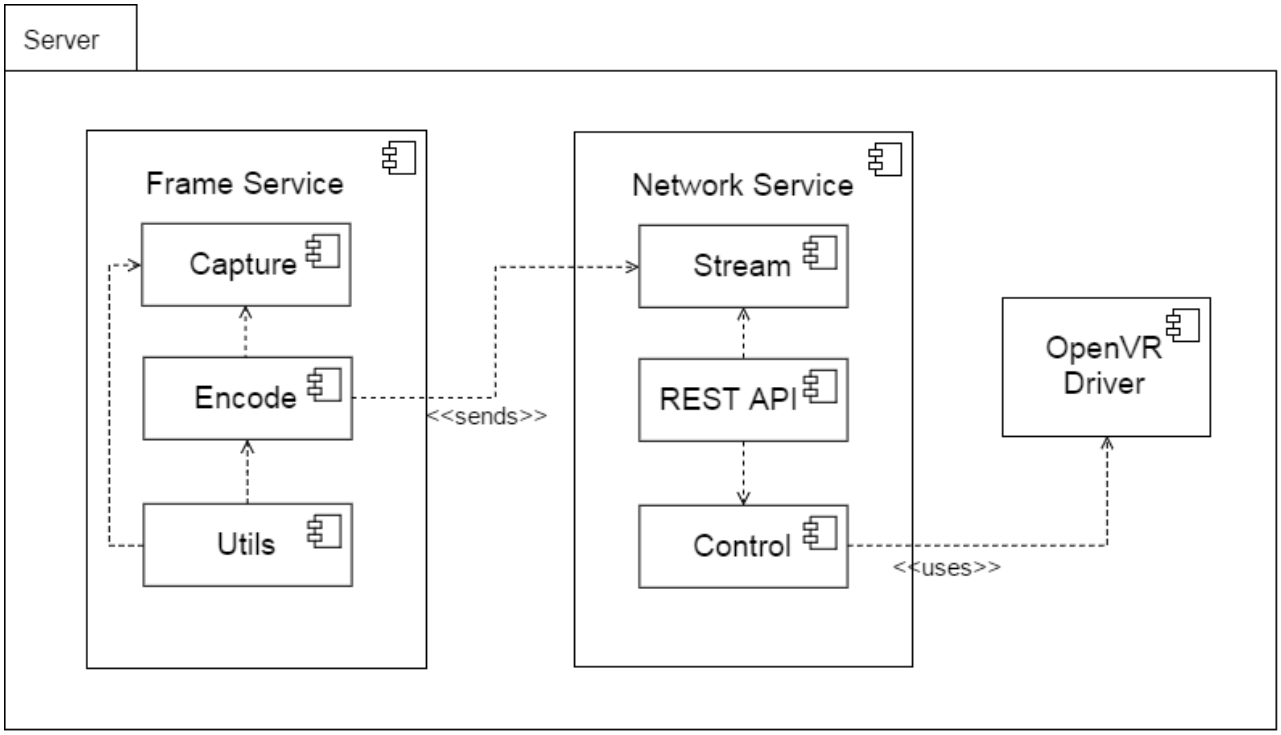
3.1. ábra. VRemote szoftverrendszer

3.2. Architektúra

A VRemote projekt két alprojektből áll, egy szerverkomponens valamint egy mobilalkalmazás. A következő fejezetben a két komponens felépítése kerül bemutatásra.

3.2.1. Szerver

A VRemote szoftverrendszer szerveralkalmazása több szolgáltatás-komponensből álló komponens, amelyet a 3.2 ábrán láthatunk standard UML diagram segítségével ábrázolva.



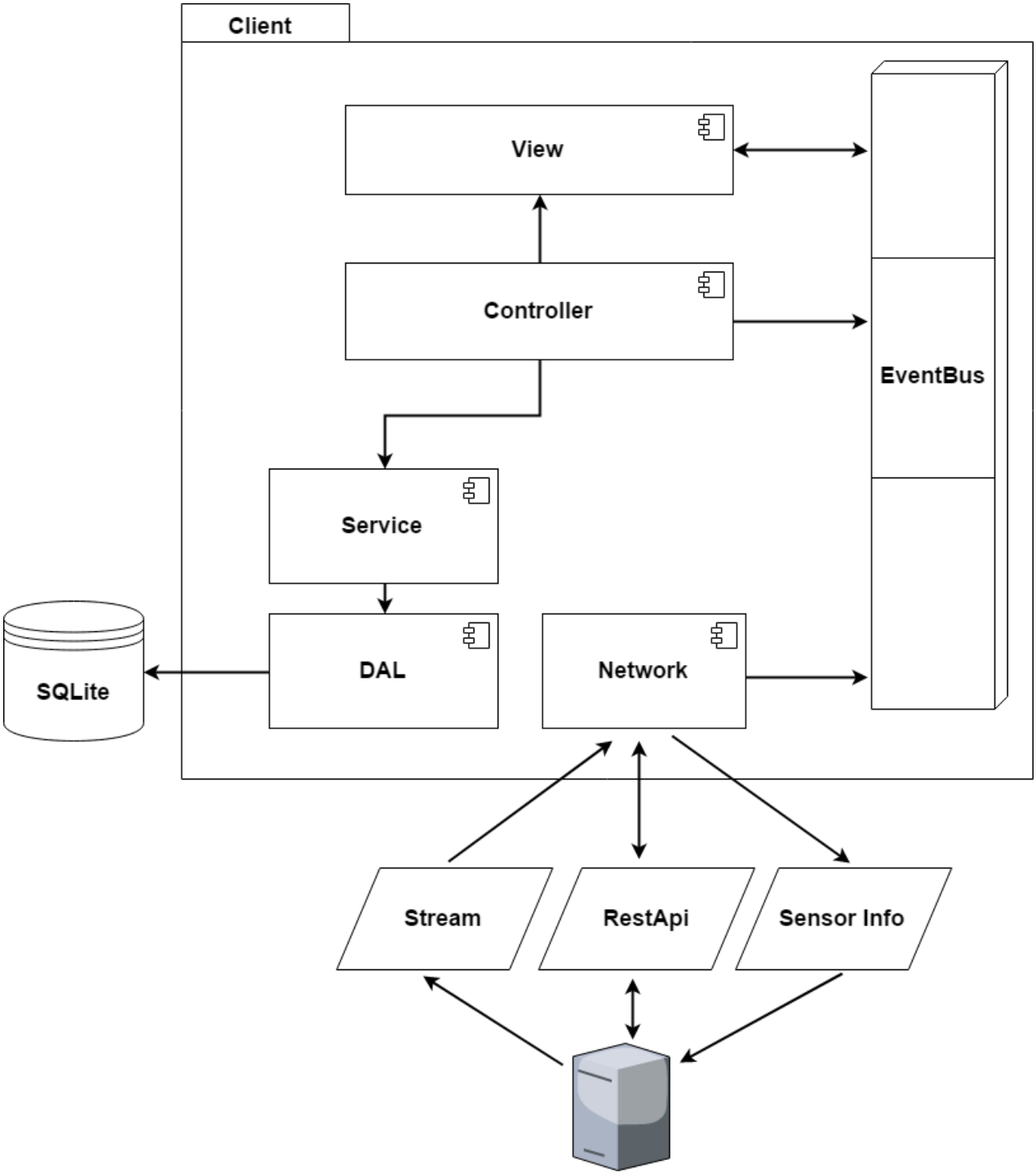
3.2. ábra. Szerver architektúra

A szerver három független fő komponensből áll: a számítógép grafikus munkafelületét elérő és közvetítő szolgáltatás (frame service), a hálózati funkcionalitásokat ellátó szolgáltatás (network service) valamint az OpenVR driver implementációt biztosító komponens.

3.2.2. Kliens

A VRemote Android kliensalkalmazás architektúrája a 3.3 ábrán látható UML diagram segítségével kerül szemléltetésre.

A kliensalkalmazás Controller komponense egy Service komponensen keresztül használja az adathozáférési réteget (DAL). A DAL az előző VRemote szerver kapcsolatok adatainak elérésehez egy SQLite adatbázishoz kapcsolódik. A Network komponens feladata a szerveralkalmazással való hálózati kommunikáció megvalósítása. Az általa ellátott feladatok a videó közvetítés fogadása, a RestApi kommunikáció valamint a telefon orientációs szenzorjából érkező adatok továbbítása. A Controller kommunikációját a View és a Network komponensekkel az EventBus segíti.



3.3. ábra. Kliens architektúra

4. fejezet

Fejlesztési módszerek és eszközök

Az alábbi fejezetben a fejlesztés során használt módszerek és eszközök részéletezésére kerül sor.

4.1. CMake

A VRemote szerverkomponens a CMake[3] automatikus projektgenerátor segítségével készíti elő az alkalmazást fejlesztésre, illetve fordításra.

A CMake egy nyílt forráskódú, cross-platform eszköz, alkalmazások építésére és csomagolására. Egyszerű konfigurációs fájlok segítségével (CMakeLists.txt) standard projektfájlok generálhatóak. A CMake által megadhatóak a külső fejlécek útvonalai, és lehetőség nyílik a külső könyvtárak linkelésére is. Előnye, hogy támogatja a komplex hierarchia-rendszerrel és több – akár statikus, akár dinamikus – függőséggel rendelkező projektek építését. Számos beállítási lehetőséggel rendelkezik, mint például az elkészül állományok helye, a generált projekt neve, a build opciói stb.

A szerveralkalmazás két fő mappába került rendszerezésre, a VRemote könyvtár tartalmazza a forrásállományokat, valamint a libraries nevű mappa, melynek elemei a porjekt során felhasznált külső függőségek. A VRemote szerveralkalmazás C++ nyelvben megírt forráskódjai a CMake által egy kiválasztott projektformátumba generálhatóak ki az operációs rendszertől függően. Windows rendszeren például egy 64 bites Visual Studio projekt míg Unix alapú rendszereken egy make fájl segítségével történhet a szerver buildelése.

Mivel a szerver a számítógép munkafelületének képét a DirectX segítségével éri el, csak Windows operációs rendszeren futtatható le. A képkinyerést biztosító szolgáltatás egy olyan komponensre történő lecserélése által, amely kompatibilis más rendszerekkel is, feloldható a Windows operációs rendszertől való függőség. Ekkor a VRemote szerveralkalmazása a CMake segítségével bármilyen projektformátumba kigenerálható.

4.2. Gradle

A mobilkliens fejlesztése során Gradle[4] build-rendszer került használatra. A Gradle egy olyan eszköz mely segítségével általános célú, Java technológiákon alapuló alkalmazások készítésére van lehetőség. Az Android operációs rendszerre történő applikációfejlesztés során elterjedt build-rendszer nagy előnye az erős függőségkezelő mechanizmusa, támogatást nyújt a már meglevő Maven tárházakhoz. A

Gradle build eszköztár beépített nyelve a Groovy, egy olyan deklaratív nyelv amely segítségével átlátható és könnyen értelmezhető scriptek készíthetőek.

Az Android Studio erős Gradle támogatással rendelkezik és a Google is a fent említett build-rendszert ajánlja Android eszközre történő fejlesztés során, ezért indokolt használata a VRemote mobilkliens esetében is.

4.3. Fejlesztői környezetek

A szerveralkalmazás fejlesztése során használt Visual Studio egy integrált fejlesztői környezet, amely segítséget nyújt a kódírásban C++ nyelvben is. Segítségével egyszerűvé válik a kódbázisban való navigálás, alkalmas gyors hibajavításra valamint mellékinformációk megjelenítésére. Számos konfiguráció szerkesztésére ad lehetőséget, de legnagyobb előnye a hibakereséshez elengedhetetlen eszközök biztosítása. A környezet debug kellékei, a futási idő alatt fellépő hibák azonosítására alkalmasak.

A mobilkliens fejlesztése az Android Studio segítségével történt, amely szintén egy integrált fejlesztői környezet, ez esetben az Android alkalmazás buildelésére. A környezet által biztosított gyors és funkcionalitás-gazdag emulátor segítségével az alkalmazás futtatható lokális számítógépen, ugyanakkor lehetőség nyílik az applikáció futtatására az Android operációs rendszerrel rendelkező eszközön is.

4.4. Verziókövetés

A fejlesztés során alkalmazott Git egy nyílt forráskódú, osztott verzió követő rendszer amely tervezését tekintve egyaránt használható kis és óriás projektek esetében is. Az osztott verziókövetők nem egy centrális tárhelyen tárolják a projekt teljes állomány készletét, hanem minden fejlesztő egy saját teljes másolattal rendelkezik. Ennek egyik előnye, hogy a munka lokálisan is végezhető, a központi kapcsolat hiányában is, majd a tárhely rendelkezésre állásakor egy szinkronizáció következhet. Másik előnye az adatvesztés elkerülése, mivel a másolatokból visszaállítható a teljes projekt.

A Git előnye a gyors művelet-végrehajtás, az adatok biztonságos tárolása és a több ágon történő munkafolyamatok menedzselése. A git tárhelyek manipulálása a GitLab webes felület segítségével történt. A GitLab egy olyan eszköz amely segítségével megvalósítható a projektfeladatok követése, felületet biztosít a felülvizsgálat (review) módszereinek, lehetőséget nyújt dokumentáció tárolásához és folytonos integráció megvalósítására is alkalmas a GitLab-CI által. Lokális számítógépen a git interakciók elvégzéséhez (commit, push, pull stb.) a fejlesztő csapat SourceTree nevű klienst használta.

4.5. Fejlesztési folyamat

A Scrum egy iteratív és inkrementáló agilis szoftverfejlesztés során alkalmazható keretrendszer, amely a VRemote projekt fejlesztés-menedzsmentjének alapjául szolgált. Az alkalmazás fejlesztése a keretrendszerből átvett elemek, ceremóniák segítségével történt. A projekt időkerete két hetes sprintekre volt osztva, melynek kezdőceremóniája egy tervező (planning) gyűlés volt, amely során különböző feladatok

dokumentálására került sor. A napi 15-30 perces scrum gyűlések fő célja az aznapi tevékenység leírása, valamint az esetleges problémák, akadályok megosztása a fejlesztő csapat többi tagjával. A sprint egy bemutatóval zárult, amely keretében bemutatásra kerültek a két hét alatt befejezett feladatok.

A projekt során több git repository is létrehozásra került, ezek közül a dolgozatban a a szerver és a kliens tárhelyének leírása jelenik meg, amelyek a gitflow előírásait követték: egy master ág, egy develop ág és minden új funkcionalitás fejlesztése során új feature ág, amelyek a kivizsgálást követően a stabil verzióval rendelkező develop ággal került összefésülésre.

A folytonos integráció megvalósítása GitLab-CI segítségével történt. Mivel a VRemote projekt célja a lokális hálózatban történő közvetítés, a szerver esetében minden interakció (commit, merge) során az alkalmazás build folyamatára kerül sor, amely eredményéről értesülnek a fejlesztők, így kiszűrhetőek a fordítás idejében keletkező hibák. A kliensalkalmazás esetében is a szerverhez hasonlóan, minden interakció a build folyamat elindulását eredményezi, de ez esetben létrejön egy letölthető .apk fájl is.

5. fejezet

Felhasznált technológiák és függvény-könyvtárak

A kutatási fázis során több technológia kipróbálásra és közelebbi megvizsgálásra került. A következő fejezetekben a felhasznált technológiák és függvény-könyvtárak kerülnek bemutatásra. A leírásban a kiválasztott eszközök melletti döntés alapos indoklása is megtalálható.

5.1. DirectX API

A szerverkomponens fő funkcionalitásai közé tartozik a számítógép teljes képének kinyerése a grafikus kártyáról. A fent említett feladat megoldása érdekében használatra került a Microsoft által implementált DirectX[5] API. A Windows 8 operációs rendszer által hozott frissítések között szerepel a desktop duplication API (továbbiakban DDA) is, amely lehetőséget nyújt a videokártya kimenetének virtuális megduplázására. Hatására az alkalmazás képes elérni – közel valós időben – a képernyő teljes képét frame-ekre bontva.

A képkinyerés során az adatok tárolása egy kétdimenziós textúrába történik, melynek mérete képernyő beállítástól függ. A textúra pixelek gyűjteménye, amelyek a legkisebb egységét jelentik a képnek, és állhatnak legalább egy, legfeljebb négy komponensből. A DDA által szolgáltatott kép pixelformátuma RGBA, egy 32 bites 4 komponensből álló formátum, amely összetétele a három színkomponensből – piros, zöld, kék –, valamint egy alfa komponensből –áttetszőség– áll.

5.2. FFmpeg

A késés csökkentése érdekében, a VRemote szerveralkalmazás élő közvetítés formájában továbbítja a számítógép munkafelületének képét. Ennek megvalósítása az FFmpeg keretrendszer segítségével történt. A Fast Forward Motion Pictures Expert Group által fejlesztett FFmpeg[6] egy nagy teljesítményű cross platform keretrendszer, amely képes számos más funkcionalitás mellett a videó kódolására, közvetítésére és lejátszására. Rengeteg ismert projekt született meg az FFmpeg keretrendszer segítségével, mint például a VLC Media Player, amely programozóknak szánt könyvtára (libvlc) a VRemote projekt kezdeti fázisában tesztelésre került, de nem bizonyult megfelelőnek a projekt célját figyelembe véve.

Az FFmpeg több alkönyvtárból áll, mint a libavcodec, amelynek szerves részét képezik a közismert videó codec-ek, mint például AVI, MPEG stb. A codec szó az angol en**co**de és **dec**ode fogalmakból ered, ugyanis a videóközvetítés előkészítése során tömörítés alkalmazása szükséges a hálózati forgalom csökkentése érdekében, míg a tartalom lejátszása csak kitömörítés után lehetséges. Egyik lehetséges videoformátum az MPEG, amely szintén a fent említett szervezet fejlesztése alatt áll, a formátum által alkalmazott tömörítő algoritmus a H.264. Mivel az MPEG videoformátum rendelkezik lezáró metainformációkkal így közvetítése nem lehetséges, ezért volt szükséges az MPEG-TS bevezetésére – szintén a fent említett szervezet által –, amely már nem tartalmaz lezáró információkat, ezért megfelel a VRemote szerver közvetítő funkcionalitásának.

Szintén az FFmpeg része a libavformat könyvtár, amely segítségével megvalósítható a élő közvetítés. A könyvtár különböző kimeneti forrásokat támogat, mint például a videó fájlba mentése vagy hálózaton történő továbbítása UDP (User Datagramm Protocol) vagy RTP (Real-Time Protocol) segítségével. A stream előkészítése során az libavformat könyvtár lehetőséget nyújt különböző konfigurációk módosítására, mint például konstans FPS (frame per sec), bit-sűrűség, videoméret, pixelformátum stb.

5.3. Poco könyvtár

A Poco[7] könyvtár egy nyílt forráskódú C++ eszköztár, amely komplex hálózati funkcionalitásokkal ellátott rendszerek kivitelezésére összpontosul, használatával leegyszerűsíthető és felgyorsítható az olyan hálózat-központosított alkalmazások fejlesztése, mint a VRemote. Más nagyobb keretrendszerekhez hasonlóan a Poco is több alkönyvtárból áll, melynek mageleme a Foundation csomag, ami által többek között megvalósítható a kommunikáció két azonos rendszeren futó folyamat között, osztott memória révén. A Net és a Util csomagok szintén részét képezik a Poco könyvtárnak, használatuk megkönnyíti az UDP és multicast socket-ek létrehozását – amely segítségével információ továbbítható egy IP csoportnak – , valamint a RESTful webszolgáltatások implementálását is, melynek része a JSON formátum dekódolása is.

5.4. OpenVR

Az OpenVR[17] a Valve által kifejlesztett keretrendszer. Felhasználásával a VRemote képes a Steam platformon futó VR-t támogató játékok irányítására.

Felhasználásával a különböző VR eszközök készítői saját implementációt valásíthatnak meg. A keretrendszert absztrakt osztályok alkotják, melyek tisztán virtuális függvényeket tartalmaznak. Egy saját driver implementálásához a rendelkezésre álló megfelelő interfészek saját megvalósítása szükséges. Egy ilyen megvalósítás lehetőséget nyújt a Steam platformon futó VR játékok irányítására és megjelenítésére valamely külső hardware készüléken keresztül.

5.5. OrmLite

Az OrmlLite[11] nyílt forráskódú ORM keretrendszer Java objektumok egyszerű módon történő leképezésére valamely SQL adatbázisba. A keretrendszer felhasználása megkönnyíti a kapcsolati információkat tartalmazó objektumok perzisztensé tevését a lokális SQLite adatbázisba. Használata nagyobb projektek esetén a leghasznosabb, azonban az alkalmazás továbbfejlesztésére gondolva előnyösnek bizonyulhat, a gyors bővíthetőségi lehetőség miatt.

5.6. EventBus

Az EventBus[13] Androidra optimalizált keretrendszer mely a különböző komponensek közötti kommunikáció egyszerűsítését valósítja meg. Az event bus feliratkozás alapú eseménykezelő rendszer. A keretrendszer használatához események definiálása szükséges, melyek egyszerű java osztályok. A figyelőként feliratkozott komponensek az event buson keresztül lesznek értesítve a más komponensek által kiváltott megfelelő eseményekre.

5.7. Dagger

A dependency injection módszert használva az objektumok közötti merev függőségek kielégitisét egy külső objektum végzi, melynek feladata a különböző függőségek injektálása. A Dagger[12] egy dependency injection keretrendszer mely Java-hoz és Android-hoz is egyaránt használható.

6. fejezet

Az Android kliensalkalmazás fejlesztésének részletei

A továbbiakban ismertetésre kerülnek a VRemote kliensalkalmazásának fejlesztési részletei.

6.1. Android

Az Android[8][9] napjaink legelterjedtebb mobil operációs rendszere melynek terjedése még mindig széleskörűen mutatkozik. Ez a tény indokolta, hogy a mobil alkalmazás fejlesztése először az Android platformra történjen.

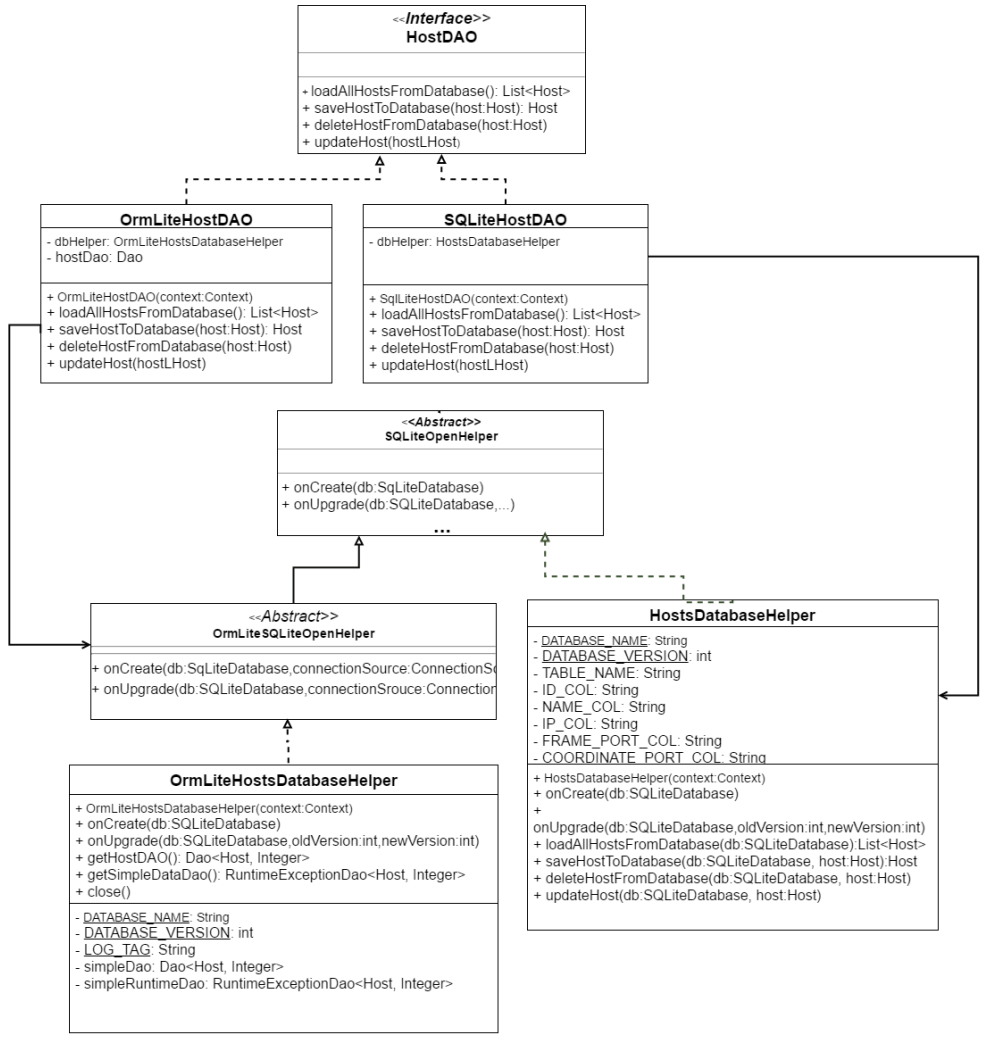
Az Android platform az Android Open Source Project (AOSP) keretén belül fejlődik, alapját a Linux kernel képezi. Ennek következtében Androidhoz elengedhetetlenül fontos minőségjelzők társíthatóak mint a biztonság, stabilitás vagy a megbízhatóság. Ezek segítségével elősegíthető a kellemes felhasználói tapasztalat.

6.2. Az adatok perzisztenciájának megvalósítása

A mobilalkalmazásnak hálózati adatokra van szüksége a szerveralkalmazáshoz való kapcsolódáshoz. Egy kapcsolat létrejöttéhez szükséges adatokhoz egy név is társítható. Az előző kapcsolatok elmentése és rendelkezésre bocsájtása jelentősen megkönnyíti az újrakapcsolódást. Ezért az alkalmazás ezeket lokálisan tárolja.

A 6.1 ábrán az alkalmazás adathozzáférési rétege van ábrázolva egy osztálydiagram segítségével. A HostService interfészt megvalósító HostServiceImlp osztály a HostDAO interfészen keresztül fér hozzá egy DAO objektumhoz.

Két osztály valósítja meg a HostDAO interface-t, az SQLiteHostDao és az OrmLiteHostDAO. A két megvalósítás két különböző SQLiteOpenHelper absztrakt osztályból származtatott segéd osztályokkal dolgozik, melyek az adatbázis műveletek végrehajtásában segítenek. Kezdtetben az adatbázisműveletek hagyományos SQL lekérdezésekkel kerültek végrehajtásra, a HostDatabaseHelper osztály segítségével mely az SLiteOpenHelper-ből lett származtatva.



6.1. ábra. Data access layer osztálydiagram

Az ORMLite keretrendszer használatához, az SQLiteOpenHelper osztály lett létrehozva. Az osztály az OrmLiteSQLiteOpenHelper-ből lett származotatva mely a maga során az SQLiteOpenHelper közvetlen leszármazottja.

6.2.1. Lokális adatbázis és a felületen levő adatok szinkronizálása

A lokális adatbázis és a felületen levő adatok szinkronizációs problémáinak elkerülése érdekében minden adatokkal kapcsolatos művelet eredménye előbb az adatbázisban kerül leképezésre, majd a felület az adatbázisból kinyert adatokkal lesz frissítésre.

6.2.2. Dependency injection Daggerrel

A HostController osztály a HostService interface-en keresztül fér hozzá a service osztályhoz. A kontroller a service osztály segítségével a kapcsolatokra vonatkozó műveleteket végzi. A HostService interface megvalósítása az Androidos Dagger dependency injection framework segítségével lesz injektálva.

Ennek megvalósítása érdekében a HostService títpusú adattag egy @Inject annotációval van felruházva. A tulajdonképpeni injektálás elvégzéséhez egy @Module-al felannotált osztály elkészítése szükséges. Az annotációnak az injects nevű paraméterében azok az osztályok szerepelnek amelyek függőségeit ki kell elégíteni.

```
@Module(
injects = HostController.class
)
```

A modul osztály @Provides annotációval ellátot metódusai térítik vissza az injektált objektumokat. A metódusok visszatérítési értékeinek a típúsa mutatja a kielégítendő függőséget.

```
@Provides HostService provideHostService(@InjectAndroidApplicationContext Context context){
        return new HostServiceImp(new OrmLiteHostDAO(context));
}
```

A HostService függősége megoldására egy HostServiceImp példánnyal lesz injektálva. A kiválasztott HostDAO objektum, a HostServicet implementlő osztály konstruktorában van átadva, ez egy OrmLiteHostDao objektum. Az @Inject és a @Provide annotációkkal ellátott objektumok egy ObjectGraph-ot alkotnak. Az ObjectGraph az injektálható valamint a függőségeket tartalmazó objektumok gyűjteményéből álló gráf. Az ObjectGraph az Application-ből származtatott VrApplication osztály onCreate metódusában van létrehozva az elkészített modul segítségével. A HostController osztály példánya az így létrehozott ObjectGraph segítségével lesz a függőségeivel injektálva.

6.3. Grafikus felület felépítése

Az alkalmazás három különböző Activityből áll. Az indításakor betöltöltődő nyitókép mely a felhasználót fogadja a SplashActivityben található. A SplashAvtivity betölti a HomeActivityt majd bezáródik, mivel célja a HomeActivity betöltődéséig várakoztatnia a felhasználót. A HomeActivity-ben egy TabHost konténerhez két Fragment van hozzáadva, a többpaneles felhasználói felület kialakítása érdekében.

Az első Fragment a HostsTab, melyben a régi kapcsolatok szerepelnek. A HostsTabon egy RecycleViewhoz a HostsAdapter van beállítva adapterként. Az Adapterben található elemek listája a MyViewHolder példányaival lesz feltöltve, ezek a kapcsolati információkat tartalmazzák. Az második fragment a NewHostTab, melyben új kapcsolatok létrehozására van lehetőség.

A harmadik Activity a StreamViewerActivity mely a stream megjelenítéséért felel. A közepén elhelyezett gomb a közvetítés megzavarásának elkerülése érdekében bármikor eltüntethető. A gomb segítségével az irányítási funkcionalitás elindítható vagy megállítható.

6.4. Nézetek közötti kommunikáció (EventBus)

A GeneralController osztály felelős a felhasználótól érkező adatok fogadásáért, valamint a nézetek aktualizálásáért. A nézetek és a GeneralController közötti kommunikáció event bus segítségével történik. Ez a megvalósítás a kétoldali kommunikációra is lehetőséget nyújt. Bizonyos eseményekre feliratkozó osztályok bejegyezik magukat figyelőként.

```
EventBus.getDefault().register(this);
```

Egy esemény kiváltása a következőképpen kerül megvalósításra. Az EventBus osztálynak egyetlen létező példányát a getDefault statikus metódusa téríti vissza, majd a post metódus segítségével kiváltható a megfelelő esemény.

```
EventBus.getDefault().post(new ConnectionFailedEvent());
```

Az eseményre feliratkozott összes létező objektum értesítve lesz az éppen kiváltott eseményről.

```
@Subscribe
public void onEvent(CancelConnectTaskEvent event) {
        ...
}
```

6.5. A stream lejátszása

A projekt kezdeti szakaszában a megjelenítés több, jól elkülöníthető fázisban történt. A frame-ek egyenként voltak fogadva TCP socketen keresztül, majd ezek kitömörítését követően következett csak a megjelenítés.

Az élő közvetítés választását követően, a szerveralmalmazás által közvetített stream az FFmpegMediaPlayer[7] felhasználásával kerül lejátszásra, mely az Androidos MediaPlayer egy megvalósítása. Mivel az Android network protokoljai között nem szerepel natívan az UDP, ezért a beépített MediaPlayer nem képes a natív UDP stream megjelenítésére. Használatával nem csak a támogatott protokollok, hanem a formátumok köre is szélesebb. Az FFmpegMediaPlayer a megjelítéshez egy **Surface** objektumot használ. Ezt egy **SurfaceView** elemtől elkért **SurfaceHolder** szolgáltatja.

6.6. Az irányítás megvalósítása

A projekt kezdetén egy desktop egérirányítás vot megvalósítva. A funkcionalitás kivitelezéséhez a telefon giroszkópjából érkező adatok megfelő átalakítására volt szükség. A legfőbb prioritást a számítógépek felhasználói felületének felbontásához való alkalmozkodás jelentette. Ebből kifolyólag az egér kurzorjának aktuális helyzete nem konkrét helyhez kötött pozícióval volt meghatározva. Ehelyett relatívan, a kijelző bal felső sarkához viszonyítva kerültek reprezentálásra, százalékban kifejezve. A hálózati adatforgalom lecsökkentése érdekében a nagyon kicsi eltérés, mely egy nem észlelhető egérmozgást eredményezne, feleslegesnek tekinthető. Ezáltal egy alap navigálásra alkalmas irányítás lett megvalósítva.

Későbbi követelmény a szerveralkalmazással azonos gépen futó játék irányítása lett, a fej mozgását figyelembe véve. Ennek megvalósítása érdekében a telefon orientációs szenzorából[15] kinyert adatok vannak felhasználva.

```
mSensorManager.registerListener(mSensorListener, mSensorManager.getDefaultSensor(Sensor.
    TYPE_ORIENTATION), SensorManager.SENSOR_DELAY_FASTEST);
```

A figyelő objektum bejegyzésekor a szenzor adatok maximálisan megengedett késési idejét a harmadik paraméter jelöli. A **SensorManager.SENSOR_DELAY_FASTEST** konstans megadásával a megendett késés értéke 0. Az Ox, Oy, Oz tengely körüli elfordulási szögekkel egy radiánba való alakítás van elvégezve. A telefon alaphelyzete miatt az Oy menti szög 180 fokból kivonva fog a megfelelő értékkel rendelkezni. A helyes szögek ezt követően UDP csomagokban lesznek továbbítva. Kezdetben a kliensalkalmazás kvaternióvá alakította, majd a négy floatot küldte hálózaton. A kvaternióba történő alakítást a szerveroldalra helyezve eltávolítható a kliens oldali plusz komplexitás. Mellesleg ezáltal négy érték helyett csupán három értéket kell a hálózaton átküldeni.

A kvaterniók négy dimenziós komplex számok, általában a következő alakúak: w + x*i + y*j + z*k. Segítségükkel orientációk és forgatások írhatóak le a három dimenziós térben. A w valós komponens, az elforgatási szöget jelöli a kvaternió tengelyei körül.

Az euler szögek kvaterniókká való alakítása a következő algoritmus segítségével valósítható meg, melyben a yaw, roll, pitch a z,x valamint az y tengely körüli elforgatási szögeket jelölik.

```
        float cosyaw = cos(yaw * 0.5);
        float cosroll = cos(roll * 0.5);
        float cospitch = cos(pitch * 0.5);

        float sinyaw = sin(yaw * 0.5);
        float sinroll = sin(roll * 0.5);
        float sinpitch = sin(pitch * 0.5);

        float cosyaw_cospitch = cosyaw * cospitch;
        float syaw_sinpitch = sinyaw * sinpitch;
        float cosyaw_sinpitch = cosyaw * sinpitch;
        float sinyaw_cospitch = sinyaw * cospitch;

        float w = cosyaw_cospitch * cosroll + sinyaw_sinpitch * sinroll;
        float x = cosyaw_sinpitch * cosroll + sinyaw_cospitch * sinroll;
        float y = sinyaw_cospitch * cosroll - cosyaw_sinpitch * sinroll;
        float z = cosyaw_cospitch * sinroll - sinyaw_sinpitch * cosroll
```

A szerveralkalmazás külön végrehajtási szálon hallgatja az UDP socketen beérkező szögek radiánban kifejezett értékeit. Ezen értékek kvaterniókká alakítása után és egy x tengely körüli forgatást követően lesznek az osztott memórián keresztül továbbítva a SteamVR driver számára.

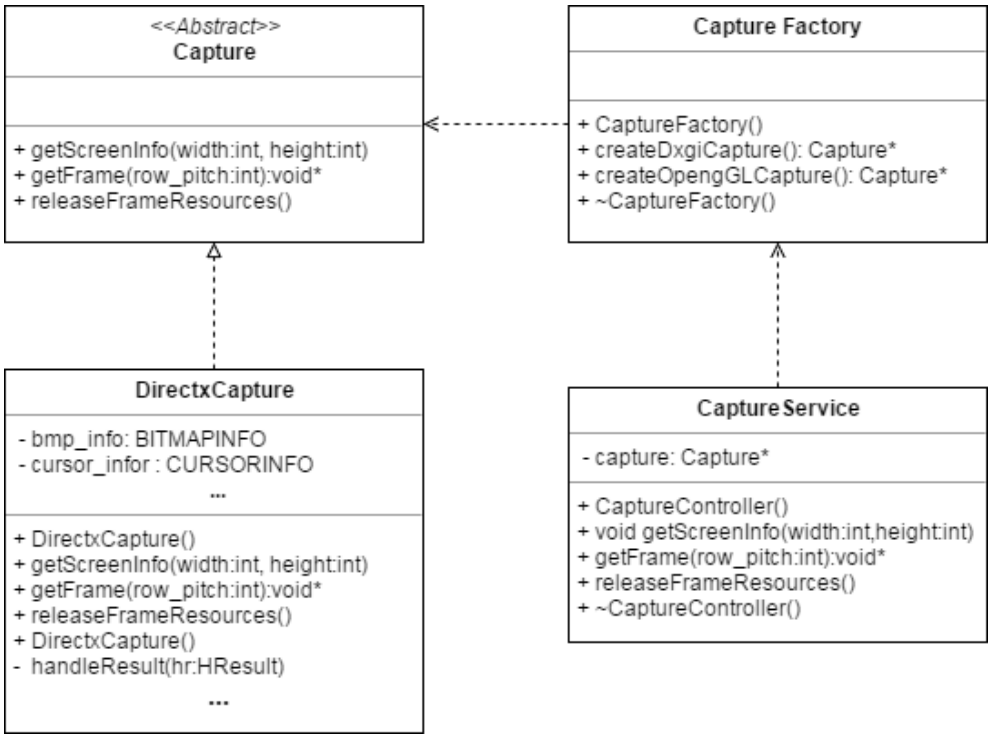
7. fejezet

Szerver alkalmazás fejlesztésének részletei

A továbbiakban ismertetésre kerülnek a VRemote szerver alkalmazásának fejlesztési részletei — ez a szerver egy C++ nyelvben írt szolgáltatás alapú rendszer.

7.1. Képernyő teljes képének kinyerési módja

A VRemote projekt első fázisában különböző képkinyerési mechanizmusok kerültek tesztelésre, mint a Windows BitBlt funkcionalitása, vagy az OpenGL által nyújtott lehetőségek. A szerverkomponens a kinyerést biztosító szolgáltatása a technológiáktól való függetlenség érdekében absztrakt gyár tervezési mintát követ, amint az a 7.1 ábrán látható. A szolgáltatást használni kívánt komponensek elől rejtett az implementáció, és szükség esetén bármikor kicserélhető egy másik megoldásra, így az alkalmazás akár más operációs rendszeren is képes működni, a képkinyerő szolgáltatás helyettesítése által.



7.1. ábra. Képkinyerő szolgáltatás osztálydiagram

A desktop duplication API inicializálásának első lépése az úgynevezett device virtuális adapter létrehozása, amely főbb funkcionalitásai a kijelző paramétereinek elérése, illetve a különböző erőforrások menedzselése. Segítségével létrehozható egy olyan külső kijelzőt figyelő adapter, amely képes megduplázni az integrált videokártya kimenetét, és ezáltal az alkalmazásnak hozzáférése van a számítógép teljes grafikus felületének képéhez.

```
// Query output
IDXGIOutput* dxgi_output;
hr = dxgi_adapter->EnumOutputs(Output, &dxgi_output);

...

// Query the first output
IDXGIOutput1* dxgi_output1;
hr = dxgi_output->QueryInterface(IID_PPV_ARGS(&dxgi_output1));

...

//Create desc duplicaton which we can acquire frames from desktop
hr = dxgi_output1->DuplicateOutput(device, &desk_dupl);
```

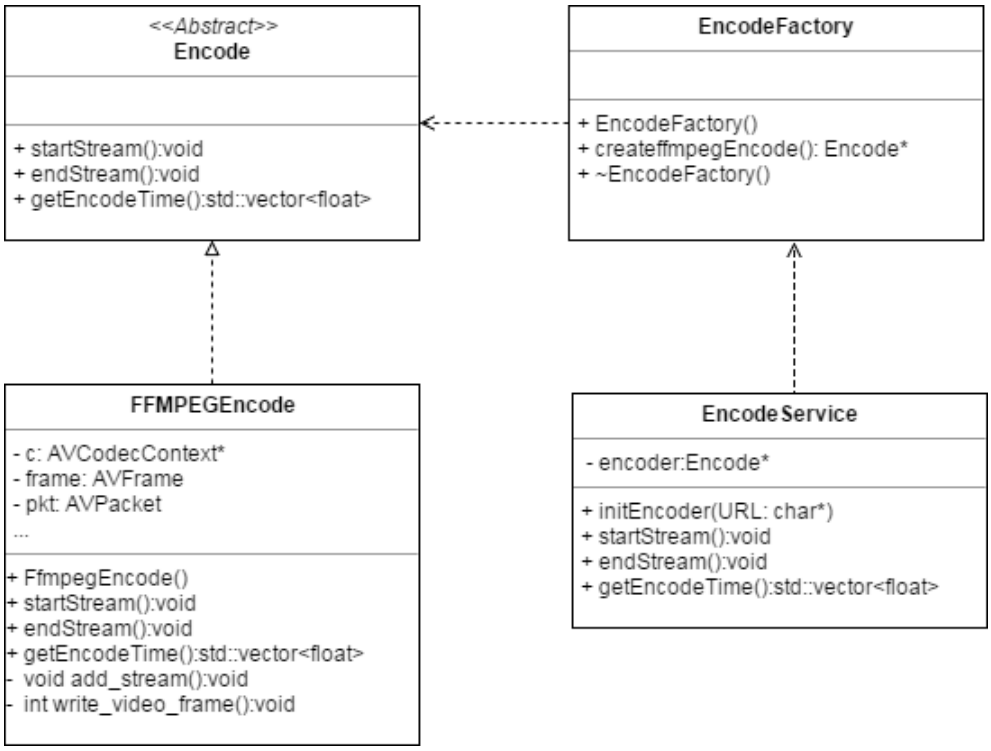
A megduplázott kimenet segítségével kinyerhetőek a videokártya kimenetre szánt tárolójából az aktuális adatok, amelyet elsőként csak mint ismeretlen erőforrást tud kezelni az alkalmazás, a tényleges adat eléréséhez szükséges átmásolni egy olyan textúrába, amelyet a központi feldolgozóegység is elér, nem csak a grafikus kártya.

```
//Acquire next frame with infinite timeout to desktop_resource
hr = desk_dupl->AcquireNextFrame(INFINITE, &frame_info, &acquired_desktop_image);

...


// Copy image into GPU access texture
immediate_context->CopyResource(gpu_texture, acquired_desktop_image);

...


// Copy image into CPU access texture
immediate_context->CopyResource(cpu_texture, gpu_texture);

// Copy from CPU access texture to bitmap buffer
immediate_context->Map(cpu_texture, subresource, D3D11_MAP_READ_WRITE, 0, &resource);
```

A textúrák lefoglalását, illetve létrehozását a device adapter végzi, viszont a kinyert textúra formátuma minden esetben RGBA, a mérete az aktuálisan beállított videokártya felbontásától függ. A kinyert textúra még nem tartalmaz kurzoradatokat, a Windows által biztosított metódusok segítségével kérhető le a kurzor aktuális pozíciója, amelyet utólag rárajzol az alkalmazás. A textúrában tárolt kép eléréshez map alkalmazására van szükség, melynek algoritmusa szintén biztosítva van a device osztály által. Az így felszínre hozott adatok már készen áll a videokódolásra.

7.2. A kép közvetítése

A projekt során számos képküldési lehetőség tesztelésre került, ezért itt is indokolt volt az absztrakt gyár tervezési minta használata, mint ahogy láthatjuk a 7.2 ábrán. Legelső megoldásként a kinyert kép bájt adatainak továbbítása került megvalósításra, különböző tömörítő könyvtárak, mint az LZ4 vagy a ZLib veszteség mentes algoritmusait alkalmazva, de a küldeni kívánt adat mérete miatt nem sikerült elérni az élvezhető kép/másodperc (FPS) rátát.



7.2. ábra. Kódoló szolgáltatás osztálydiagram

A megteremtett virtuális valóság élvezhetőségét tekintve szükség volt bevezetni egy gyorsabb megoldást, amely által biztosított egy konstans FPS, ugyanakkor a valós idejű kép megjelenítéséért elengedhetetlen volt a kép késésnek csökkentése. Ezért a fejlesztés későbbi szakaszában bevezetésre került az élő videoközvetítés alkalmazása, amely során több könyvtár tesztelésre került, mint a libvlc vagy a live555, de végül az FFmpeg könyvtár használata volt a legindokoltabb.

A videokódolás első lépéseként szükség van a kiválasztott codec betöltésére – a VRemote projekt esetében mpeg-ts –, amely beállításai manuálisan konfigurálhatóak, mint például bit sűrűség, szélesség, magasság. A H.264 tömörítő algoritmus bemenete csak YUV formátummal rendelkező kép lehet, ezért a tömörítés megkezdése előtt szükség volt a kinyert RGBA kép transzformációjára.

```
//Prepare converter
if (!video_st.sws_ctx) {
        video_st.sws_ctx = sws_getCachedContext(video_st.sws_ctx,
                width, height, AV_PIX_FMT_RGB32,
                width, height, AV_PIX_FMT_YUV420P,
                SCALE_FLAGS, NULL, NULL, NULL);
        if (!video_st.sws_ctx) {
                throw FfmpegException("Could not initialize the conversion context\n");
        }
}

//Convert RGBA byte array to yuv420p
sws_scale(video_st.sws_ctx, (const uint8_t * const *)&act_frame, in_linesize, 0, height,
    video_st.frame->data, video_st.frame->linesize);
```

Az YUV[16] pixelformátum analóg tévé-átvitelre volt kifejlesztve, ezzel biztosítva kompatibilitást a színes és a fekete-fehér tévékészülékek között. A formátum az adott képet egy fényerő (luminance, Y) csatornára, valamint U és V színcsatornákra bontja, előnye a skálázhatóság; mivel az emberi szem érzékenyebb a fényhatásokra, mint a színekre, lehetőség van a színtömörítés alkalmazására. A YUV420

speciális pixelformátum során a színcsökkentés 4:2 arányban kerül alkalmazásra, a fényerőcsatorna minden négy eleméhez tartozik egy U elem, valamint egy V elem. Ez a tárolási módszer lehetőséget ad csökkenteni egy képkocka méretét olyan módon, hogy a felhasználó nem tapasztal minőségromlást.

A transzformált pixelformátummal rendelkező kép az FFmpeg által tömörítésre, majd továbbításra kerül, konstans 25 kép/másodperc aránnyal.

```
// encode the image
avcodec_encode_video2(video_st.enc, &pkt, video_st.frame, &got_packet);

...

// Write the compressed frame (pkt) to stream
av_interleaved_write_frame(oc, pkt);
```

Szükség esetén lehetőség van az elküldött kép skálázására gyengébb hálózat esetén, de ez minőségromlást eredményez. Ugyanekkor a szerver statisztikákat készít a képkinyerés, illetve a kódolás alatt eltelt idők átlagáról.

7.3. OpenVR driver implementáció

A megvalósíta során HmdDriverFactory függvény definíciójának elkészítése szükséges amely egy-egy implementálációt biztosít az interfacekre. A IVRWatchdogProvider interfészt megvalósító osztály a CWatchdogDriver_Sample melyet a vrclient.dll tölti be, az IServerTrackDeviceProvider interfészt a CServerDriver_Sample osztály implementálja melyet vrserver.exe fog felhasználni. A CSampleDeviceDriver megvalósítja az ITrackedDeviceServerDriver interfészt, rajta keresztül történik az irányítás.

A ControlInterface osztályból származtatott ControlInterfaceImpl osztály GetRotation metódusát használja fel a CSampleDeviceDriver osztály GetPose() metódusa. Ezen a metóduson keresztül fér hozzá az irányításos szükséges adatokhoz a SteamVR.

Az elkészített driver .dll kiterjesztéssel rendelkezik. A SteamVR almalmazás a drivereket a **steamapps\commonSteamVR\drivers** alkönyvtárából tölti be. A drivers.cfg állomány tartalmazza a driverek felsorolását. Az egyes driverek konfigurációs paraméteri a Steam config könyvtárának steamvr.vrsettings nevű állományában szerepelnek. Az egyébb betöltésre kerülő külső driverek, a **felhasználói könyvtár \Appdata\Local\openvr** útvonalán található openvrpath.vrpath nevű állományban jelenthetőek be.

7.4. REST szolgáltatások megvalósítása

A REST szolgáltatás megvalósításának alapját a Poco könyvtár biztosítja, a könyvtár szerkezete bővebb kifejtésre került a fejezet 5 fejezetben. Az aktív HTTP szerver fogadja, feldolgozza és választ készít a beérkezett kérésekre amelyeket az URI-k segítségével azonosít. REST hívások segítségével lehetőség van a közvetítés indítására, leállítására valamint a beállításainak szerkesztésére, illetve a szerver által elkészített statisztikák lekérésére. A fent említett szolgáltatás bevezetésére a felhasználói élmény növeléséért volt szükség, ugyanis így egyszerűbb a szerver konfigurációja mint a szerver paraméterezett indítása.

7.5. Szolgáltatás hirdetése lokális hálózatban

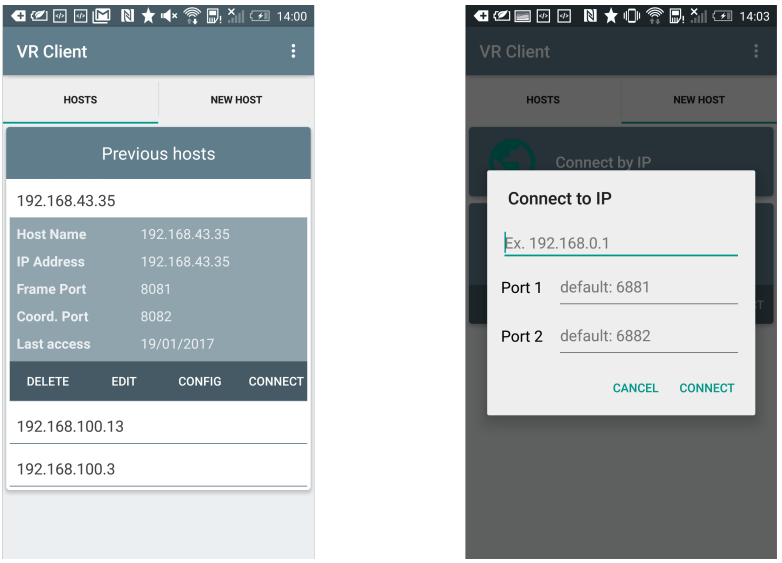
A multicast olyan egy a többhöz vagy több a többhöz kapcsolatot jelent amelyben az információ egy csoport számára van címezve. IP multicast esetén a végpontok csatlakozhatnak, illetve elhagyhatják a csoportot. Egy csomópont által kapott információ rendelkezik a küldő adataival (mint IP cím, port, név stb.). A VRemote szerver komponense az alkalmazás indulásakor csatlakozik egy IP csoporthoz, melynek a címe ismert a kliensalkalmazások által, és "VRemote online" üzenetek segítségével jelzi a csoport tagjainak jelenlétét amelyet bizonyos időközönként ismétel. Így azok a kliensek, amelyek az előre megegyezett IP csoport figyelői, automatikusan rácsatlakozhatnak a szerveralkalmazásra, specifikus információk hiányában is.

8. fejezet

A VRemote működése

Az alábbi fejezetben a VRemote működése kerül ismertetésre. A funckionalitások könnyebb bemutatása érdekében az alkalmazás megfelelő részeiről képek vannak társítva szemléltető anyagként.

A szerveralkalmazás, elindítása után, gondoskodik az IP címe és a beállított portok megjelenítéséről, melyen keresztül ő egyenesen elérhető. Ahogy a 8.1 ábrán is látható, a kliensalkalmazás NEW HOST fülön belüli Connect by IP gombja megnyomásával lehet egy új szerverre kapcsolódni, az IP címe ismeretében. A portok kitöltése már nem szükséges, mivel a kliens RestApi kéréseket intéz ezek kiderítésére.

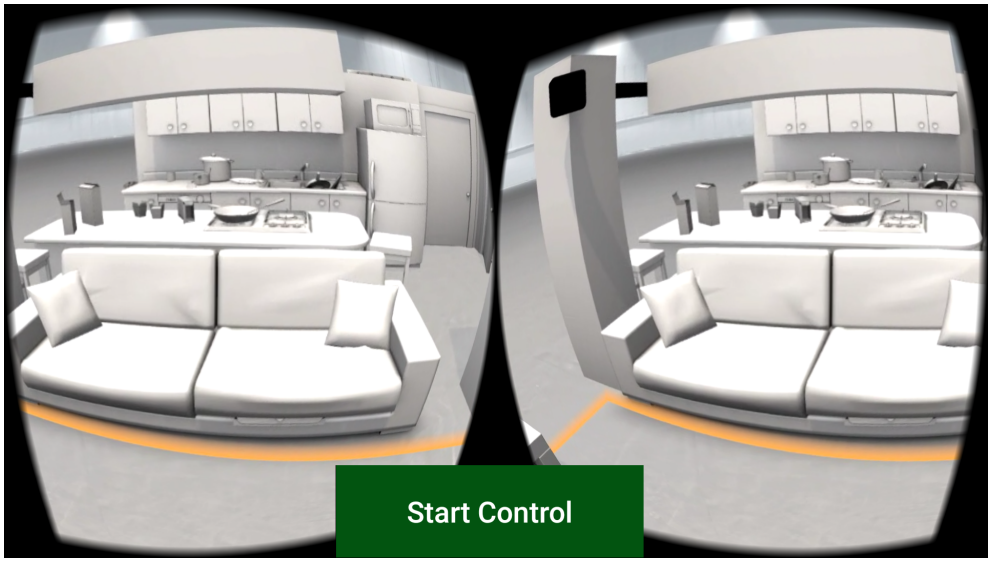


8.1. ábra. Android kliensalkalmazás - Kapcsolatok és kapcsolódás

Az előző, automatikusan elmentett kapcsolatok a HOSTS fülön belül vannak kilistázva, az újrakapcsolódás megkönnyítése érdekében. A szerveralkalmazás elérési paramétereiben valamilyen változás fellépésekor, a kapcsolatok adatai egyenként módosíthatóak. Törlésre is lehetőseg van, abban az esetben ha már nem lesz szükség egy kapcsolatra. A HOSTS fül szintén a 8.1 ábrán látható.

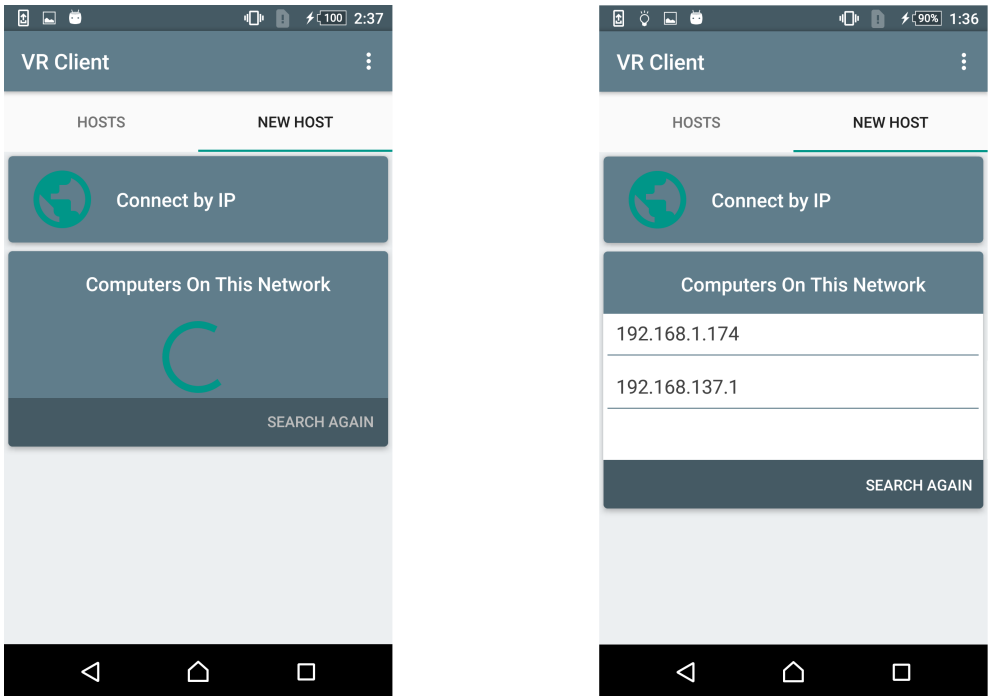
Egy sikeres kapcsolódást követően a kliensalmalmazás RestApi hívásokon keresztül közli a saját IP címét, majd kéri a videó közvetítés elindítását. A 8.2 ábrán látható egy VR módban futó játék a kliensalkalmazásban megjelenítve. Az irányítás ki- és bekapcsolására lehetőséget adó gomb bármikor eltüntet-

hető vagy megjeleníthető a telefon képernyőjének egyszeri érintésével.



8.2. ábra. Android kliensalkalmazás - Stream megjelenítése

A 8.3 ábrán a közvetítési szolgáltatást hírdető szerveralkalmazások listája. Ez a funkcionalitás a NEW HOST fülön belül érhető el. Egy automatikus keresés a fül kiválasztásakor elindul, de ez a SEARCH AGAIN gombbal megismételhető.



8.3. ábra. Android kliensalkalmazás - Kapcsolatok felfedezése

9. fejezet

Következtetések és továbbfejlesztési lehetőségek

A kitűzött célokat követve elkészült egy olyan szoftverrendszer amely segítségével a felhasználók képesek a virtuális valóság megtapasztalására. A drágább eszközökkel ellentétben a VRemote esetében egy könnyen és elérhető áron beszerezhető VR szemüvegre van szükség, amelybe egy okostelefon helyezhető. Ezáltal lehetőség nyílik a Steamen elérhető játékok kipróbálására, játszására, nagyobb mozgási területen, mivel a VRemote az elő közvetítést kábel nélküli lokális hálózatban valósítja meg.

A játékélmény akkor válik teljessé, ha a felhasználó képes interakciók elvégzésére is, ezért a fő továbbfejlesztési lehetőséget a kontrolleszközök bevezetése jelentené, amely segítségével a felhasználó műveletek elvégzésére lehetne képes az adott játékban. Az OpenVR biztosít eszközöket a kommunikáció megvalósítására a szerveralkalmazás és a kontrolleszközök között, ez egy megfelelő kiindulópontot jelenthet az ötlet kivitelezésében.

Egy második továbbfejlesztési irány a számítógép munkafelületének közvetítése virtuális valóság módban, ahol a különböző ablakok térben helyezkednek el, és a fejmozgatás segítségével történik az egér irányítása. Mivel az alkalmazás képes a DirectX API segítségével a számítógép teljes munkafelületének képét elérni, lehetőség adódna az így kinyert frame transzformációjára és az ablakok szerinti különválasztására, amelyet a felhasználó tetszés szerint rendezhet a kialakított virtuális térben.

Irodalomjegyzék

[1] Ivan E. Sutherland. *The Ultimate Display*. Proceedings of IFIP Congress, pp. 506-508, 1965.

[2] Google Cardboard,
https://vr.google.com/cardboard/

[3] CMake
https://cmake.org

[4] Gradle
https://gradle.org

[5] Justin Stenning *Direct3D Rendering Cookbook*. Birmingham-Mumbai
http://1.droppdf.com/files/lHuAM/packt-publishing-direct3d-rendering-cookbook-2014.pdf

[6] FFmpeg,
https://www.ffmpeg.org

[7] Poco,
https://pocoproject.org

[8] Android platform,
https://developer.android.com/guide/platform/index.html

[9] Android Interfaces and Architecture,
https://source.android.com/devices/

[10] About SQLite,
https://www.sqlite.org/about.html

[11] OrmLite,
http://ormlite.com/

[12] Dagger,
http://square.github.io/dagger/

[13] greenrobot EventBus,
http://greenrobot.org/eventbus/

[14] FFmpegMediaPlayer,
http://wseemann.github.io/FFmpegMediaPlayer/

[15] Android orientation sensor,
https://developer.android.com/reference/android/hardware/Sensor.html#STRING_TYPE_ORIENTATION

[16] Joe Maller. *RGB and YUV Color*.
http://joemaller.com/fcp/fxscript$_y uv_c olor.shtml$

[17] OpenVR Driver Documentation,

https://github.com/ValveSoftware/openvr/wiki/Driver-Documentation