Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnikai és Információs Rendszerek Tanszék

Lóska Ádám

AGY-SZÁMÍTÓGÉP INTERFÉSZEK INTEGRÁLÁSA VIRTUÁLIS VALÓSÁG RENDSZEREKBE

KONZULENS
Mészáros Tamás

BUDAPEST, 2012

Tartalom

1 Bevezető

Napjainkban egyre nagyobb figyelmet kapnak az agy-számítógép interfészek (BCI). Ez egy relatíve új technológia, számos ez idáig megvizsgálatlan lehetőséggel. Bár ipari vagy kereskedelmi alkalmazásuk még nem terjedt el, manapság már megfizethető áron elérhetőek ilyen eszközök, amik bár elmaradnak az orvosi műszerek színvonalától, mégis kutatási és fejlesztési célokra jól használhatóak. Az elterjedésük egyik komoly akadályának azt tartom, hogy számos alkalmazási körben vagy nem elég megbízhatóak még, vagy maga az alkalmazási környezet jelent olyan zavaró körülményeket, amik használhatatlanná teszik az ilyesfajta eszközöket.

Emiatt fordultam a virtuális valóságok (VR) felé. A VR rendszerek szintén új kutatási és fejlesztési terület, alkalmazásuk a szórakoztatóipartól kezdve a robotikáig számos területen megkezdődött. A mai virtuális valóság rendszerek már igen élethűek lehetnek, legtöbb esetben valósághű grafikával és fizikai szimulációval kiegészítve kellően jól szimulálják a valóság egy szeletét. Képesek arra, hogy a legfontosabb érzékszerveinket becsapják, és egy ingergazdag, immerzív környezetet teremtsenek, így a felhasználó szinte elhiszi, hogy a szimulált világban tartózkodik.

Ennek a két rohamosan fejlődő területnek a lehetőségeit szeretném ötvözni, hogy azok egymást kiegészítsék, támogassák. Elképzelésem alapja, hogy azokat a felhasználási területeket, ahol a BCI-ok alkalmazása még komoly akadályokba ütközne, reprodukáljuk VR rendszerekben, így egy szimulált szcenárió keretein belül fejleszthetjük és tesztelhetjük az algoritmusainkat, módszereinket. Ezáltal lehetőségünk nyílik olyan alkalmazási területeken is kipróbálni a BCI eszközöket, ahol korábban biztonsági, vagy egyéb okokból nem lehetett. Ilyenek lehetnek például a járműirányítás, ahol a nagy megbízhatóság és gyors válaszidő igen kritikus, vagy azok a területek, ahol korábban ezeket az eszközöket méretük, rossz hordozhatóságuk miatt nem alkalmazhattuk. Egyik célom tehát egy olyan környezet megteremtése, ahol a BCI technológiákat bármilyen alkalmazási terület szimulált változatában kipróbálhatjuk.

Továbbá úgy vélem, hogy egy ilyen integráció nem csak a BCI-ok számára jelent újabb lehetőségeket, hanem a virtuális valóságok számára is. Személyes tapasztalataim, és mások beszámolói alapján kijelenthetem, az immerzív, ingergazdag környezetek egyik hátránya, hogy bár érzékszerveinknek különösen kiterjedő élményt nyújtanak, a bemeneti eszközök terén sokkal kevesebb lehetőségünk kínálkozik. Egy olyan virtuális valóságban, ahol a nézeti irányt a fejtartás határozza meg, míg az avatár mozgatása egy ettől független vektor alapján történik, a konvencionális kontrollerek (joystick, billentyűzet, stb.) nem kielégítőek. Erre kívánok alternatívát ajánlani a BCI eszközök integrációjával.

Kutatások kimutatták, az immerzív környezetek pozitívan befolyásolják az agy-számítógép interfészek teljesítőképességét [1]. Emiatt voltak már próbálkozások BCI-ok VR rendszerekbe integrálására, néhány kutatóintézetben [11], vagy éppen piaci termékben [2] már alkalmazzák a módszert. Ezeknél a kísérleteknél azonban a VR rendszer csak, mint irányítani kívánt szoftver jelent meg. A felhasználó valamilyen BCI megoldással vezérelheti a virtuális valóságban való mozgását. Munkámban én egy ennél szorosabb, többrétű integrációt kívánok megvalósítani, ahol a BCI eszköz szerves része lehetne a VR rendszernek, és nem csak, mint egy új bemeneti eszköz használhatnánk, hanem a virtuális világ objektumait, funkcióit alapozhatnánk rájuk, kiterjesztve ezzel mindkét rendszer alkalmazhatóságát.

Munkám során tanulmányoztam a BCI-ok területén elért eddigi eredményeket, alkalmazott módszereket és a jelenlegi hardveres megoldásokat. Megismerkedtem az EEG jelek jellemzőivel, típusaival, és mérésének alapjaival.

Tanulmányoztam egy konkrét BCI eszköz, az Emotiv EPOC neuroheadset lehetőségeit, és alapos tesztelésnek vetettem alá, hogy alkalmazásának korlátait megállapítsam. Megismerkedtem továbbá egy VR rendszer, a VirCA Virtuális Kollaborációs Aréna felépítésével, az alkalmazott technológiáinak alapjaival, és komponenseinek fejlesztésével.

Megterveztem két megoldást a virtuális valóság, és annak objektumainak irányítására, és implementáltam őket. Az elkészült rendszereket teszteltem, felmértem alkalmazhatóságukat, problémáikat.

Megvizsgálva az elkészült alkalmazásokat megállapítottam, hogy a legnagyobb előrelépést a BCI eszköz motorjának lecserélésével érhetem el. Kiterjedt irodalomkutatást végeztem az EEG jelek BCI célra való feldolgozásának témakörében és megismerkedtem az alkalmazott paradigmákkal. A terület azonban igen jelentős neurobiológiai háttértudást és jelfeldolgozási ismeretet igényel, így célom egy olyan keretrendszer elkészítése lett, amit felkészítek arra, hogy a különféle jelfeldolgozó modulokat befogadja, az azok számára szükséges bemeneteket szolgáltassa, és az általuk előállított kimeneteket megfelelően kezelje. Megterveztem és kialakítottam tehát egy alkalmazást, amit felkészítettem új jelfeldolgozó motor befogadására.

2 A terület alapjainak áttekintése

Mind az agy-számítógép interfészek, mind a virtuális valóságok témaköre jelentős háttérismeretet igényel. A későbbi fejezetek mélyebb megértéséhez tekintsük előbb át azokat a technológiákat, módszereket, amikre a munkám során építettem.

2.1 Elektroenkefalográfia

Az Elektroenkefalográfia (EEG, [3]) tágabb értelemben egy pszicho-fiziológiai mérési eljárás, amivel a pszichés működés élettani hátterét vizsgálhatjuk, míg szűkebb értelembe egy elektro-fiziológiai mérőműszer (elektroenkefalográf), mely a neuronok elektromos aktivitásának mérésére szolgál valós időben. Az EEG-vel rögzíthető jel az elektroenkefalogram, amely egy komplex, több komponensű periodikus görbeként írható le, és számos értelmezésben, ábrázolásban (montázs) használatos.

2.1.1 EEG jelek mérése

A jel rögzítésének két módja létezik. Egyik, az invazív módszer, mikor a koponyába fúrt lyukakon közvetlenül az agyszövetbe helyeznek mikroelektródákat, míg másik esetben a koponyára tapasztott elektródákkal dolgoznak (non-invazív módszer). Mindkét módszer érzékeny a zajra, előbbi az agyvíz, betokozódás, utóbbi pedig főleg a koponyacsont és a távolság miatt. A non-invazív esetben létezik egy 10-20-as rendszer [4] az elektródák elhelyezésére, amivel leggyakrabban 31, 63 vagy 123 elektródát alkalmaznak (manapság egyre elterjedtebb a 10-10-es rendszer, ahol még több elektródát használnak). A hullámok mindig elektródák közötti potenciálkülönbségekként érzékelhetőek, és attól függően, hogy mit választunk referencia-értéknek, különböztetjük meg a montázsokat.

2.1.2 Hullámtípusok

A mért agyhullámokat frekvenciájuk alapján több kategóriába sorolják. Ezek általában valamilyen éberségi szintre, vagy valamilyen mentális cselekvésre jellemző hullámok, ahogyan a különféle agyi aktivitások más és más jellemzőjű agyhullámokat eredményeznek.

- **Delta-hullám**

  1–4 Hz-es, nagy amplitúdójú, kis frekvenciájú hullám, főleg a bal oldali temporális kéregben jellemző. Felnőtteknél mély NREM alvásban jelentkezik, de éber állapotban egyes kognitív funkciók alatt is jellemző, bár ilyenkor igen kis jelentőséggel bír.

- **Theta-hullám**

  4–7 Hz-es frekvenciájú, változó amplitúdójú hullám. Éber állapotban felnőtteknél időszakosan, rendszertelenül fordul elő, főleg a frontális, frontocentrális területeken. A frontális területeken mért theta-aktivitás feszült koncentrációt és erős érzelmeket jelenthet, de feltehetőleg az emlékek elmélyülésében is szerepet játszik.

- **Mu-hullám**

Lokalizációját tekintve centrális, alfa-frekvenciájú hullám (általában 8–10 Hz), amely a szenzomotoros kéreg nyugalmi állapotát reprezentálja. Fő különbség az Alfa-hullámokhoz képest, hogy kontralaterális (ellentétes irányú) mozdulatok végrehajtásakor blokkolódik.

- **Alfa-hullám**

Éber állapotban bilaterális posterior (hátul, két oldalt) területeken mérhető 8–12 Hz-es alaphullám, általában az occipitális (nyakszirt körüli) területek felett magasabb amplitúdóval. Az alfa-ritmus szemcsukáskor, nyugalmi állapotban occipitális területek felett fokozódik. Vizuális és mentális cselekvések blokkolják, és egy kisebb amplitúdójú, nagyobb frekvenciájú hullám lesz a dominánsabb (béta-hullám). Ez a folyamat a deszinkronizáció (egy nagyobb amplitúdójú és kisebb frekvenciájú hullámot egy kisebb amplitúdójú, de nagyobb frekvenciájú komponens vált fel).

- **Béta-hullám**

A pontosabb definíció szerint a 13 Hz feletti frekvenciájú, 20 μV-nál kisebb amplitúdójú hullámok. A normál megfigyelt tartomány 18–25 Hz között van, a sávszélesség ritkán haladja meg a 30 Hz-t, és dominánsan a frontális kéreg felett jelenik meg. Éber állapotban nyitott szemmel ez az alapaktivitás, feltehetőleg kognitív folyamatokat jelez. Szorosan összefügg a motoros viselkedéssel.

- **Gamma-hullám**

30–100 Hz közötti hullám, amely feltehetőleg különböző neuropopulációk összeköttetését jelzi az agyi régiók közötti kommunikáció céljából. Jelentéssel bíró ingerek feldolgozásához, egyes kognitív folyamatokhoz és motoros funkciók végrehajtásához köthető.

2.1.3 EEG jelek feldolgozása

EEG jelek számítógépes feldolgozását több célból is végzik. Egyrészt diagnosztikai célból [5], mivel jól bevált módszerek léteznek különféle neurológiai elváltozások, mint például epilepszia vagy demencia detektálására, vagy éppen alváskutatás témakörében [6].

Másik megközelítés, az agy-számítógép interfészek (BCI) alkalmazása. Ebben az esetben az EEG jeleket, mint egy nem-motorikus biológia jelet tekintenek, és különféle módszerekkel igyekeznek azt feldolgozni, és a számítógép számára értelmezhető parancsokká alakítani. Munkám során ezzel foglalkoztam, így ezekről bővebben írok a 6. fejezetben.

2.2 Virtuális valóság

A virtuális valóság olyan szimulált világ, ahol valós, vagy akár fiktív környezetben lehetséges a jelenlétünket emulálni. A virtuális valóság rendszerek pedig szűkebb értelemben, azok a hardver és szoftver komponensek, amik segítségével ez a szimuláció megvalósulhat.

Az ilyen VR rendszerek legtöbbször vizuális élményt nyújtanak: HMD (Head Mounted Display), sztereoszkópikus képmegjelenítő, vagy éppen CAVE (lásd később) technológiák alkalmazásával [7]. Napjainkban azonban egyre elterjedtebb a többi érzékszervre is kiterjedő VR rendszerek: hallás, vagy éppen a szomatoszenzorikus érzékek (pl. tapintás, mozgás- és helyzetérzékelés) stimulálása segítségével igyekeznek elérni a minél immerzív környezetet.

Számos jelentős intézet foglalkozik VR kutatásokkal, csak néhányat emelnék ki közülük [8]:

Intézet	**Kutatási területek**
CRVM - Mediterranean Virtual Reality Center	4-falú CAVE, Motion Capture (mozgás-digitalizálás), 3D hangrendszer [9]
Max Planck Institute for Psycholinguistics	VR szemüveg, mozgáskövetés [10]
University of Michigan	Motion Capture, BCI, 4 falú CAVE, haptikus eszközök (tapintás, erő-visszacsatolás), 3D digitalizáció [11]
Virginia Tech	Nagyfelbontású 4-falú CAVE [12]

2-1. táblázat: VR kutatóintézetek

2.3 Alkalmazott protokollok

Munkám során egy komponens-alapú rendszerrel dolgoztam, ami egy jól definiált protokoll-architektúrára épül. Álljon itt egy rövid összefoglaló ezekről.

2.3.1 CORBA

A Common Object Request Broker Architecture (CORBA) az Object Management Group (OMG) által létrehozott architektúra, és szabványok összessége számítógépes hálózatok kommunikációjának szabványosítására [13].

A CORBA szabványok meghatározásakor a cél az volt, hogy lehetővé tegyék a különböző operációs rendszereken futó, más és más programozási nyelveken megírt alkalmazások közötti kommunikációt is. Ennek érdekében az egymással kommunikáló alkalmazások közötti interfészt egy, a CORBA szabvány által meghatározott interfészleíró nyelven (IDL) kell definiálni.

2.3.2 RT middleware

Az RT-Middleware (RTM) egy szoftverplatform robot rendszerek (RT system) építésére oly módon, hogy a robot funkcionális elemeinek (RT functional element) a szoftver moduljait kombináljuk [14].

Egy RT funkcionális elem egy olyan robot komponens, ami valamilyen átfogó funkcióval rendelkezik, legyen az egy szervomotor, szenzor, kamera, vagy éppen ezek kombinációjából felépülő eszköz. Ezen felül pusztán szoftveres elemek, mint egy útkeresési, vagy képfeldolgozási algoritmus is lehetnek RT funkcionális elemek.

RT-Middleware terminológiában, az RT fonkcionális elemek szoftveres komponenseit hívjuk RT komponenseknek (RTC). Minden RTC-nek vannak Portjai (interfészei), hogy adatokat cseréljen, vagy kommunikáljon más komponensekkel. Maga az RT-rendszer úgy épül fel, hogy ezeket a Portokat más komponensek Portjaihoz kapcsoljuk, ezáltal azok funkcionalitását összegezzük.

A middleware definíciója általában „az a szoftver, ami az operációs rendszer és az alkalmazás között található, és olyan könyvtárakat, programokat, stb. jelent, amik bizonyos alkalmazási körben kényelmesebb felhasználást tesz lehetővé”.

Az RT-Middleware szó szerint egy middleware robot-rendszerekhez, ami a következő elemekből áll:

- RT-Component framework
- RT-Middleware
- Basic RT-Component Group
- Libraries
- Basic Service Group
- Basic tool Group

Munkám során ennek két implementációjával találkoztam: az OpenRTM-aist 1.0-val [15] és az OpenRTM.NET 1.3-mal, aminek sajnálatos módon csak japán nyelvű dokumentációja érhető el. Ezek mind CORBA alapú üzenetszórásos rendszerek, azaz a komponensek közötti kommunikáció egy névszerver közreműködésével jut el az egyes RTC-khez.

2.3.3 ICE

Az Internet Communications Engine (ICE) egy modern, objektum-orientált eszköztárat nyújt elosztott alkalmazások hatékony és könnyű fejlesztéséhez [16]. Az ICE platformfüggetlen, így alkalmazásainkat akár különféle operációs rendszereken, különböző programozási nyelveken is fejleszthetjük. Alkalmazása elfedi előlünk a hálózati kapcsolat kiépítésének, adatok sorosításának, sikertelen kapcsolatfelvétel esetén való újrapróbálkozások, és számos más, alacsonyszintű feladat problémáját.

Munkám során a távoli függvényhívások szolgáltatását használtam fel. Egy szabványos leíró segítségével definiálhatunk interfészeket, majd ezeket a leírókat nyelvspecifikus burkolókká fordíthatjuk, ezáltal egyszerűen megoldva a távoli eljáráshívás problémáját.

3 Alkalmazott eszközök

BCI eszközök közül az Emotiv EPOC neuroheadsettel dolgoztam, mivel a tanszéken ez az eszköz elérhető, és ennek használatában volt már korábbi tapasztalatom. Ez egy olcsó, könnyen beszerezhető eszköz, ugyanakkor kiterjedt alkalmazói és fejlesztői támogatással rendelkezik.

Virtuális valóságok közül a VirCA virtuális kollaborációs arénával dolgoztam. Ezt a MTA SZTAKI 3DICC laboratóriuma fejleszti, és a Távközlési és Médiainformatika Tanszék által meghirdetett tárgyak révén ezzel a rendszerrel is volt már tapasztalatom.

3.1 Emotiv Epoc neuroheadset

Az Emotiv EPOC neuroheadset egy vezeték nélküli EEG jelek mérésére alkalmas eszköz [17]. Átnedvesített szivacsok biztosítják a fejbőrtől az elektródákig az elektromos jelek vezetését. Kialakítása miatt a legtöbb fejformára jól illeszkedik, és ezeken a jelek mérésének helyei nagyjából megegyeznek, ami elengedhetetlen a jelek komolyabb vizsgálata szempontjából. Felépítése igen robosztus, könnyedén felhelyezhető, viselete nem zavaró, így alkalmazása számos környezetben kivitelezhető.

3-1. ábra: Emotiv Epoc neuroheadset

A headset a következő paraméterekkel rendelkezik:

Csatornák száma	14 (és CMS/DRL referencia elektródák)
Használt csatornák (Nemzetközi 10-20-as rendszer alapján)	AF3, AF4, F3, F4, F7, F8, FC5, FC6, P7, P8, T7, T8, O1, O2
Mintavételezés	Szekvenciális mintavételezés, egy ADC, kb. 128 Hz (2048 Hz előfeldolgozás előtt)
Felbontás	16 bit (14 bit előfeldolgozás után) 1 LSB = 1.95μV
Sávszélesség	0.2 – 45 Hz, 40 Hz és 60 Hz szűrve
Csatlakozás	Wireless, 2.4G Hz csatorna
Akkumulátor	Li-poly, kb. 12 óra üzemidő

3-1. táblázat: Emotiv EPOC jellemzői

A fentieken kívül, az eszközben található még egy két szabadsági fokú giroszkóp, a fejmozgás követésére.

3.2 EmoEngine

A headset funkcióit egy EmoEngine nevű jelfeldolgozó logikán keresztül érhetjük el. Az EmoEngine egy C++ nyelven írt könyvtár, ami magába foglalja a headset-től érkező adatok elérését, és azok különböző szempontok alapján történő feldolgozását (Suite-ok). Ezt felhasználva több, az eszközzel együtt szállított alkalmazással akár programozás nélkül is használhatjuk az eszköz funkcióit, de egy SDK segítségével mi magunk is fejleszthetünk újabb programokat.

Az EmoEngine szolgáltatásai a következők:

- Profilkezelés: különböző felhasználói beállítások menedzselése
- Expresssiv Suite: arcmimika detektálása.
- Affectiv Suite: hangulati jelek mérése
- Cognitiv Suite: gondolati parancsok tanulása és értelmezése

A három Suite a következő állapotokat definiálja:

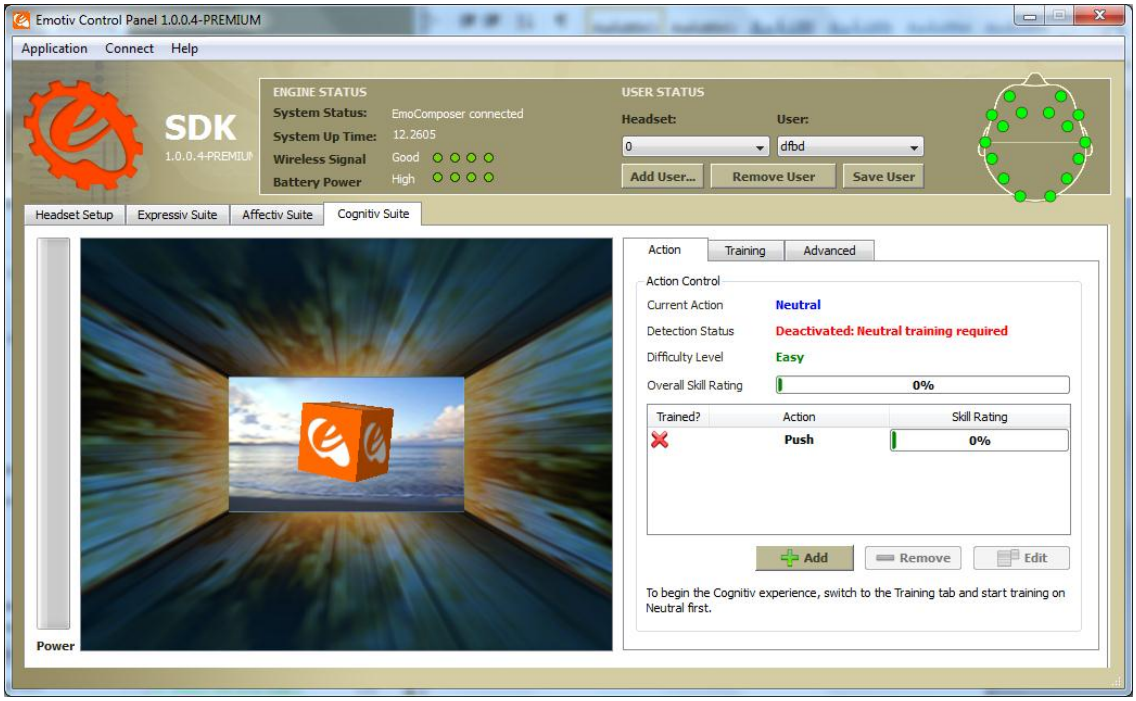
Suite	Állapot	Tapasztalataim
Expressiv	Pislogás (Blink) Bal/jobb kacsintás (Wink) Balra/jobbra pillantás (Look left/right) Szemöldök felhúzása (Raise brow) Szemöldök összehúzása (Furrow brow) Mosoly (Smile) Fogösszeszorítás (Clench) Jobb/bal szájhúzás (Smirk) Nevetés (Laugh)	Kalibrálható bináris állapotok, de nagyjából mindenkinél meg-egyeznek. Tapasztalataim alapján nagy pontosságot el lehet érni velük, de mivel nem agyi jelek, nem foglalkoztam velük.
Affectiv	Unalom (Boredom) Frusztráció (Frustration) Meditatív állapot (Meditation) Rövid távú izgatottság (Instantaneous Excitement) Hosszú távú izgatottság (Long term excitement)	Nem kalibrálható, százalékos jellemzők, nem személyfüggőek. Az elnevezésük nem feltétlen pontos, de nagyjából azt az érzelmi állapotot jelentik.
Cognitiv	3 tengely mentén való elmozdítás (Push, Pull, Left, Right, Up, Down) 3 tengely mentén való elfordítás (Rotate left, right, clockwise, anti-clockwise, forwards, reverse) Eltüntetés (Disappear)	Százalékos jellemzők, később ismertetett kalibrálási folyamat során taníthatóak.

3-2. táblázat: EmoEngine funkciói

Munkám során főleg a Cognitiv Suite-tal foglalkoztam, annak tanításával és a kognitív parancsok kalibrálásával. Az Expressiv Suite nem konkrétan BCI jellegű szolgáltatást ad, míg az Affectiv által definiált állapotok nem elég pontosan definiáltak, és nem lehet őket kellő magabiztossággal tudatosan befolyásolni.

3.2.1 Control Panel

A fentebbi szolgáltatásokat elérhetjük az Emotiv Control Panel segítségével, amivel nyomon követhetjük az összes jellemző aktuális értékét.



3-2. ábra: Emotiv Control Panel Cognitiv panelje

Ez az alkalmazás egy kész felületet ad a kezünkbe, aminek segítségével konfigurálhatjuk a profilunkat, és beállíthatjuk a különféle Suite-okat. Lehetőségünk van konkrét headset-hez csatlakozni az EmoEngine használatával, de akár egy EmoComposer nevű szoftverhez is, amivel emulálhatjuk az interfészt. Ez esetben az egész jelfeldolgozó motort megkerüljük, és konkrét, értelmezett állapotokat tudunk kiváltani, ami tesztelési célból hasznos.

3.2.2 Kalibrálási folyamat

Az EmoEngine Cognitiv Suite-ját használat előtt kalibrálni kell. Ez a kalibráció a következő lépésekből áll:

1. Profil kiválasztása: vagy új profilt, vagy már meglévőt töltünk be, hogy a beállításokat elmenthessük. Meglévő esetén egyből a 4. Lépésre is léphetünk.
2. Semleges jel felvétele: szükséges felvenni egy semleges mintát, ami az alapjelnek felel meg. Ez az az agyi állapot, amikor semmilyen parancsot nem akarunk aktiválni.
3. Különböző kognitív állapotok (pl. Push) betanítása: kiválasztunk egy gondolatot, amire tanítani szeretnénk az adott parancsot, és a tanítás idejére igyekszünk minél jobban azt előidézni. Célszerű nem túlzottan erőlködni, illetve jól elkülönülő gondolatokat választani a különböző parancsokhoz. A semleges állapotot is taníthatjuk, sőt, gyakran szükség is van rá (ha túl sok a hamis-pozitív detekció).
4. Alkalmazás: a tanítás után a Cognitiv Suite készen áll az alkalmazásra. Ha elégedetlenek vagyunk a hatékonysággal, bármikor visszaléphetünk a 3. Lépésre.

Ezt a folyamatot implementálja az Emotiv Control Panel, ahol ezeket a lépéseket a grafikus felületen végezhetjük el, és az egyes parancsok tanításához egy lebegő kocka ad segítséget, ami a tanítandó parancsnak megfelelő mozgást végez.

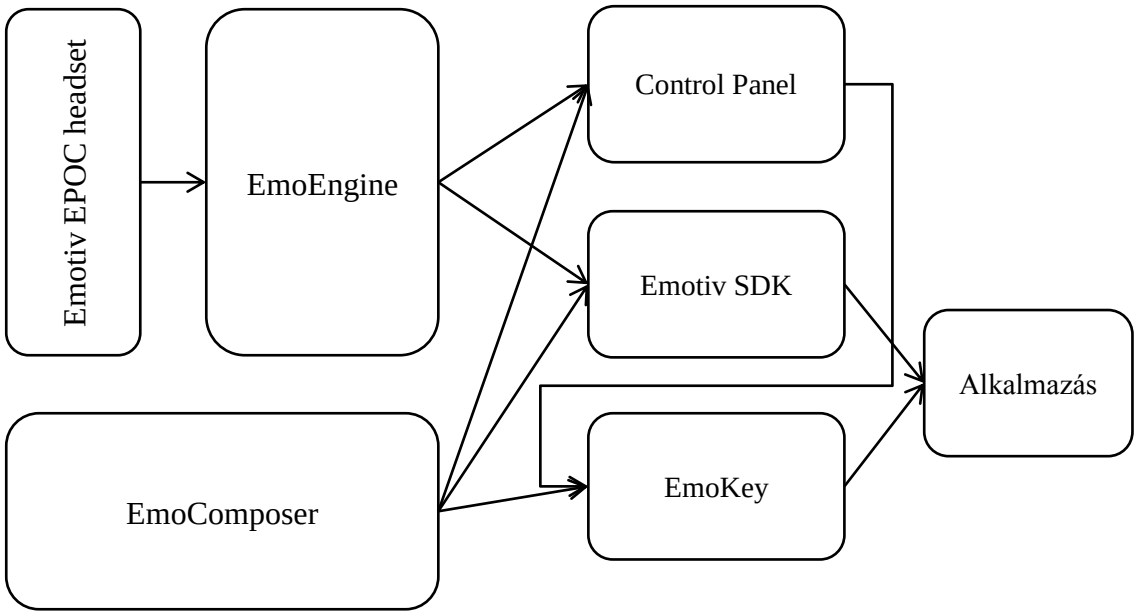
3.2.3 EmoKey

A Control Panelhez kapcsolható egy EmoKey nevű alkalmazás, aminek segítségével billentyűzet, illetve egér eseményekké konvertálhatjuk az érzékelt parancsokat. Különböző szabályokat adhatunk hozzá, amik meghatározzák, hogy milyen billentyű- vagy egér eseményt szeretnénk kiváltani, és azok melyik alkalmazás eseménysorába kerüljenek. Minden szabályhoz kapcsolnunk kell feltételeket, amik az EmoEngine aktuális állapotára vonatkoznak, például a „Push" erőssége több mint 0.3, és ezek együttes teljesülésekor aktiválódik a szabály. Az EmoKey-t is lehet a fentebb említett EmoComposer-rel használni.

3.2.4 Emotiv SDK

A fentebb ismertetett alkalmazásokkal könnyen fejleszthetünk programokat, amik kihasználják a headset lehetőségeit, ha a megfelelő billentyűzet-eseményeket kezeljük. Azonban az eszközhöz tartozik egy SDK, amivel az egész EmoEngine-t beépíthetjük az alkalmazásainkba. Ez egy C++ nyelven írt modul, amihez C# és Java wrappert is szállítanak. Az SDK segítségével a fentebb leírt szolgáltatásokat (profilkezelés, Expressiv, Affectiv, Cognitiv suite), és azok konfigurálását metódushívásokkal érhetjük el. Továbbá, mivel az általam használt SDK verzió a Research Edition, lehetőségünk van a nyers szenzor-adatok lekérdezésére is. Tesztelési célból, az SDK is képes az EmoComposer-hez csatlakozni, de ez esetben természetesen nincs lehetőség nyers adatok lekérdezésére.

3.2.5 Emotiv Epoc architektúra



3-3. ábra: Emotiv EPOC architektúra

Mint az a fentebbi ábrán is látszik, alkalmazás fejlesztésére tehát két lehetőségünk van, az SDK és az EmoKey alkalmazása. Mivel célom először egy

működő prototípus elkészítése volt, így az EmoKey-t választottam, és az eszköz kalibrálását a Control Panel segítségével, az alkalmazástól függetlenül végeztem.

3.3 VirCA

A VirCA [18] nem más, mint egy elosztott virtuális valóság, melyben a komponensek közötti kommunikáció szabványos csatornákon keresztül (OMG RTM és ICE) zajlik. Mindez egy egyszerű fizikai szimulációval, és Ogre3D alapú megjelenítéssel kiegészítve egy teljes értékű virtuális környezetet teremt. Ezen felül nagy hangsúlyt fektettek a komponensek (CyberDevice-ok) térbeli és logikai szeparációjára, így azok külön hardveren futhatnak akár földrajzilag távol egymástól, ezzel elősegítve a különböző kutatócsoportok együttműködését.

A VirCA legfontosabb jellemzői a következő négy elképzelésen alapulnak.

a) „3D Internet Kollaboráció”

A VirCA lényegében egy platformot ad a kezünkbe, melyen a felhasználók létrehozhatnak, megoszthatnak és valós időben interakcióba léphetnek 3D-s tartalommal, mindezt úgy, hogy a háttérben működő hardverek és szoftverek térben elosztottan működhetnek, és csak egy IP hálózat kapcsolja őket össze. Például több kutatócsoport dolgozhat egy akadály-elkerülő rendszeren egy robothoz. Az egyik elkészíti a robotot, míg a másik a vezérlőt, és ezeket a platformon keresztül összekapcsolhatják. Ezen felül, a VirCA grafikus komponense fel van készítve rá, hogy sztereoszkópikus 3D megjelenítőkkel dolgozva a későbbiekben részletezett CAVE rendszert üzemeltesse, ezzel teremtve egy nagyon valósághű, immerzív környezetet.

b) „Kiterjesztett kollaboráció”

A VirCA 3D-s tartalmát és a vezérlő folyamatait lehetőségünk van szinkronizálni a valós világgal, ezzel tovább növelve a lehetőségeinket. Például egy tényleges robot-kart beköthetünk a rendszerbe, majd a virtuális terünkben ezt vezérelve a fizikális kar ugyan azt a mozgást fogja elvégezni.

c) „Plug & Play”

A VirCA szabványos, moduláris RT-Middleware alapú keretrendszere lehetővé teszi, hogy a komponenseinket különböző forrásokból is egyszerűen és hatékonyan összekapcsolhassuk. Ennél egy réteggel feljebb találjuk a CyberDevice-oknak nevezett VirCA-képes szoftver komponenseket, melyek egy szabványos felületen keresztül becsatlakozhatnak a 3D-s virtuális környezetbe. Hosszú távú céljuk, hogy olyan komponensek legyenek széles körben elérhetők, amikkel tipikus feladatok egyszerűen, drag & drop módon megoldhatóvá váljanak. Ennek figyelembe vételével igyekeztem elkészíteni a BCI komponensemet.

d) „Csúcstechnológián túl”

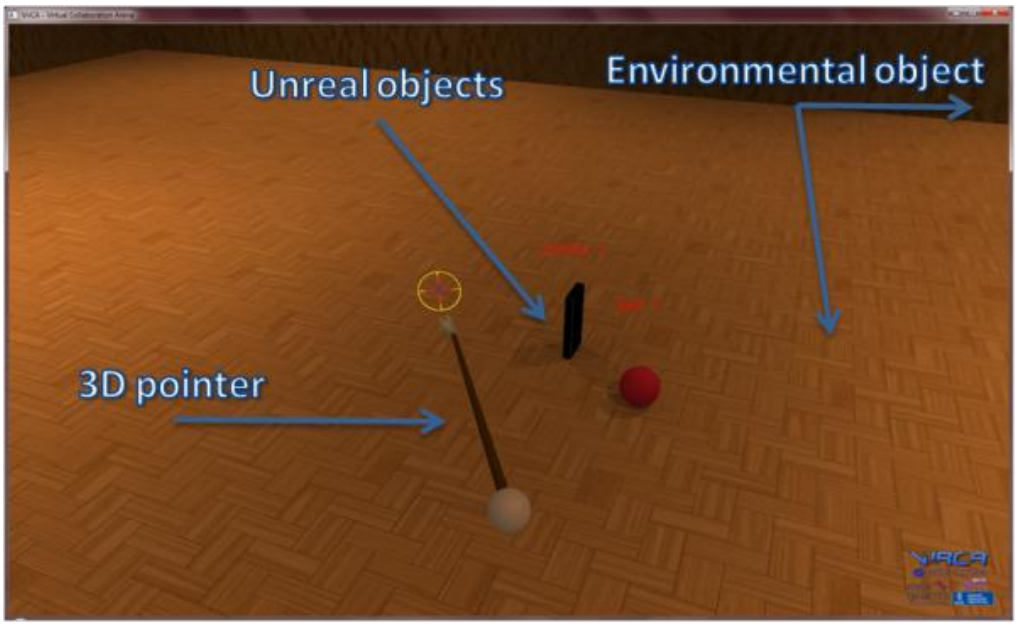
Mivel a virtuális világunk teljes egészében számítógépes szimuláció, így bármilyen mérési adat azonnal rendelkezésre állhat virtuális szenzorokon keresztül, amik még lehet, hogy nem is léteznek a valóságban. Ez a virtualitás lehetővé teszi, hogy olyan technológiákat is teszteljünk, amik még nem is léteznek. Például egy robot vezérlő komponensét anélkül tesztelhetjük, hogy ténylegesen el kellene készíteni a hardvert hozzá, ami jelenleg még miniatürizálási és energetikai problémákba ütközne.

3.3.1 Hardver

A VirCA hardveres komponense jelenleg egészen egyszerűen bármilyen, IP hálózatra köthető eszköz lehet. Fontos azonban külön megemlíteni magát a VirCA grafikus komponenst, amin keresztül „beláthatunk” a virtuális térbe. Ez bármilyen PC-n futhat, ami kellően nagy grafikus teljesítménnyel rendelkezik, és támogatja a DirectX 9.29-es verzióját. Egy másik fontos komponens, a VirCA System Editor, ami a komponensek összekapcsolását végzi, és megemlítendőek még maguk a CyberDevice-ok, az egyes komponensek. Tekintve, hogy ezek mind külön hardveren futhatnak, így a szükséges teljesítmény nem nagy, azonban a hálózatra nagyobb teher nehezedhet. Ezeket vagy helyi hálózaton kapcsolhatjuk össze, vagy VPN-en keresztül egy virtuális helyi hálózatot építünk ki.

3.3.2 Szoftver

A VirCA három fő szoftver-komponensből áll, a 3D vizualizációból, a System Editorból és a többi Cyber Device-ból (habár maga a 3D-s felület is egy Cyber Device).



4-3-4. ábra: a VirCA 3D felülete

VirCA fő komponens

A fő komponens a 3D vizualizációs komponens, ami magába foglalja a 3D virtuális tér megjelenítését, a fizikai szimulációt (Bullet engine), beszédszintézist és menü alapú vezérlést. Egyéb komponenseket különféle interfészeken keresztül csatlakoztathatunk hozzá. Ezek lehetnek RT adatportok, RT service-portok és kiterjesztett RT-ICE portok, amiket VirCA portoknak is neveznek.

Létezik még továbbá egy SDK, amiben definiálva vannak többek között a felhasználói inputért felelős függvények ICE leírói. Ezeket Slice fájloknak nevezik, és a bennük definiált függvények az alábbi 5 csoportra bonthatók:

a) Kamera

Ezen az interfészen keresztül lehetséges a kamera mozgatása. Többfajta kamera-mód létezik, a szabad mód, amikor tetszőlegesen lehet mozgatni a nézőpontot, és a követő mód, amikor a kamera egy objektumhoz, például a pointerhez van kötve. Ez utóbbi az alapértelmezett.

b) Pointer

Lényegében ez a tényleges avatár, ezzel manipulálhatjuk a virtuális környezetet. Olyan, mint egy kar, aminek a vége mutat valamire, és a hosszának állításával „átmutathatunk" közeli objektumok mögé, vagy akár azokat megragadva mozgathatjuk őket. Ezt mozgatva, és a kamerát ehhez kötve a játékokban megszokott belső nézetben láthatjuk a virtuális teret.

3-5. ábra: VirCA Pointer

c) Menu

Egyszerű és áttekinthető vízszintes overlay menük és almenük. Ezekben navigálás jobbra illetve balra, elfogadás (almenübe lépés) és kilépés lehetséges.

3-6. ábra: VirCA menürendszere

d) Wiki

Beszélő wikipédia interfész, jelenleg még fejlesztés alatt áll, különösebben nem foglalkoztam vele.

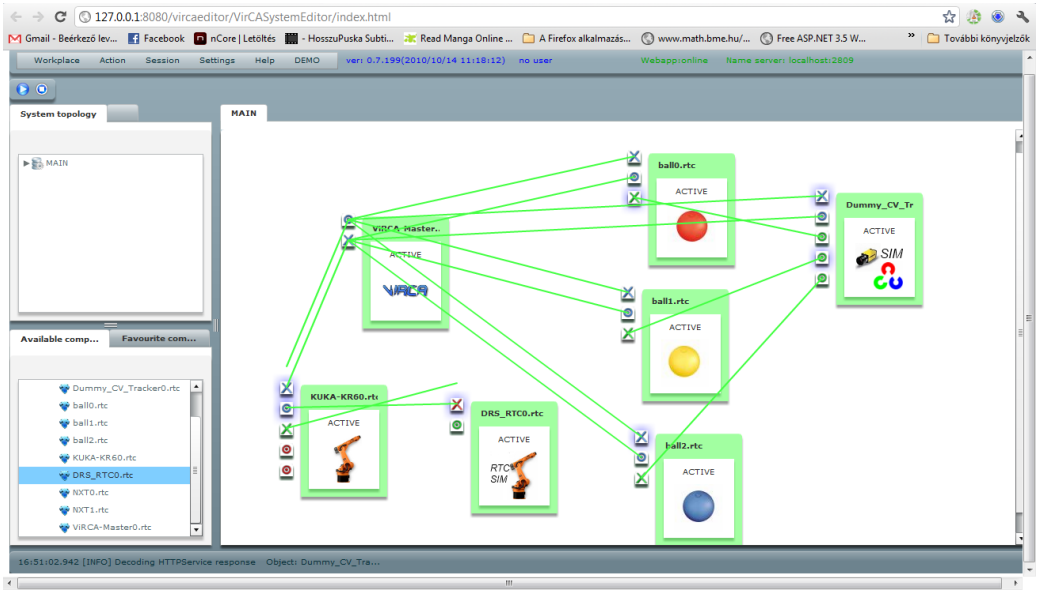
e) LiteralCommand

A VirCA konzol parancsok bevitelére alkalmas script nyelvének interfésze, komolyabban ezzel sem foglalkoztam.

Ezeket az ICE leíró Slice (Specification Language for ICE) fájlokat lehet nyelv-specifikus interfészekké (C++, C#, Java, stb.) fordítani, amikkel kezelhető a felhasználói bemenet. A lefordított interfészeket felhasználva transzparens módon hívhatjuk meg a bennük definiált függvényeket, anélkül, hogy nekünk az adatok konvertálásával, sorosításával, vagy a programozási nyelvek közötti eltérésekből eredő problémákkal kellene foglalkoznunk.

3.3.3 System Editor

A VirCA System Editor-ban lehetséges a komponensek összekapcsolása egy könnyen kezelhető és átlátható grafikus felületen. A System Editorban állíthatjuk össze a rendszerünket alkotó komponensek architektúráját, és indíthatjuk el, illetve állíthatjuk le a rendszert akár komponensenként is. Mindezt egyszerű webes felületen keresztül (4-7. ábra), kényelmes drag & drop módon tehetjük meg.



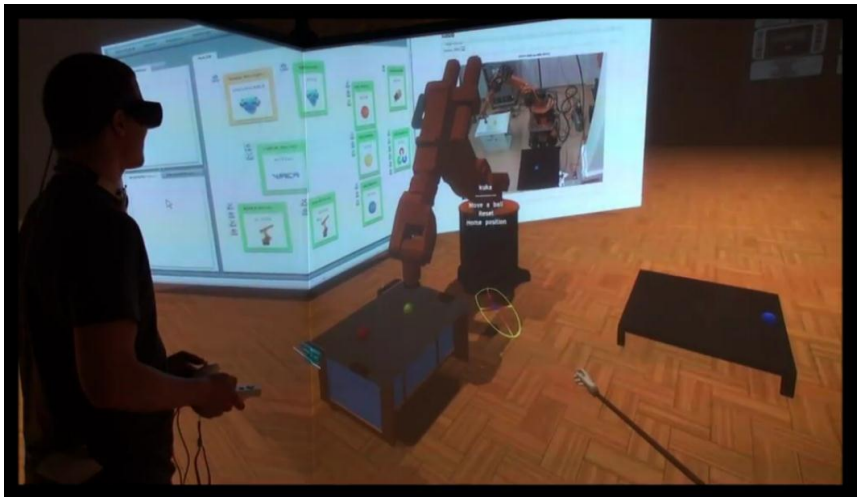
3-7. ábra: VirCA System Editor

Láthatóak a VirCA portok (kék), RT service-portok (piros) és RTM adatportok (zöld). Az X jelűek kimenetek, míg az O jelűek a bemenetek, és a felület biztosítja, hogy csak kompatibilis portokat köthessünk össze.

Az egész rendszer az OMG RTM 1.0-ra épül, ami biztosítja a komponensek közötti kommunikációt.

3.3.4 3D Cave

A 3D CAVE (4-8. ábra) nem más, mint egy vizualizációja a VirCA virtuális világának. Három darab 2x2 méteres hátulról 3D projektorral megvilágított vászon alkotja, melyeknek képeit egy nagy grafikus teljesítményű hardver állítja össze úgy, hogy a közrefogott területen található 3D-s szemüveggel folytonos képet láthassunk. Ehhez használnak egy elektromágneses helyzetérzékelőt, hogy megállapítsák a szemüveg helyét és orientációját. A rendszerrel való interakció különféle input-eszközökkel történik, például Kinect kamera, Wii kontroller vagy hang alapú kommunikáció. Munkám során külön figyelmet fordítottam rá, hogy a BCI eszköz a CAVE-ben is használható legyen.



3-8. ábra: VirCA CAVE vizualizációja

4 BCI eszköz integrálása virtuális valóságba

A célom az volt, hogy valamilyen módon összekössem a virtuális valóságot a BCI eszközzel. Ehhez megvizsgáltam, milyen vezérlési lehetőségek vannak a VirCA rendszerben, illetve hogy az Emotiv EPOC milyen lehetőségekkel rendelkezik. A két rendszer közé kellett tehát kialakítanom egy interfészt, ami a közöttük történő kommunikációt kezeli, és a parancsokon a megfelelő transzformációt elvégzi.

A VirCA vezérlését két funkcióra osztottam. Egyrészt szükség van a virtuális térben való mozgásra, az avatár-vezérlésre. Az irányítás dimenzióit az EmoEngine limitálja. Egyelőre egy két szabadsági-fokú irányítás: a felhasználó haladhat előre-hátra, illetve fordulhat jobbra és balra. Ennél több szabadsági fokot az EmoEngine nem enged, mivel egyszerre maximum 4 utasítást képes elkülöníteni egymástól.

Másik funkció a virtuális tér objektumainak vezérlése. Mivel rengeteg CyberDevice létezik, első lépésként kiválasztottam egyet, a ColoredBall-t, aminek vezérlése annyiban merül ki, hogy be tudjuk állítani a pozícióját

Úgy döntöttem, hogy a két funkciót egymástól függetlenül implementálom, így egyszerűbb architektúrát tudok kialakítani, és hamarabb jelentkeznek az esetleges problémák. Mindkét esetben a fentebb említett EmoKey-féle megoldást választottam, és különösképpen ügyeltem arra, hogy a VirCA architektúrájának megfelelő kommunikációs csatornákat alkalmazzak.

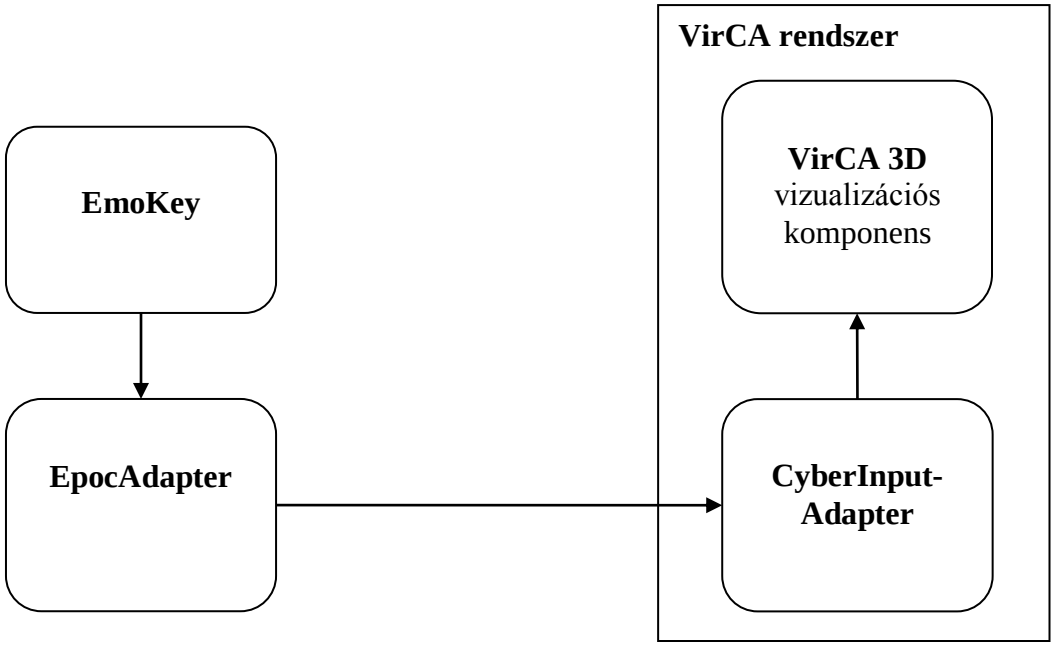
4.1 Avatár-vezérlés

A VirCA 3D világában a megszokottól kicsit eltérő az avatár fogalma. Objektumokkal interakcióba lépni az úgynevezett „Pointer”-rel tudunk, amit az előző fejezetben említett ICE interfészen keresztül tudunk irányítani. Ez azonban csak bemeneti eszköz, a felhasználó által látott kép ettől teljesen független. Különféle virtuális kamerákat helyezhetünk el, amiken keresztül beláthatunk a virtuális világba. Ezeket a kamerákat a pointertől teljesen függetlenül tudjuk mozgatni, azonban lehetőség van objektumokhoz csatolni őket. A VirCA 3D vizualizációs komponens egy ilyen kamerát csatol a Pointerhez, így ezt a két objektumok együttesen nevezhetjük „Avatár”-nak, aminek irányítását a Pointer irányításával végezhetjük.

4.1.1 Architektúra

Megterveztem azt a rendszert, ami az Emotiv EPOC headset-et összeköti a VirCA avatár-mozgatásért felelős interfészeivel. Ehhez felhasználtam a korábban említett EmoKey alkalmazást, a VirCA fő komponenst, a későbbiekben bemutatott CyberInputAdaptert, és természetesen az általam tervezett EpocAdapter komponenst.

A rendszer felépítése a következő látható:



4-1. ábra: Avatár-vezérlés architektúrája

A VirCA fő komponenssel egy, a VirCA SDK-t használó CyberInputAdapter nevű CyberDevice tartja a kapcsolatot. Ehhez csatlakozik az általam készített EpocAdapter, ami pedig az EmoKey-től kapja Windows billentyűzet eseményekként az EmoEngine által detektált parancsokat.

A VirCA fő komponens, az EmoKey és a CyberInputAdapter adott volt, tehát megterveztem az EpocAdaptert úgy, hogy képes legyen az EmoKey által kiadott utasításokat a CyberInputAdapter számára továbbítani.

4.1.2 CyberInputAdapter

Ez egy előre elkészített VirCA komponens, feladata különféle bemeneti eszközök illesztése a VirCA fő komponenshez. Ezt használja például a Nintendo Wii kontroller is. Publikálja a Pointer3DManipulation és CameraManipulation interfészeket a VirCA rendszeren kívülre, ami az én céljaimra teljesen megfelelt.

4.1.3 EmoKey

Ehhez az interfészhez az EmoKey-t a következőképpen konfiguráltam:

Szabály	Feltétel	Feladat
Előre	Push értéke > 40%	Küldje a „W” karaktert lenyomását a kijelölt alkalmazás üzenetsorába, és tartsa lenyomva, amíg a feltétel igaz.
Hátra	Pull értéke > 40%	Mint fentebb, csak az ”S” karakterrel.
Jobbra fordul	Right értéke > 40%	Mint fentebb, csak az ”D” karakterrel.
Balra Fordul	Left Értéke > 40%	Mint fentebb, csak az ”A” karakterrel.

4-1. táblázat: EmoKey konfigurációja

Az EmoKey aktiválásakor lehetőség van kiválasztani, hogy melyik alkalmazás üzenetsorába küldje az eseményeket, vagy akár az aktuális ablakot is kiválaszthatjuk.

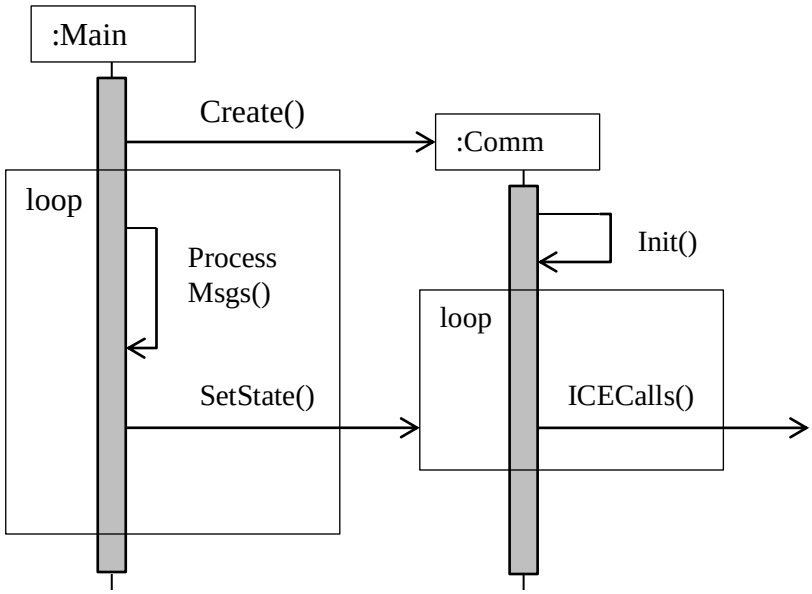
Ennek a konfigurálása után már csak az Emotiv Control Panel segítségével kellett tanítanom egy olyan profilt, ami a fentebb említett 4 állapot detektálására alkalmas.

4.1.4 EpocAdapter

Egy C#-ban fejlesztett Windows Forms alkalmazás, aminek egyetlen egy Form-jának célja fogadni és feldolgozni az Emokey-től kapott billentyűket. Az alkalmazást két-szálúra terveztem:

1. Szál: Windows események kezelése, Form rajzolása, billentyűk állapotváltozásának kezelése
2. Szál: A billentyűk állapotától függő időzített ICE függvényhívások.

A működést az alábbi szekvencia-diagram szemlélteti:



4-2. ábra: EpocAdapter szekvencia diagram

Az alkalmazás létrehozza az ICE kommunikációs szálat, ami inicializálja magát, majd a belső állapota alapján rendszeres időközönként meghívja a megfelelő ICE függvényeket, amik a Pointert mozgatják.

A főszál pedig kezeli az ablakhoz érkező eseményeket, és ha azok releváns billentyűzet-események voltak, akkor frissíti a kommunikációs szál belső állapotát.

Implementáció

Az ICE kommunikációhoz szükség van az interfészeket leíró C# modulokra, amiket a Slice fájlokból fordítottunk:

```
using Visualization3D.ExternalInput;
```

Egyrészt kell egy ICE kommunikátor, ami a központi ICE objektum, másrész szükséges a konkrét Pointer3DManipulation interfészt megvalósító proxy:

```
private Ice.Communicator communicator;
private Pointer3DManipulationPrx Pointer3DInterface;
```

Az inicializálás a kommunikátor inicializálásával kezdődik, majd utána két lehetőségünk van a proxy esetében.

```
communicator = Ice.Util.initialize(null);

//remote
Ice.ObjectPrx obj =
communicator.stringToProxy("Pointer3DManipulationProxy:default
-p 10001 -h 192.168.1.3");

//local
Ice.ObjectPrx obj =
communicator.stringToProxy("Pointer3DManipulationProxy:default
-p 10001");

Pointer3DInterface =
Pointer3DManipulationPrxHelper.checkedCast(obj);
```

Egyrészt, ha az egész rendszert a saját gépünkön futtatjuk, akkor az alapértelmezett host-ot, azaz a localhost-ot használjuk. Ha viszont a CyberInputAdapter másik gépen fut, akkor annak a helyi IP címét kell megadnunk a proxy létrehozásakor. Tesztelési és fejlesztési célból a localhost-ot használtam, viszont a CAVE-ben való tesztelés esetében egyszerűbb, ha a CyberInputAdapter is a VirCA vizualizációs komponenssel egy gépen fut, így az EpocAdaptert már annak az IP címével konfiguráltam.

Miután a proxy-t létrehoztuk, már csak a megfelelő függvényhívásokat kell elvégezi, például az előre haladáshoz:

```
Pointer3DInterface.action(
DataTypes.Pointer3DActionType.P3DJumpFwdType);
```

A DataTypes névtérben a VirCA SDK típusai vannak definiálva.

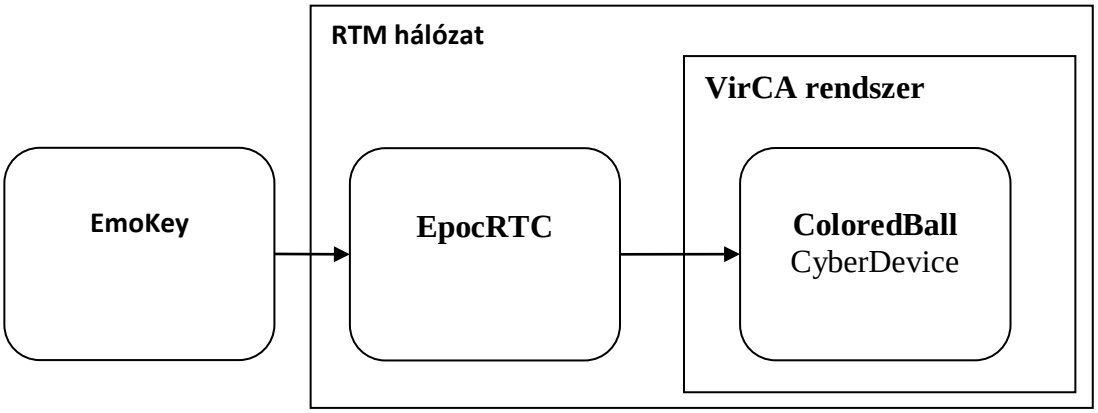
4.2 Virtuális objektum vezérlése

Mint korábban írtam, a virtuális valóság irányítása egyrészt az avatárunk irányításából, másrészt viszont a virtuális tér objektumaival való interakcióból áll. Következő célkitűzésem tehát az volt, hogy a virtuális valóság többi komponensét is képes legyek irányítani. Számtalan különböző fajta objektum létezhet azonban egy ilyen rendszerben, így hosszútávon valamilyen szabványosított megoldást kell kialakítani erre a célra, azonban első lépésként egy konkrét virtuális objektumot céloztam meg.

4.2.1 Architektúra

A virtuális objektumok vezérléséhez felhasználtam az EmoKey alkalmazást, egy, a fejlesztés céljára kiválasztott CyberDevice-t, a ColoredBall-t, és az általam készített EpocRTC komponenst.

A rendszer architektúrája a következő ábrán látható:



4-3. ábra: Objektum-interakció architektúra

Tekintettel arra, hogy a VirCA rendszer egy RTM alapú rendszer, minden VirCA CyberDevice egyben egy RT komponens is. Emiatt, ahogyan már korábban is írtam, a komponenseknek lehetnek szabványos RT portjaik is (bár szigorú értelemben véve, egy komponenst a rendszerben csak akkor nevezhetünk CyberDevice-nak, ha létezik VirCA portja, ami nem más, mint egy kiterjesztett RT szolgáltatásport). Emiatt úgy döntöttem, hogy a szabványos RT adatportokat kihasználva valósítok meg objektum-vezérlést, mivel ez a módszer a komponensek széles körénél alkalmazható.

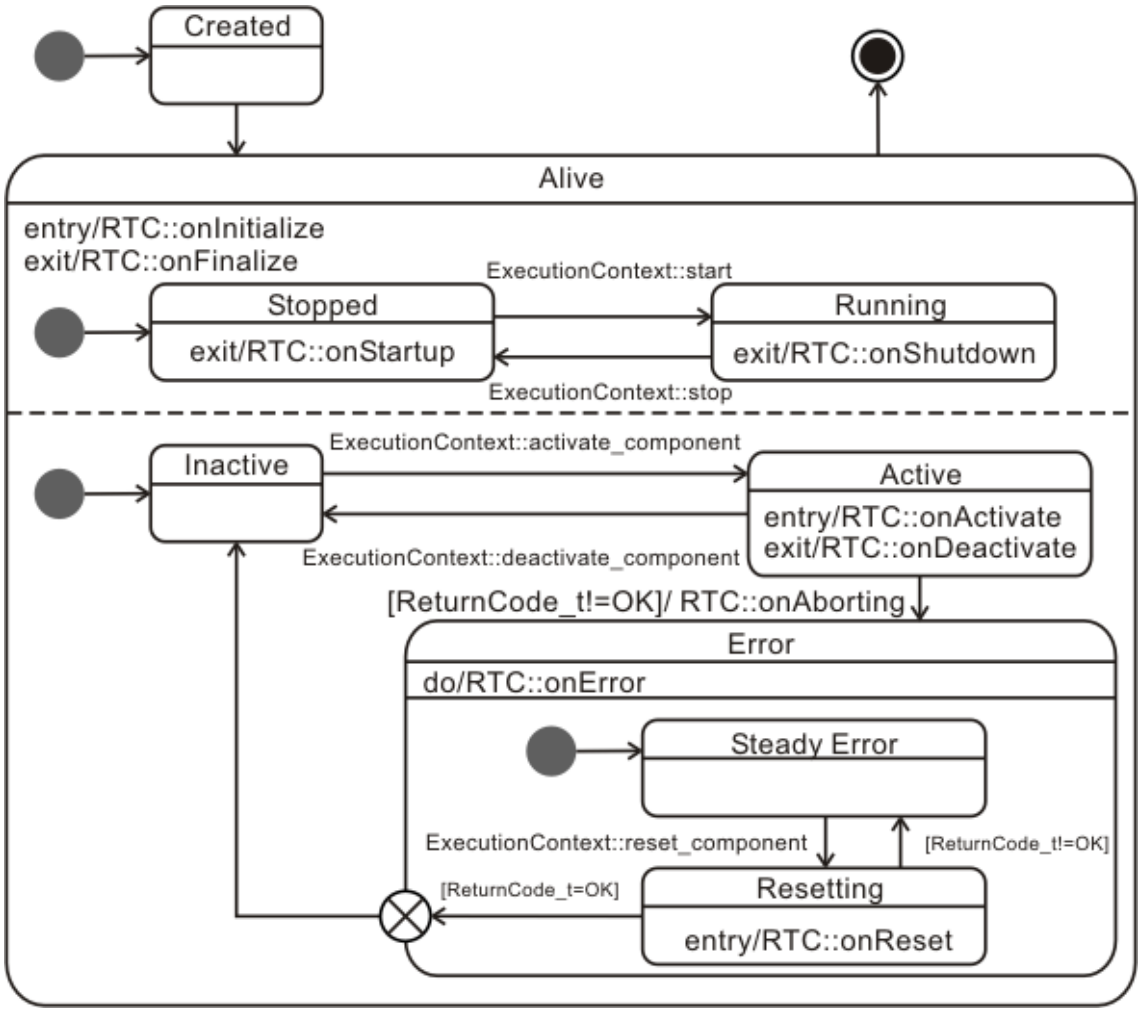
Így tehát az EpocRTC RT adatporton kommunikál a ColoredBall CyberDevice-szal, míg az EmoKey-től a korábban bemutatott módon, Windows billentyűzet-eseményekként kapja a parancsokat. A rendszert úgy terveztem meg, hogy az előző szcenárióban bekalibrált EmoKey és Control Panel ebben az esetben is használható legyen.

4.2.2 ColoredBall

A fejlesztéshez a VirCA rendszerhez készített, igen egyszerű ColoredBall CyberDevice vezérlését választottam. Ez a komponens egy labdát reprezentál a virtuális világban, aminek a szabványos VirCA portján kívül van egy RT adatportja is, amin a pozícióját lehet beállítani a 3D-s térben. Ezt a labdát akartam az Emotiv Control Panel kockájához hasonlatosan mozgatni azzal a különbséggel, hogy ebben az esetben csak a fel, le, jobbra és balra mozgásokkal dolgoztam.

4.2.3 EpocRTC

Ahogyan az az ábrán is látszik, ebben az esetben az általam készített komponens közvetlenül a vezérelni kívánt CyberDevice-hoz kapcsolódik, és teljesen független a VirCA fő komponenstől. Mivel VirCA porttal nem rendelkezik, így már magának a VirCA rendszernek nem a része, azonban az alatta működő RTM hálózatnak még igen. Ebből kifolyólag az EpocRTC egy teljes értékű RT komponens, aminek működése követi az OMG RTC életciklust [19]:



4-4. ábra: RT komponensek életciklusa

A komponens létrehozása után Alive állapotba kerül, ami maga is több al-állapotra bontható. Egyrészt, mint szál kezelve a komponenst, az lehet Stopped és Running. Előbbi esetben semmilyen belső logika vagy funkcionalitás nem is fut, maga a szál van leállítva.

Másrészt, a komponens lehet Inactive, Active illetve Error állapotban, amiből számomra az Active állapot a lényeges: ebben végzi a feladatát.

A bemenet kezelése az avatár irányításos esettel megegyezik: külön szálon fut maga az RT komponens, és külön szálon folyik a bemenetek kezelése.

4.2.4 Implementáció

OpenRTM.NET használatakor egy DataFlowComponent ősosztályból származtathatjuk a komponensünket, amihez definiáljuk a megfelelő portokat:

```
[Component(Category = "BCI", Name = "EpocRTC")]
public class CallbackOut : DataFlowComponent
{
    [OutPort(PortName = "out")]
    OutPort<TimedPoint3D> outport = new OutPort<TimedPoint3D>();
```

Attribútumokként adhatjuk meg a komponensünk nevét, és a port típusát és nevét is. Maga a port egy generikus osztályból példányosítható, jelen esetben TimedPoint3D típussal, ami az RTM névtér része, és a ColoredBall bemeneti portja is ilyen típusú.

Mivel jelenleg maga a komponens csupán a kommunikációért felelős, így semmi komolyabb inicializálásra nincs szükség, csupán a DataFlowComponent OnExecute metódusának felüldefiniálására:

```
protected override ReturnCode_t OnExecute(int execHandle)
{
    this.SetExecutionRate(execHandle,100);

    TimedPoint3D data = new TimedPoint3D();
    data.Time.SetCurrentTime();
    data.Data.X = X;
    data.Data.Y = Y;
    data.Data.Z = Z;

    outport.Write(data);
    return ReturnCode_t.RTC_OK;
}
```

Amiben is a belső állapotnak megfelelő koordinátákat beletöltjük egy időbélyeggel ellátott vektorba, majd azt a portra írjuk.

Az implementált komponenst egy Manager nevű objektum fogja létrehozni:

```
Manager manager = new Manager(args);
manager.AddTypes(typeof(CorbaProtocolManager));
manager.Activate();
var comp = manager.CreateComponent<CallbackOut>();
```

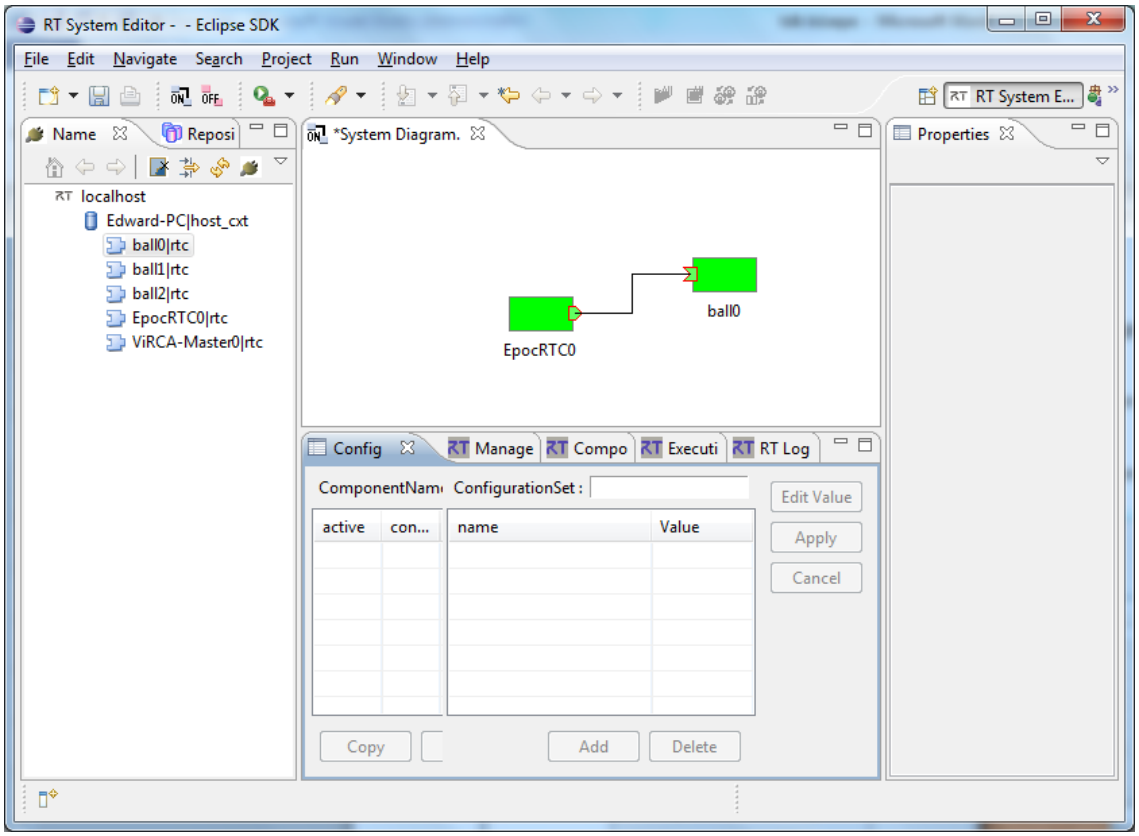
Aminek a konfigurációja egy rtc.config állományban található, ami egy XML alapú konfigurációs fájl, ahol többek között a névszerver címét kell megadnunk:

```
<?xml version="1.0"?>
<RtcSettings xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Setting Name="protocol.type">CORBA</Setting>
  <Setting
Name="protocol.CORBA.naming">195.111.2.139:2809</Setting>
  <Setting Name="exec.rate">1</Setting>
  <Setting Name="logger.type">log4net</Setting>
  <Setting Name="naming.formats">%h.host_cxt/%n.rtc</Setting>
</RtcSettings>
```

Mindezt egy triviális fizikai szimuláció egészíti ki, ami alapján, ha egyik irányt jelentő billentyű sincs lenyomva, akkor a koordináták konvergálnak a kiindulási ponthoz.

4.2.5 Összekapcsolás

Az elkészült komponenst, mivel nem valódi VirCA komponens, egy Eclipse alapú RT System Editorral kapcsoltjuk össze a ColoredBall CyberDevice-cal:



4-5. ábra: Eclipse System Editor

Jól látható, hogy az elérhető komponensek között szerepel a VirCA fő komponens is, hiszen minden VirCA CyberDevice egyben RT komponens is, viszont a System Diagram-on is látszik, hogy a ColoredBall-nak csak az RT adatportja látszik, a VirCA portja nem, mivel az már nem része az RTM szabványnak. Természetesen emiatt a teljes rendszer felépítéséhez a VirCA System Editort is használni kellett.

4.3 A rendszer alkalmazása

Elkészítettem tehát két teljes értékű interfészt, ami a kiválasztott BCI eszközt a virtuális valósággal összekapcsolja. Ezek alkalmazásával irányíthatjuk a virtuális avatárunkat, bejárhatjuk a virtuális világot, illetve a kiválasztott virtuális objektumot, a ColoredBall-t is mozgathatjuk pusztán mentális parancsok használatával. Ezek a funkciók úgy lettek kialakítva, hogy a Control Panel tanítása közben is aktívak legyenek, így a tanítási fázisban már eleve élesben, a rendszert használva kalibrálhatjuk a Control Panelt, így olyan mentális parancsokat választhatunk, amiket könnyedén aktiválni tudunk a VirCA rendszer használata közben is.

4.3.1 Rendszerek használata

Az elkészült alkalmazásokat a következő módon használhatjuk:

1. Emotiv Control Panel indítása és konfigurálása (4.2.2. fejezet)
2. Emokey indítása és konfigurálása (5.1.3 fejezet)
3. VirCA indítása
    a. Névszerver indítása
    b. VirCA fő komponens indítása
    c. CyberInputAdapter indítása
    d. ColoredBall indítása

4. CyberDevice-ok összekapcsolása a System Editor használatával
5. EpocAdapter indítása
6. EpocRTC indítása
7. EpocRTC és ColoredBall összekapcsolása System Editor használatával

Ezzel a rendszer működésre kész, a két vezérlési mód között úgy tudunk váltani, hogy az EmoKey-ben a megfelelő alkalmazást (EpocAdapter vagy EpocRTC) választjuk ki célalkalmazásnak az üzenetek küldéséhez.

4.3.2 Rendszer értékelése

Három független felhasználóval teszteltem az elkészített rendszert, mind egyszerű asztali számítógép, mind CAVE segítségével. Maga a használat nem okozott különösebb problémát egyiküknek sem, képesek voltak mind a virtuális térben való mozgásra, mint a labda mozgatására.

Két jelentősebb probléma merült azonban fel. Egyrészt igen körülményes volt az Emotiv Control Panel kalibrálása, amit mindenképpen a VirCA rendszeren kívül kellett elvégezni, ami a CAVE esetében kifejezetten zavaró volt, mivel a felhasználónak a kijelölt területen kellett maradnia, így a Control Panel-t futtató számítógépet másnak kellett kezelnie. Ráadásul a kalibrálási folyamat is igen hosszadalmas, és bizonyos esetekben nem is sikerült mind a négy utasítást kellő megbízhatósággal betanítani.

Másik probléma a vezérlés dimenzióinak alacsony száma volt. Egy ilyen immerzív környezetben természetes, hogy a felhasználó nem csak horizontálisan, hanem vertikálisan is szeretne elfordulni.

Ezek a problémák azonban nem a megvalósítás hibái, hanem az alkalmazott EmoEngine hiányosságaiból erednek.

5 EmoEngine kiváltása

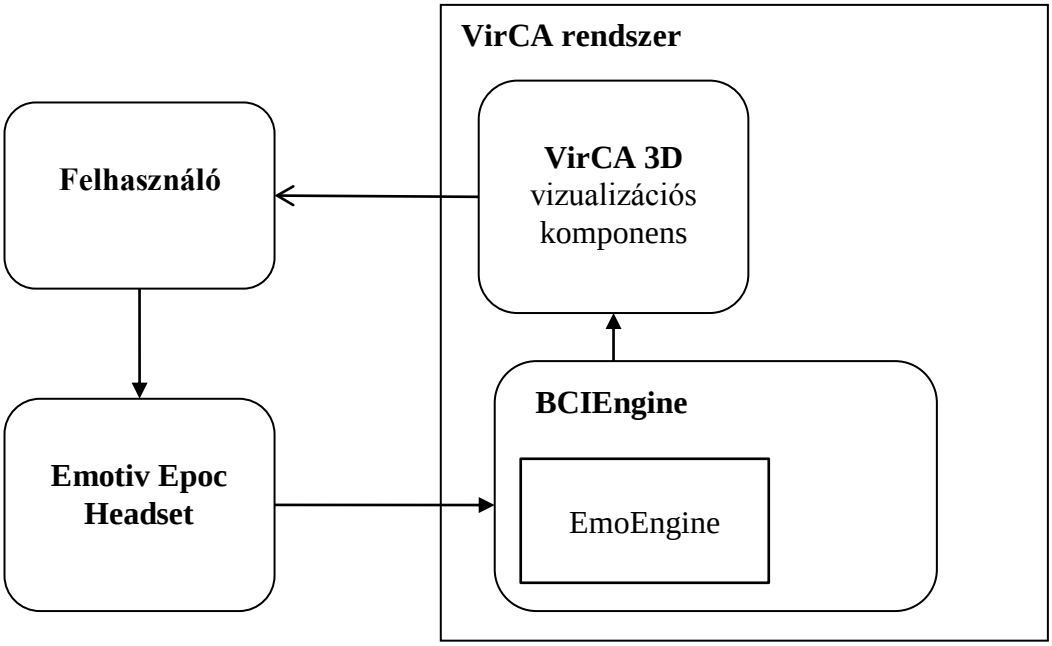
Célom tehát a fentebb említett problémák orvoslása volt. Egyik ezek közül, hogy az EmoEngine kalibrálását a VirCA-n kívül kell elvégezni. Természetesen a rendszert összeállíthatjuk, és a tényleges VirCA vizualizációt használva végezhetjük a tanítást: például kalibrálhatjuk az előrehaladást arra a gondolatra, ahogy elképzeljük, amint a virtuális térben előrefele mozdulunk, azonban a Control Panel felületét nem hagyhatjuk el. Ez nehézkessé teszi az amúgy is hosszú tanítási folyamatot. Egy lehetséges megoldás lenne erre, ha az SDK használatával az EmoEngine tanítási és profilkezelési funkcióit beépítenénk egy VirCA CyberDevice-ba, így a kalibrálás során csak a VirCA rendszert kellene használni. Minthogy ez tisztán programozási feladat, ráadásul a hosszadalmas betanítást sem kezelné, inkább más megoldást kerestem.

Másik probléma az irányítás dimenzióinak alacsony száma. A jelenlegi megoldás egyszerre maximum négy gondolati utasítást képes kezelni. Ennek megoldására számos lehetőséget megvizsgáltam. Egyrészt kiterjeszthetném az EmoEngine Suite-jainak használatát, így például arcmimikára, fejmozgásra vagy érzelmi állapotra is lehetne parancsokat aktiválni. Ezt a megoldást elvetettem, mivel hosszú távú célom egyértelműen a nem-motorikus, pusztán kognitív vezérlések tanulmányozása. Másik lehetőség lett volna a kontextus-érzékeny utasítások használata: a maximálisan használható négy parancs a rendszer állapotától függően más és más utasításra képződik le. Így például, ha menübe lépünk, akkor az abban való navigálásra, míg onnan kilépve magában a Virtuális térben való mozgásra használhatjuk a headsetet. Ez a módszer mindenképpen alkalmazandó, mivel így nagyságrendekkel csökkenteni lehet a szükséges különböző mentális parancsok számát, ami növeli a rendszer teljesítményét.

Mindkét problémára megoldást jelentene, ha az EmoEngine Suite-jait kihagyva, magát a motort csak a nyers jelekhez való hozzáférésre használva egy teljesen egyedi jelfeldolgozó modult készítenék. Ebben nem kellene limitálni az egyszerre aktiválható parancsok számát négyre, és a kalibrálási folyamat sem lenne a Control Panelhez kötve. Ezt a megoldást választottam.

5.1 Architektúra

Elsőnek csak a VirCA fő komponens irányítását tűztem ki célul: az avatár irányítását, illetve a menüben való navigálást (korábban említett MenuManipulation interfész). Ez utóbbival akár egyéb objektumokat is vezérelhetünk, mivel a VirCA rendszerben bármilyen CyberDevice-nak lehet előhívható menüje.



5-1. ábra: BCIEngine architektúra

Ennek fényében az új komponens, a BCIEngine mindenképpen kapcsolatban kell legyen közvetlenül a VirCA fő komponenssel. Így már nincs szükség a korábbi CyberInputAdapter használatára, ami amúgy sem kezeli a MenuManipulation interfészt. Továbbá, mivel a nyers szenzoradatokra van szükségem, így mindenképpen az EmoEngine SDK-t kell használnom, mivel erre egyik elkészített alkalmazás sem ad lehetőséget.

5.2 BCIEngine

Az általam megtervezett CyberDevice-t neveztem el BCIEngine-nek. Mivel ez egy szabványos VirCA CyberDevice, így természetesen örökli a korábban már említett RT komponensek életciklusát. Ezen felül, a következő feladatokat kell, hogy elvégezze:

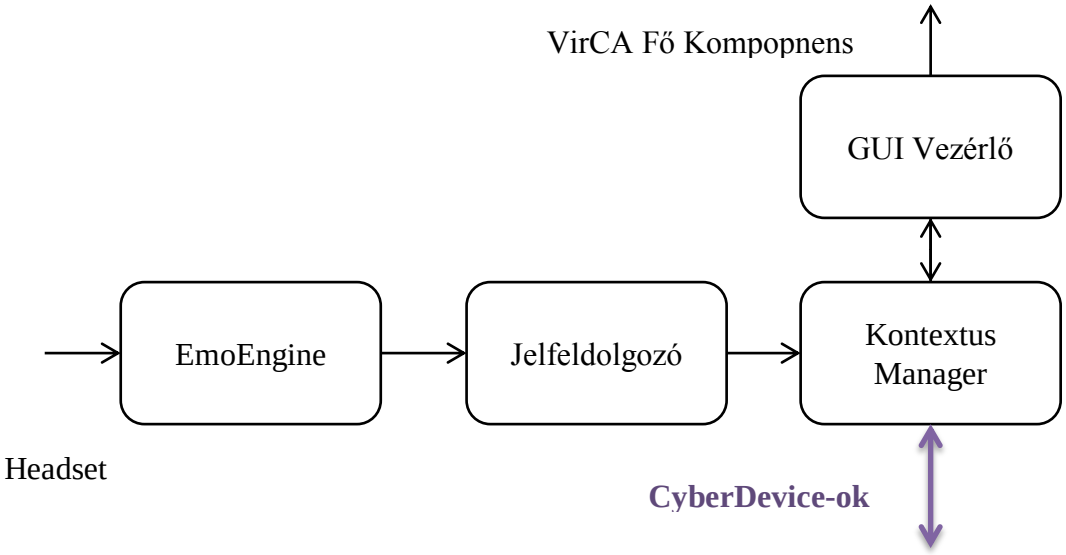
a) Emotiv EPOC headset-től a nyers szenzoradatok lekérdezése
b) Vizuális stimulusok kezelése
c) Jelfeldolgozás a szinkronizált EEG jelek és ingerek állapota alapján
d) Kontextus kezelése

Opcionálisan:

e) Interfész felhasználó által definiált parancsok és a hozzájuk tartozó ingerek kezelésére.

5.2.1 BCIEngine felépítése

A fentiek alapján a következő architektúrát alakítottam ki:



5-2. ábra: BCIEngine felépítése

EmoEngine - Szenzoradatok lekérdezése

Az Emotiv SDK Research Edition-ja lehetőséget nyújt a headset által mért nyers szenzoradatok lekérdezésére. Ezt az EmoEngine használatával érhetjük el. Először szükségünk van egy EmoEngine objektumra:

```
EmoEngine engine; // Access to the EDK is via the EmoEngine
```

Ezt példányosítjuk, majd a UserAdded és UserRemoved eseményre feliratkozunk: ezek jelzik a Headset csatlakoztatását illetve leválasztását, majd csatlakozunk az EmoEngine-nel a headset-re:

```
// create the engine
engine = EmoEngine.Instance;
engine.UserAdded += new EmoEngine.UserAddedEventHandler(
  engine_UserAdded_Event);
engine.UserRemoved += new EmoEngine.UserRemovedEventHandler(
  engine_UserRemoved_Event);

// connect to Emoengine.
engine.Connect();
```

Az engine dokumentációja szerint 128 Hz-es mintavételezéssel dolgozik. Ahhoz, hogy ezt hatékonyan kihasználhassuk, egy belső buffer áll a rendelkezésünkre, és mi adhatjuk meg ennek a méretét, majd gondoskodunk róla, hogy sose teljen meg. A headset csatlakoztatásakor megtehetjük ezt, majd elmentjük az engine által generált userID-t, végül engedélyezzük az adatgyűjtést:

```
void engine_UserAdded_Event(object sender, EmoEngineEventArgs e)
{
      // record the user
      userID = (int)e.userId;

      // ask for up to 1 second of buffered data
      engine.EE_DataSetBufferSizeInSec(1);
```

```
        // enable data aquisition for this user.
        engine.DataAcquisitionEnable((uint)userID, true);
    }
```

Egy másodperces bufferméretet választottam, és az adatok elkérését 500 ms-enként végzem majd. Ezáltal a rendszer kellően gyors reagálású lesz, viszont a Windows időzítési problémái még nem jelentkeznek.

Headset leválasztásakor csak leállítjuk az adatgyűjtést:

```
void engine_UserRemoved_Event(
   object sender, EmoEngineEventArgs e)
{
        // disable data aquisition for current user.
        engine.DataAcquisitionEnable((uint)userID, false);

        // reset the user
        userID = -1;
}
```

Ezután csak arról kell gondoskodnunk, hogy az adatokat elkérjük minden 500 ms-ben. Erre egy Timer-t hoztam létre, ami a megadott időközönként meghívja az alábbi metódust:

```
public void Run(Object stateInfo)
{
  try
  {
    engine.ProcessEvents();
    if (userID != -1)
    {
      Dictionary<EdkDll.EE_DataChannel_t, double[]> data =
        engine.GetData((uint)userID);

      //Process the acquired data

    }
  }
  catch (Exception ex)
  {
    // Error Handling
  }
}
```

Mint látható, az EmoEngine eseménykezelőjét nekünk kell meghívnunk. Ezután a GetData() metódussal hozzáférhetünk a bufferben tárolt adatokhoz. Ez egy Dictionary<EdkDll.EE_DataChannel_t, double[]> típusú adatstruktúrát ad vissza, amiben minden egyes csatornához annyi dupla pontosságú lebegőpontos érték tartozik, ahány rekord van a bufferben.

A csatornák listája a következő:

ED_COUNTER	ED_FC6
ED_AF3	ED_F4
ED_F7	ED_F8
ED_F3	ED_AF4
ED_FC5	ED_GYROX
ED_T7	ED_ GYROY
ED_P7	ED_TIMESTAMP
ED_O1	ED_FUNC_ID
ED_O2	ED_FUNC_VALUE
ED_P8	ED_MARKER
ED_T8	ED_SYNC_SIGNAL

5-1. táblázat: EmoEngine csatornák

Ezek között számos, számomra lényegtelen adat is megtalálható, így a jelfeldolgozó csak az ED_AF3-tól az ED_AF4-ig terjedő, ténylegesen EEG értékeket, és az ED_COUNTER számlálót, illetve az ED_TIMESTAMP időbélyeget fogja használni.

GUI vezérlő - Vizuális stimulusok kezelése

A felhasználó számára látható felület a VirCA fő komponens által megjelenített kép. A manapság alkalmazott BCI módszerek általában a kiváltott potenciálokon alapulnak (lásd 6.3. fejezet), amik pedig valamilyen inger (legtöbbször vizuális) generálását igényli. A BCIEngine feladata tehát biztosítani a fő komponens számára, hogy az a megfelelő időben felvillanthassa az ingereket kiváltó ikonokat. Ezt a GUI (Graphical User Interface) Vezérlő fogja végezni.

Jelenleg a VirCA rendszerben egyféleképpen lehet grafikus vizualizációt rendelni egy CyberDevice-hoz: egy 3D modell-t regisztrálunk be a VirCA service-porton keresztül a fő komponensnek, amit utána mozgathatunk, illetve animálhatunk. Ez a megoldás nem éppen előnyös az elkészítendő BCIEngine számára, mivel a klasszikus esetben ikonok felvillantását kell megvalósítani. Ezt ki lehet kerülni azzal, ha az ikonokat, mint 3D-s objektumokat készítjük el, amiket mindig a kamera nézőpontjához igazítva jelenítünk meg, a felvillanó hatást pedig különböző textúrázással érjük el. A VirCA rendszerhez szerencsére fejlesztés alatt áll egy overlay-elemeket támogató GUI, ami ezt a problémát kényelmesen megoldaná, azonban ez a dolgozat írásának idejében még nem elérhető.

A GUI vezérlő másik feladata a BCIEngine menüjének, funkcióinak megjelenítése. A VirCA rendszerben erre jelenleg egyetlen mód van: valamilyen grafikus objektumot, egy „szobrot” kell beregisztrálni a virtuális valóságba, és azon keresztül hívhatjuk elő a CyberDevice menüjét. Ez a módszer kényelmetlen, mivel a szobor elhelyezése virtuális tér-függő, ráadásul, ha a felhasználó nem tartózkodik annak a közelében, nem érhető el a menü. Ennek a problémának két megoldásával foglalkoztam.

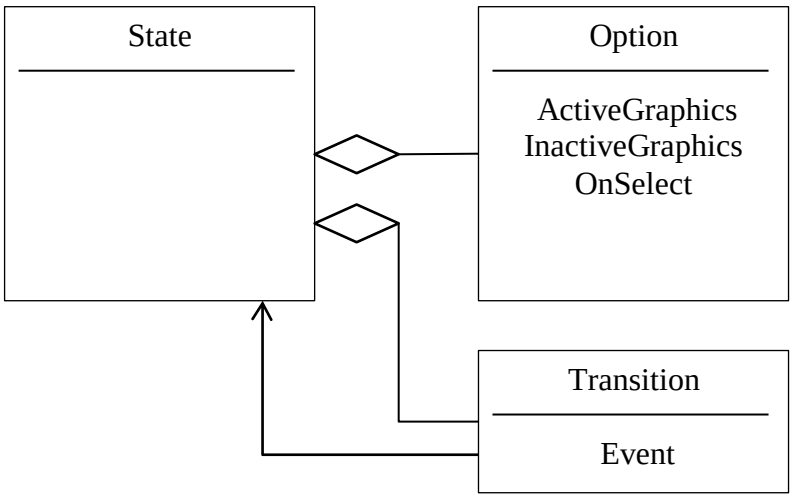
Egyrészt megtehetjük, hogy az ikonokhoz hasonlóan, a felhasználó nézőpontjához igazítjuk a szobrot, ezáltal, mint egy újabb ikont implementáljuk. A pointer megfelelő mozgatásával ezt is aktiválni tudjuk egyéb bemeneti eszközökkel, így

a BCIEngine kezdeti kalibrálása megoldható. Emiatt viszont célszerű a többi ikont is ily módon aktiválhatóvá tenni, hogy egységesen kezelhessük őket.

A másik módszer alapja jelenleg még fejlesztés alatt áll. A VirCA főmenüjéhez terveznek ugyanis delegálási lehetőséget biztosítani, ezáltal más CyberDevice-ok is beregisztrálhatnak almenüket, így a szobros kényszer-megoldásokra nem lenne szükség.

Kontextus Manager

A grafikus megjelenítéstől függetlenül, mindenképpen szükséges kezelni, hogy adott időben milyen opciók érhetőek el, azok ikonjai hol helyezkednek el, illetve hogyan néznek ki. Ennek nyilvántartása lesz a Kontextus Manager feladata, amit a következő adatstruktúrában fogja tárolni:



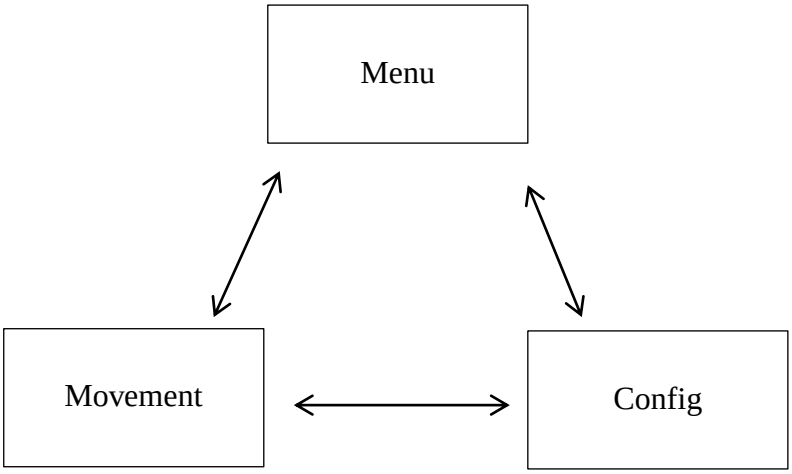
5-3. ábra: Kontextus Manager állapotai

Különböző állapotokat definiálok, és minden állapothoz tartoznak elérhető opciók, illetve állapotátmenetek. Minden opciónak tartalmaznia kell információt arról, hogyan kell megjeleníteni a VirCA fő komponensnek aktív (éppen felvillanó), illetve inaktív állapotban. Ezen felül minden opciónak van egy Callback metódusa, ami meghívódik, ha az adott opció kiválasztását érzékeli a rendszer.

Az állapotátmenetek mindenképpen eseményekhez vannak kötve, és meghatározzák azt is, hogy melyik másik állapotba kerülünk az esemény bekövetkezésekor.

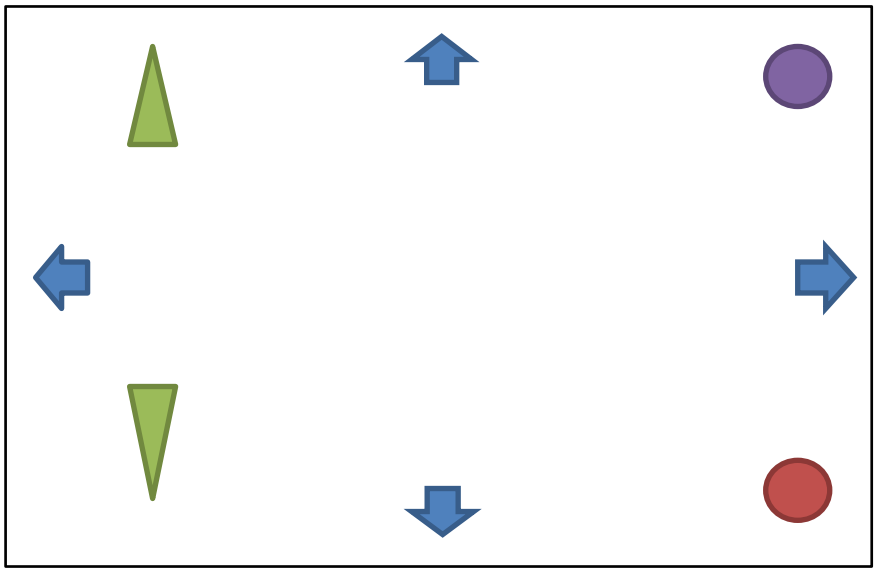
Állapotok

A rendszer tervezésekor három állapotot definiálok: egyet az alapértelmezett, virtuális térben való mozgáshoz (Movement), egyet a menüben való navigáláshoz (Menu), és egyet a konfigurációhoz (Config).



5-4. ábra: BCIEngine állapotai

Movement állapotban elérhető opciók az avatár irányítása, a menübe lépés, illetve a saját menü meghívása. A menübe lépés a Pointerrel aktuálisan kiválasztott CyberDevice menüjét nyitja meg, míg a saját menü meghívása mindig a BCIEngine funkcióit éri el. Ezeknek tervezett helyét a következő ábra mutatja:



5-5. ábra: BCIEngine tervezett ikonkészlete

A kékkel jelölt nyilak a 4 irányba történő kameraforgatás, a zölddel jelöltek az előre illetve hátrafele haladás, míg a lila kör az aktuális CyberDevice menüjébe, míg a piros kör a saját menübe lépés ikonjai. Ebből az állapotból az aktuális CyberDevice menüjébe lépés hatására kerül át a rendszer a Menu, míg a konfigurációs ikonon keresztül a Config állapotba.

Menu állapotban csak a kékkel, lilával és pirossal jelölt ikonok elérhetőek: a jobbra-balra nyilak a menüben való jobbra illetve balra lépést, a lefele nyíl az almenübe

lépést/kiválasztást, a felfele nyíl a visszalépést, a lila kör a menüből való kilépést (Movement állapotba visszatérést), a piros a Config állapotba lépést jelenti.

Config állapotban a BCIEngine beállításai érhetőek el a szabványos VirCA menürendszeren keresztül. Ekkor a jelfeldolgozást leállítom, hogy az esetlegesen rosszul beállított rendszer ne okozhasson problémát.

Fontos, hogy a kontextus váltásnak nem csak a BCIEngine által érzékelt parancsoktól kell függenie, mivel elképzelhető, hogy más bemeneti eszközzel aktiválja az ikonokat a felhasználó.

A Kontextus Manager feladata továbbá, hogy a Jelfeldolgozó modul kimenete, és a GUI állapota alapján megállapítsa, melyik parancsot akarta a felhasználó aktiválni, és annak a Callback függvényét meghívnia.

5.3 Jelfeldolgozó

A modul feladata a beérkező EEG jelek feldolgozása. Első lépésként utánanéztem, milyen módszerekkel dolgoznak a BCI-ok területén. A legtöbb ilyen módszert a vizsgált agyi tevékenységről nevezték el [20].

a) β és μ hullámok

Ezek az elektromos tevékenységek a 8-12 Hz (Mu) illetve a 12-30 Hz (Beta) tartományban figyelhetőek meg. A motoros tevékenységhez leginkább kapcsolódó jelek, de lehetséges a szándékos aktiválásuk is, például képzeletbeli mozgásvégzéssel. Számos BCI kutatásban használták már ezeknek a jeleknek a változását, mint megfigyelt műtermékek. Bár az EmoEngine algoritmusai nem publikusak, véleményem szerint főleg ezen az elven működhet, mivel a legtöbb másik módszer (az SCP-t leszámítva) valamilyen kiváltó ingert igényel, ami a rendszerben nem található. [21]

b) P300 kiváltott potenciál

A P300 a hullámkomponens az ERP-k (Event Related Potentials) csoportjába tartozik, és valamilyen vizuális, hangalapú vagy szomatoszenzorikus (pl. tapintás, hőérzet) inger hatására jelenik meg. Az inger után, kb 300 ms-el jelentkezik, és legerősebben ritka, de várt események hatására aktiválódik. Ennek a tulajdonságának köszönhetően gyakran alkalmazzák különböző ingerek közötti választás esetén, például betűzésre, ahol az egyes betűk felvillanásai az ingerek, és a kiválasztott betű a várt esemény, az úgynevezett target-stimuli. [22],[23]

c) VEP

A VEP (Visually Evoked Potentials) kis késéssel jelentkező agyi tevékenység gyors vizuális ingerlésre. Jelalakjukat egy 100 ms késésű negatív (N100) csúcs jellemzi, amit követ egy 200 ms késésű pozitív (P200). A 70-es évek óta használják ezeket annak a megállapítására, hogy melyik irányba néz az alany. [24]

d) SSVEP

Az SSVEP (Steady-State Visually Evoked Potentials), a P300-hoz hasonlóan szintén a kiváltott potenciálok családjába tartozik. Ezek a jelek természetes válaszok különböző frekvenciájú vizuális ingerekre. Ha a retinát 3.5 és 75 Hz közötti frekvencián ingerlik, például ikonokat villogtatnak, az agy elektromos aktivitást generál megegyező (vagy egész számú többszörös) frekvencián. Ezt használják arra, hogy megállapítsák, az alany melyik ingerre reagál, például különböző frekvencián villogó ikonok közül. [25],[26]

e) SCP

Az SCP (Slow Cortical Potentials) lassú, akár 10 másodperces késleltetésű jelek, amiket kellő képzés és gyakorlás után az alanyok képesek lehetnek befolyásolni, ezáltal például kurzort mozgatni. Elképzelhető, hogy az EmoEngine ezen az elven működik, azonban az SCP esetlegesen igen nagy késését nem tapasztaltam, így valószínűbbnek tartom a béta és mű hullámokon alapuló algoritmusokat. [27]

A fentebb említett paradigmák közül a P300-at és SSVEP-et ítéltem alkalmasnak a rendszerhez. A többi nem orvosolná rendesen a hosszas kalibrálás problémáját, mivel az alany tanulásán alapulnak, azaz a felhasználónak kell megtanulnia produkálni valamilyen jellemzőjű agyi tevékenységet. Ezzel szemben a P300 és SSVEP, sőt alapvetően az ERP-k csoportjába tartozó módszerek esetén, a vizsgált jelenség mindenkinél előfordul, ráadásul az Emotiv EPOC headset használatával implementáltak már P300 paradigmán alapuló rendszert [28].

Azonban a BCI paradigmák tanulmányozása során világosan kiderült, hogy ezek hatékony alkalmazása egészen más jellegű feladat, mint az eddigiek. Eddig egy „fekete doboz"-szerű jelfeldolgozónak a már értelmezett parancsaival dolgoztam, azaz inkább alkalmazás-szinten fejlesztettem az Emotiv által elkészített eszközökkel. Saját tanuló- és jelfeldolgozó motor írásához azonban kiterjedtebb jelfeldolgozási- és neurobiológiai ismeretek kellenek, amik megszerzésén jelenleg is dolgozok.

Emiatt munkámat rendszer-szintű megközelítésben folytattam. Kialakítottam egy shellt, amibe a későbbiekben elkészített jelfeldolgozó motor minimális munkával beilleszthető. Felkészítettem a BCIEngine-t a megfelelő időzítéssel történő vizuális ingerek generálására, hogy a kiváltott potenciál alapú paradigmák hatékonyan alkalmazhatóak legyenek.

5.4 BCIEngine értékelése

Munkám során megterveztem az EmoEngine-t kiváltó BCIEngine komponenst, amivel a korábbi problémák kezelhetőek lesznek. A felismerhető parancsok maximális számát testre szabhatom, a hosszú és körülményes kalibrálási folyamatot pedig elhagyhatom.

Kialakítottam egy vázat, amibe a későbbiekben fejlesztett (P300 vagy SSVEP alapú) jelfeldolgozó motor beilleszthető, így az EEG jelek feldolgozását saját algoritmusaimmal végezhetem. Ezzel teljesen kiküszöbölöm a korábbi „fekete doboz"-problémát, azaz az Emotiv EPOC headset-től érkező adatok végig általam megtervezett csatornákon keresztül kerülnek majd értelmezésre, megnyitva ezzel az utat a jelek mélyebb tanulmányozásának.

6 Összegzés

Dolgozatomban bemutattam, hogy az Emotiv EPOC BCI eszköz, és a VIRCA virtuális valóság integrációja megvalósítható. Megterveztem, implementáltam és teszteltem két teljes értékű interfészt, amivel a VirCA legfontosabb funkcióit vezérelni lehet. Identifikáltam a VirCA vezérlésének lehetőségeit, megvalósítottam avatár-vezérlést, amivel a virtuális világot bejárhatjuk, illetve egy virtuális objektum irányítását, amivel egy labdát mozgathatunk a szimulált térben. Ez elkészült rendszereket alapos tesztelésnek vetettem alá, és több felhasználó véleménye alapján értékeltem.

A tesztelés során felmerülő főbb problémák okának az EmoEngine használatát azonosítottam. Ez alapján, végül előkészítettem a további kutatásaim alapját, megterveztem és megvalósítottam egy alkalmazást, ami kiváltja az EmoEngine-t, és felkészítettem a manapság legelterjedtebb BCI paradigmák alkalmazására. Ennek segítségével a későbbiekben kizárólag az EEG jelek feldolgozásával kell majd foglalkoznom, mivel a rendszer az többi komponenssel való kapcsolattartást, felhasználói felületet már megvalósítja.

Az agy-számítógépek interfészek szemszögéből vizsgálva tehát létrejött egy tesztkörnyezet, amiben virtuálisan bármilyen alkalmazási területen megvizsgálhatjuk a BCI-ok működését. A rendszerrel könnyedén kezelhetőek a kiváltott potenciál-alapú paradigmák vizuális stimulusai, és az immerzív környezet megnöveli az eszközök teljesítményét.

Virtuális valóságok szempontjából pedig egy új bemeneti eszközt készítettem, ami nem igényel motorikus funkciót, így akár fogyatékkal élők is használhatják. Megalapoztam továbbá egy szabványos interfészt, amivel a virtuális tér tetszőleges objektuma vezérelhetővé válik, ezáltal egy szabadabb, lehetőségekben gazdag komponenssel bővítettem a rendszert.

7 Irodalomjegyzék

[1] G. Pfurtscheller, R. Leeb, J. Faller, és C. Neuper: *Brain-Computer Interface Systems Used for Virtual Reality Control*, Virtual Reality, J.-J. Kim, Ed. InTech, 2011

[2] g.Tec BCI-VR rendszere, a g.VRsys leírása: http://www.gtec.at/Products/Hardware-and-Accessories/g.VRsys-Specs-Features, letöltve: 2012.10.22.

[3] E. Niedermeyer és F. L. da Silva: *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, 2004

[4] R. W. Homan, J. Herman, és P. Purdy: *Cerebral location of international 10–20 system electrode placement*, Electroencephalography and Clinical Neurophysiology, vol. 66, no. 4, pp. 376–382, 1987

[5] D. S. Sanei és J. A. Chambers, *EEG Signal Processing*, John Wiley & Sons, 2007

[6] H. W. Agnew, W. B. Webb, és R. L. Williams, *THE FIRST NIGHT EFFECT: AN EEG STUDYOF SLEEP*, Psychophysiology, vol. 2, no. 3, pp. 263–266, 1966

[7] C. Cruz-Neira, D. J. Sandin, és T. A. DeFanti, *Surround-screen projection-based virtual reality: the design and implementation of the CAVE*, Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp. 135–142, New York, NY, USA, 1993

[8] Galambos Péter „VR kutatóintézetek”, személyes beszélgetés, 2012. Október 22.

[9] Mediterranean Virtual Reality Center honlapja: http://139.124.68.168/en/center-main-features-crvm, letöltve: 2012.10.22.

[10] VR-lab honlapja, Max Planck Institute for Psycholinguistics: http://www.mpi.nl/resources/labs/vr-lab, letöltve: 2012.10.22.

[11] 3D Lab honlapja, University of Michigan: http://um3d.dc.umich.edu/, letöltve: 2012.10.22.

[12] Visionarium honlapja, Virginia Tech: https://snoid.sv.vt.edu/visionarium/?page_id=698, letöltve: 2012.10.22.

[13] Corba honlapja: http://www.corba.org/, letöltve: 2012.10.22.

[14] RT Middleware leírása: http://www.openrtm.org/openrtm/en/content/what-rt-middleware-0, letöltve: 2012.10.22.

[15] openRTM-aist leírása: http://www.openrtm.org/openrtm/en/content/what-openrtm-aist, letöltve: 2012.10.22.

[16] ICE technológiai áttekintő: http://www.zeroc.com/overview.html, letöltve: 2012.10.22.

[17] Emotiv EPOC neuroheadset honlapja: http://www.emotiv.com, letöltve: 2012.10.22.

[18] VirCA Virtuális Kollaborációs Aréna honlapja: http://www.virca.hu, letöltve: 2012.10.22.

[19] RT komponensek életciklusa, felépítése: http://www.openrtm.org/openrtm/en/content/programming-rt-component-0, letöltve: 2012.10.22.

[20] Fabrizio Beverina, Giorgio Palmas, Stefano Silvoni, Francesco Piccione, és Silvio Giove: *User adaptive BCIs: SSVEP and P300 based interfaces*, PsychNology Journal, vol 1, no. 4, pp.331 – 354, 2003

[21] Pfurtscheller G.: *Functional topography during sensomotor activation studied with event-related desynchronization mapping*, J. Clin. Neurophysiol, vol. 6, pp. 75-84, 1989

[22] Farwell L.A. és Donchin E.: *Talking off the top of your head: toward a mental prothesis utilizing event-related brain potentials*, Electroencephalogr clin Neurophysiol, vol. 70, pp. 510-523, 1988

[23] Donchin E., Spencer K.M. és Wijesinghe R.: *The mental prothesis: assessing the speed of a P300-based brain-computer interface*, IEEE Trans. Rehabil. Eng., vol. 8, pp. 174-179, 2000

[24] Vidal, JJ.: *Direct brain-computer communication*, Ann. Rev. Biophys Bioeng, vol. 2, pp. 157-158, 1973

[25] Morgan S.T., Hansen J.C. és Hillyard S.A.: *Selective attention to stimulus location modulates the steady-state visual evoked potential*, Neurobiology, vol. 93, pp. 4770-4774, 1996

[26] Muller M.M. és Hillyard S.A.: *Effects of spatial selective attention on the steadystate visual evoked potential in the 20-28 hz range*, Cognitive Brain Research, vol. 6, pp. 249-261, 1997

[27] Birbaumer N., Elbert T. Canavan A.G.M. Roch B.: *Slow potentials of the cerebral cortex and behaviour*, Physiol. Rev., vol. 70, pp. 1-4, 1990

[28] Ivan Martinovic, Doug Davies, Mario Frank, Daniele Perito, Tomas Ros és Dawn Song: *On the Feasibility of Side-Channel Attacks with Brain-Computer Interfaces*, USENIX Security '12, 2012