



GÁBOR DÉNES FŐISKOLA

Mobile-RC

Diplomaterv sorszáma: 568/2012

Farkas Zoltán

Budapest

2012

Tartalomjegyzék

Előszó.....	5
1. Bevezetés.....	6
2. Felhasználói útmutató.....	9
2.1. Rendszerkövetelmények.....	9
2.2. Beszerzés.....	9
2.3. Telepítés Android rendszerre.....	9
2.4. Telepítés Windows rendszerre.....	10
2.5. Telepítés GNU/Linux rendszerre.....	11
2.6. Telepítés Mac OS X rendszerre.....	13
2.7. A vezérlőkliens használata.....	13
2.7.1. Az alkalmazás indulása.....	13
2.7.2. Az alkalmazás beállítása.....	14
2.7.3. Kapcsolódás a Hídszerverhez, jelszóbekérés.....	15
2.7.4. Kapcsolódáskezelő-ablak.....	15
2.7.5. A jármű kiválasztása.....	17
2.7.6. Járművezérlés, a főablak bemutatása.....	17
2.7.7. Járművezérlés, a jármű irányítása.....	18
2.7.8. Járművezérlés, a dialógusablakok bemutatása.....	19
2.8. A hídszerver használata.....	19
2.8.1. Az alkalmazás indulása.....	19
2.8.2. Az alkalmazás indítása Mac OS X alatt.....	20
2.8.3. Az alkalmazás paraméteres indítása konzolból.....	20
2.8.4. A Hídszerver konfigurációja.....	20
2.8.5. Engedélyek és prioritások.....	21
2.8.6. Tiltólista.....	21
2.8.7. Fehérlista és feketelista.....	21
2.8.8. Naplózás.....	22
2.9. A járműkliens használata.....	23
2.9.1. Az alkalmazás használatának előfeltételei.....	23
2.9.2. Kapcsolódás a Hídszerverhez.....	23
2.9.3. A járműkliens konfigurálása.....	25
3. Betekintés a háttérbe.....	27
3.1. Eszközválasztás.....	27
3.2. Az áramkör megalkotása.....	28
3.2.1. IOIO for Android.....	28
3.2.2. Kezdeti lépések.....	28
3.2.3. Az áramkör felhasználása.....	29
3.2.4. A nyomtatott áramkör megtervezése.....	30
3.2.4.1. Az első lépések.....	30
3.2.4.2. NyÁK-tervező program választása.....	30
3.2.4.3. A NyÁK kialakulása.....	30
3.2.5. A nyomtatott áramkör kivitelezése.....	31
3.2.5.1. Fototechnikás NyÁK készítés.....	31
3.2.5.2. Vasalásos NyÁK készítés.....	32
3.2.5.3. A NyÁK maratása.....	32
3.2.5.4. A NyÁK befejezése.....	32
3.2.6. A nyomtatott áramkör felhasználása.....	33
3.3. A szoftverek megalkotása.....	33
3.3.1. A hálózati kommunikáció felépítése.....	33

3.3.1.1. TCP Socket a Javaban.....	33
3.3.1.2. Eszköz- és kapcsolataazonosító.....	34
3.3.1.3. Titkosított kapcsolat és felhasználó azonosítás.....	34
3.3.1.4. Az általános terv.....	35
3.3.1.5. Az általános terv felhasználása.....	38
3.3.1.5.1. A kapcsolat ellenőrzése.....	38
3.3.1.5.2. Üzenetküldés és fogadás.....	38
3.3.1.5.3. Üzenetobjektumok: adatok, részadatok.....	39
3.3.1.6. A kamrakép közvetítése.....	41
3.3.1.7. Összegzés.....	42
3.3.2. A vezérlőkliens fontosabb elemei.....	42
3.3.2.1. A grafikus felület felépítése.....	42
3.3.2.1.1. Look And Feel (LAF).....	42
3.3.2.1.2. Standard Widget Toolkit (SWT).....	43
3.3.2.1.3. A térkép kivitelezése, Google Map.....	43
3.3.2.1.3.1. Swing versus SWT.....	44
3.3.2.1.3.2. Native Swing és bővítése.....	44
3.3.2.1.3.3. Az SWT betöltése, SWTJar.....	44
3.3.2.1.4. Töltőképernyő (Splash screen).....	46
3.3.2.1.5. A kamerakép megjelenítése, átméretezés (imgsclr).....	46
3.3.2.1.6. Az alkalmazás lokalizálása (ResourceBundle).....	47
3.3.2.2. A konfiguráció tárolása.....	48
3.3.2.2.1. Java Native Access.....	48
3.3.2.3. Összefoglalás.....	49
3.3.3. A járműkliens fontosabb elemei.....	49
3.3.3.1. Az Android platform.....	49
3.3.3.2. Az Android SDK.....	50
3.3.3.3. AndroidManifest.xml.....	51
3.3.3.4. Az alkalmazás főablaka, Activity.....	52
3.3.3.4.1. Az Activity életciklusa.....	53
3.3.3.4.2. SherlockActivity, ActionBarSherlock.....	54
3.3.3.5. Az alkalmazás konfigurációja, PreferenceActivity.....	55
3.3.3.5.1. SherlockPreferenceActivity, az ActionBarSherlock előnye.....	57
3.3.3.6. Nyelvi fájlok.....	57
3.3.3.7. Az alkalmazás háttérfolyamatai, Service.....	58
3.3.3.7.1. IOIO Service.....	58
3.3.3.7.2. Impulzusszélesség moduláció (PWM).....	59
3.3.3.7.3. A motor vezérlése.....	59
3.3.3.7.4. A telefon szenzorainak kezelése.....	60
3.3.3.7.4.1. Északi mágneses pólus, földrajzi észak.....	61
3.3.3.8. Összefoglalás.....	62
4. Befejezés: továbbfejlesztés.....	62
4.1. Hang továbbítása.....	62
4.2. Összes jármű a térképen.....	62
4.3. Játék mód útvonaltervezéssel.....	62
4.4. Egyedi karosszéria, nagyobb teljesítmény.....	62
Felhasznált források.....	63

Ábrajegyzék

1. ábra: Hálózati példa a vezérlő számítógép és a mobiltelefon kapcsolatára.....	7
2. ábra: Google Play.....	9
3. ábra: Automatikus lejátszás.....	10
4. ábra: Windows telepítő - kezdőképernyő.....	10
5. ábra: Windows telepítő - testreszabás.....	11
6. ábra: Windows telepítő - befejezés.....	11
7. ábra: Linux telepítő - kezdőképernyő.....	12
8. ábra: Linux telepítő - válaszadás.....	12
9. ábra: Linux telepítő - telepítés.....	12
10. ábra: OS X telepítő.....	13
11. ábra: Töltőképernyő.....	13
12. ábra: Kapcsolatbeállító-ablak 1.....	14
13. ábra: Kapcsolatbeállító-ablak 2.....	14
14. ábra: Névjegy és nyelvek.....	15
15. ábra: Tanúsítványjelszó bekérése.....	15
16. ábra: Kapcsolódáskezelő-ablak.....	15
17. ábra: Járműválasztó-ablak.....	17
18. ábra: Járművezérlő-főablak és dialógusablakai.....	17
19. ábra: Tájékoztató-üzenet.....	18
20. ábra: Térkép.....	19
21. ábra: Szerver hibaüzenet.....	19
22. ábra: Alkalmazásválasztó OS X alatt.....	20
23. ábra: Értesítési terület OS X alatt.....	20
24. ábra: A járműkliens.....	24
25. ábra: IP Webcam.....	24
26. ábra: Járműkliens beállítások 1.....	25
27. ábra: Járműkliens beállítások 2.....	26
28. ábra: Eszközök és kapcsolatuk.....	27
29. ábra: IOIO Board.....	28
30. ábra: BJT H-híd, feszültség-polaritásváltó áramkör.....	29
31. ábra: Bipoláris tranzisztor.....	29
32. ábra: Feszültségesztás.....	30
33. ábra: A nyomtatott áramkör.....	31
34. ábra: Process osztálydiagram.....	36
35. ábra: Handler osztálydiagram.....	37
36. ábra: DisconnectProcess és MessageProcess osztálydiagram.....	39
37. ábra: Data és PartialData osztálydiagram.....	40
38. ábra: Az Android rendszer felépítése.....	50
39. ábra: Az Activity életciklusa.....	54

Előszó

Tiszttel Olvasó!

Mielőtt belefogna az olvasásba, engedje meg, hogy bemutassam a szakdolgozatom felépítésének elvét és a benne használt jelölések értelmezésének módját.

A szakdolgozat két fő részből áll. Az első két fejezet nem csak az informatikában jártas olvasóknak szól. A bevezetés és a felhasználói útmutató könnyen értelmezhető módon írja le a diplomamunka célját, valamint az alkalmazások képességeit és azok használatát. A harmadik fejezetben találhatóak meg a mérnöki szempontból lényeges, háttérben meghúzódó működési elvek.

A szakdolgozatban két jelzést is használtam. Az egyik a szám alapú felső index, ami azt jelzi, hogy az oldal alján a lábjegyzetben az aktuális fogalom meg van magyarázva, vagy ki van bővítve. A másik jelzés a betű alapú felső index, ami az utolsó oldalon található végjegyzetet jelöli. A végjegyzetben a diplomamunka írása közben felhasznált források vannak feltüntetve. A betűk segítségével egyértelműbb, hogy melyik forrás, mit tartalmaz.

Ezúton szeretnék köszönetet mondani a családomnak, hogy támogattak miközben a diplomamunkámon dolgoztam. Köszönet a legjobb barátomnak is, hogy bemutatott bátyjának, aki útmutatást adott az áramköri rész megtervezéséhez. És végül köszönet a konzulensemnek is a türelméért és segítőkészségéért.

Kellemes olvasást kíván
Farkas Zoltán

1. Bevezetés

Diplomamunkám célkitűzése egy közönséges távirányítós-autó (a továbbiakban: jármű) átalakítása úgy, hogy azt számítógép segítségével Internetről és helyi hálózatról is biztonságosan lehessen vezérelni.

Ahhoz, hogy a járművet irányítani lehessen, a vezérő számítógépnek el kell tudnia érni a járművet és minden félnek egyazon „nyelvet” kell beszálnie, hogy értsék egymást. Mivel távvezérlésről van szó, a felhasználó nem kap közvetlen környezetéből jelzést a jármű helyzetéről és állapotáról. Ebből az következik, hogy a járműre különféle szenzorokat kell elhelyezni, melyek segítik az irányítót tájékozódni. Legalapvetőbb követelmény az, hogy legyen egy kamera, melynek jelét a számítógép monitorán jól lehet látni. Sokat segít az is, ha a felhasználó látja, hogy hol is van a jármű, merre néz, mekkora a pillanatnyi sebessége és az sem árt, ha tudja a felhasználó, mekkora távot bír még a járműve megtenni.

Ezen követelmények alapján és a rendelkezésemre álló eszközök figyelembe vételével úgy döntöttem, hogy az Android operációs-rendszert futtató telefonomat használom fel a jármű oldalán. A telefonban van Wi-Fi¹ adapter valamint HSPA² modem, így a kommunikáció lebonyolítható a helyi hálózaton és az interneten keresztül is. A telefon továbbá tartalmaz még gyorsulásmérőt³, magnetométert⁴, GPS⁵ szenzort és egy 5 megapixeles mikrokamerát a hátoldalán. A telefont a jármű elejéhez rögzítve élő kép sugározható. A gyorsulásmérő és a magnetométer adatait egybevetve megtudható, hogy a telefon hány fokkal tér el az északi mágneses sarktól, tehát megtudható, merre néz a telefon és ezáltal az autó. A GPS szenzor alapján méter pontossággal megtudható, hol tartózkodik a jármű és megállapítható az is, hogy körülbelül mekkora a pillanatnyi sebessége. Ismerve a térbeli elhelyezkedést és a pontos időt, meghatározható a valódi északi iránytól való eltérés is, ami alapján már térképen is megjeleníthető a jármű haladási iránya.

Elektronikai szempontból nézve a feladatra meg kellett oldanom, hogy a telefon képes legyen olyan jelek leadására, melyekkel a jármű precízen irányítható. (Precíz irányítás alatt azt értem, hogy a jármű sebessége és kanyarodásának iránya tetszés szerint állítható.) A telefont ismervén kétféle átviteli közeg jöhetett szóba a jelek leadására: USB⁶ kábel, valamint Bluetooth⁷. (Mivel a telefonban csak egy Wi-Fi adapter van, az nem jöhetett szóba, mivel már hálózati kommunikációra le lett foglalva az előző bekezdésben leírtak alapján.) Mindkét közegnek vannak előnyei és hátrányai is. Ha az összeköttetést USB kábellettel valósítom meg a telefon és a jármű között, akkor stabil, gyors és biztonságos kapcsolat jön létre, amit nem zavar be más jeladó; ha viszont Bluetooth alapú kapcsolatot alakítok ki, mobilról is irányítani lehet a járművet. A vezeték nélküli kapcsolatot sajnos könnyen működésképtelenné lehet tenni. Egyszerűen ugyan azon a frekvencián kell sugározni, amin a telefon, így a jel annyira legyengíthető, hogy a kommunikáció megszűnik a telefon és a jármű között. Éppen ezért úgy döntöttem, hogy a járművet USB kábellettel kötöm össze a telefonnal. Mivel rendelkezem laptopjal és otthoni Wi-Fi hálózattal, nem jelentett számomra hiányt a telefonról való vezérlés lehetőségének megszűnése.

1 A Wi-Fi semmilyen angol kifejezésnek nem rövidítése, csupán szójáték a Hi-Fi szóra.

Az IEEE 802.11 - vezeték nélküli mikrohullámú kommunikációt megvalósító szabvány - népszerű neve.

2 High Speed (Download/Upload) Packet Access, azaz nagy sebességű le- és feltöltésű csomagelérés.

Mobilinternet-szabvány, mely elméletileg képes akár 28 Mb/s adatátviteli sebesség elérésére is.

3 A gyorsulásmérő a rá ható nehézségi erőket méri külön-külön a három tengellyel párhuzamosan.

4 A magnetométer a mágneses térrősségi mérésére alkalmas műszer.

5 Global Positioning System, azaz Globális Helymeghatározó Rendszer.

A Föld bármely pontján, a nap 24 órájában működő műholdas helymeghatározó rendszer.

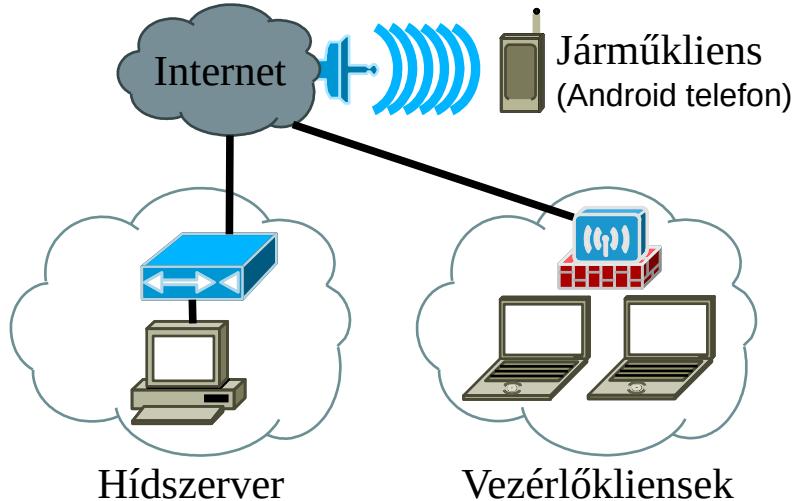
6 Universal Serial Bus, azaz univerzális soros busz. Egy nagyon elterjedt számítógépes csatlakozó.

Az USB 2.0 névleges adatátviteli sebessége 480 Mb/s.

7 A Bluetooth rövid hatótávolságú, adatcseréhez használt, nyílt, vezetéknélküli szabvány.

A Bluetooth 2.0-ás verziója 3 Mb/s adatátviteli sebességet tesz lehetővé a 2,4 GHz-es frekvenciasávban.

Hálózati szemszögből tekintve a feladatra felmértem, milyen nehézségekkel kell szembenéznem. Ehhez elképzeltem egy használati esetet, ami tartalmazza az összes problémát, de még viszonylag egyszerű hálózati struktúrát alkot. Arra a következtetésre jutottam, hogy három alkalmazást kell írnom ahhoz, hogy az ötletem kivitelezhető legyen. Az alábbi ábra segít ennek a megértésében.



1. ábra: Hálózati példa a vezérlő számítógép és a mobiltelefon kapcsolatára.

Az ábrán három elkülönült hálózat látható. A telefon mobilinternetet használ, aminek eredményeképpen egy olyan elkülönített hálózat részévé válik, melynek egyetlen tagja van: önmaga. Természetesen ez a hálózat valójában csak logikai szinten létezik, amit a mobiltorony hoz létre úgynevezett NAT⁸-olás segítségével. Egy azon toronyhoz több mobileszköz is kapcsolódhat és mindenki között megkaphatja ugyan azt az internetes IP-címet (WAN⁹ IP). (A mobilszolgáltatók így oldották meg, hogy több ügyfelük lehessen egyszerre az Interneten, mint amennyi IP címet ki tudnak osztani. Az IPv4¹⁰ címek mind ki lettek már osztva a szolgáltatóknak, így abból gázdálkodhatnak, amire sikerült szert tenniük. Megoldást jelent az IPv6¹¹ címekre való átállás, de ez költséges és időigényes folyamat, ezért egyhamar nem fog megtörténni széles körben.) Az én nézőpontomból ez csupán annyit jelent, hogy a telefon nem képes szerverként funkcionálni, mivel egy aktív porttal¹² se rendelkezik, csak a vezérlőkliensként használható. (Ez logikus is, mivel a WAN IP mögötti eszközök nem címezhetők egyértelműen, csak akkor, ha már van kiépített kapcsolat, tehát mindenki irányban le van foglalva egy-egy port a kommunikációs csatornának.) A NAT-olás megkerülhető lenne VPN¹³ használatával, de több problémát hozna fel, mint amennyit megoldana: nem értenek hozzá a felhasználók és egy központi szerver is kéne hozzá, így a VPN nem jöhetsz szóba. A vezérlő programnak kéne ezek szerint szerverként üzemelni?

8 Network Address Translation, azaz hálózati címfordítás.

Lehetővé teszi a belső hálózatra kötött gépek közvetlen kommunikációját a külső gépekkel anélkül, hogy azoknak saját nyilvános IP-címmel kellene rendelkezniük.

9 Wide Area Network, azaz nagy kiterjedésű hálózat.

Egy olyan számítógép-hálózat, mely nagyobb területet fed le. Lényegében az Internet a WAN hálózatok összessége.

10 Internet Protocol version 4, vagyis az internetprotokoll 4-es verziója.

Tartománya 2^{32} azaz 4 294 967 296 darab egyedi IP címet tesz ki.

11 Internet Protocol version 6, vagyis az internetprotokoll 6-os verziója, melyet az IPv4 leváltására terveztek.

Az IPv6 címek 128 bitesek, tehát tartománya 2^{128} darab (39 számjegy) egyedi IP címet tesz ki.

12 A hálózati portok egy hálózati kommunikációs csatorna végpontjai.

A portok használata teszi lehetővé, hogy a beérkező csomagokból a számítógépen futó programok csak a nekik szóló csomagokat kapják meg. Tehát a port egy - konkrét gépen futó, - konkrét programot címz meg.

Aktív port alatt olyan hálózati portot értünk, melyet használva egy alkalmazás (a tűzfalakat figyelmen kívül hagyva) az Internet bármely pontjáról elérhetővé válik.

13 Virtual Private Network, azaz virtuális magánhálózat.

Egy számítógéphálózat fölött kiépített másik hálózat. A VPN-en keresztülmenő adatok nem láthatók az eredeti hálózaton, mivel titkosított adatcsomagokba vannak becsomagolva. Ezért hívják magánhálózatnak.

Az ábrán a jobb oldali felhőben lévő eszközök is egy elkülönített hálózat részei. Ez a hálózat tekinthető egy étterem nyilvános hálózatának, de akár egy otthoni zárt hálózat is lehet. Az egyszerűség kedvéért ebben a hálózatban csak két számítógép található és minden csatlakozó csak a vezérlő program fut. Arra a kérdésre, hogy melyik program legyen a szerver ez az elkülönített hálózat adja meg a választ. A mobilinternet esetén tapasztalt akadályok itt is előjönnek: a vezérlő számítógép is NAT-olva van (pl. router által), mivel nem közvetlenül kapcsolódik az Internetre, hanem egy belső hálózat része így nem ő birtokolja a WAN IP-címet. Ebből az következik, hogy az ebben a hálózatban lévő gépeknek sincs aktív portjuk, amit el lehetne érni a telefon mobilinternetéről. Természetesen port átirányítással bármely gépnek kiosztható egy vagy több port, de ezt csak a rendszergazda teheti meg és a felhasználók természetesen ehhez sem értenek; a felhasználók otthoni hálózat esetén se fognak ezzel bajlódni. Ha be van állítva tűzfal, tovább nehezítheti a dolgukat. Ebből már lehet sejteni, hogy nem jó ötlet a vezérlő programot szerverként használni, de a legfőbb érv ellene az, hogy több vezérlő számítógép is van a hálózatban, melyekről más-más felhasználók szeretnék irányítani a járművet. Tovább megyek: több jármű van, amit lehet irányítani, vagyis sok-sok kapcsolat áll fent a vezérlők és a járművek között. Ha a vezérlő program lenne a szerver, nem lehetne több gépről nyomon követni a járművet és a sok-sok kapcsolatot se lehetne megvalósítani. A relációs adatbázisoknál használt trükk a sok-sok kapcsolat létrehozására itt is alkalmazható. A megoldást egy közbülső szerver alkalmazás jelenti, ami összeköti a vezérlőket a járművekkel. Tehát a válasz arra a kérdésre, hogy a vezérlő program legyen-e a szerver: Nem, hanem egy harmadik program.

Természetéből adódva ezt a szerver alkalmazást elneveztem Hídnak. Ezt az alkalmazást egy olyan gépen kell üzemeltetni, ami aktív porttal rendelkezik, tehát a WAN IP és a port megadásával bárki elérheti azt az Internetről és vagy fix IP-címe van, vagy domain név alapján címzhető, tehát olyan címmel rendelkezik, ami fix és minden ő rá mutat. A többi feltétel (pl. internetes sávszélesség, processzor órajel) jelen nézőpontból nem érdekes. A hídszert bárki üzemeltetheti, aki ezen feltételeknek eleget tesz. Innentől kezdve a kliensalkalmazásokat használó felhasználók hálózati ismeretek nélkül is könnyen el tudják érni a szervert, csak a pontos címre van szükségük. Az ábrán a bal oldali felhőben látható a legegyszerűbb eset, ami kielégíti a fenti feltételeket; a szerver közvetlenül kapcsolódik az internetre egy közösséges kábelmodem segítségével.

Összefoglalva a három alkalmazás elnevezése és feladata:

- Járműkliens: a mobiltelefonon fut és az USB-kábelen keresztül vezérli a jármű áramkörét.
- Vezérlőkliens: asztali számítógépen fut, leadja a vezérlőjelet és kijelzi a jármű adatait.
- Hídszerver: összeköti a járműklienseket a vezérlőkliensekkel és jogosultságot kezel.

2. Felhasználói útmutató

Mielőtt belekezdenék az alkalmazások működési elvének részletezésébe, bemutatom, hogyan lehet őket beszerezni és leírást adok a használatukhoz. Az alkalmazások funkcióinak ismeretében sokkal egyszerűbb megérteni a mögöttük álló működési elvet, amit a későbbi fejezetekben még részletezni fogok.

2.1. Rendszerkövetelmények

Az asztali számítógépekre kiadott hídszerver és vezérlőkliens Windows, GNU/Linux és Mac OS X rendszereken lettek tesztelve és minden rendszeren teljes funkcionalitással használhatóak, de legalább Java 1.6-os virtuális gép szükséges az elindításukhoz. A telefonon használandó járműkliens az Android 2.1-es verziójától telepíthető.

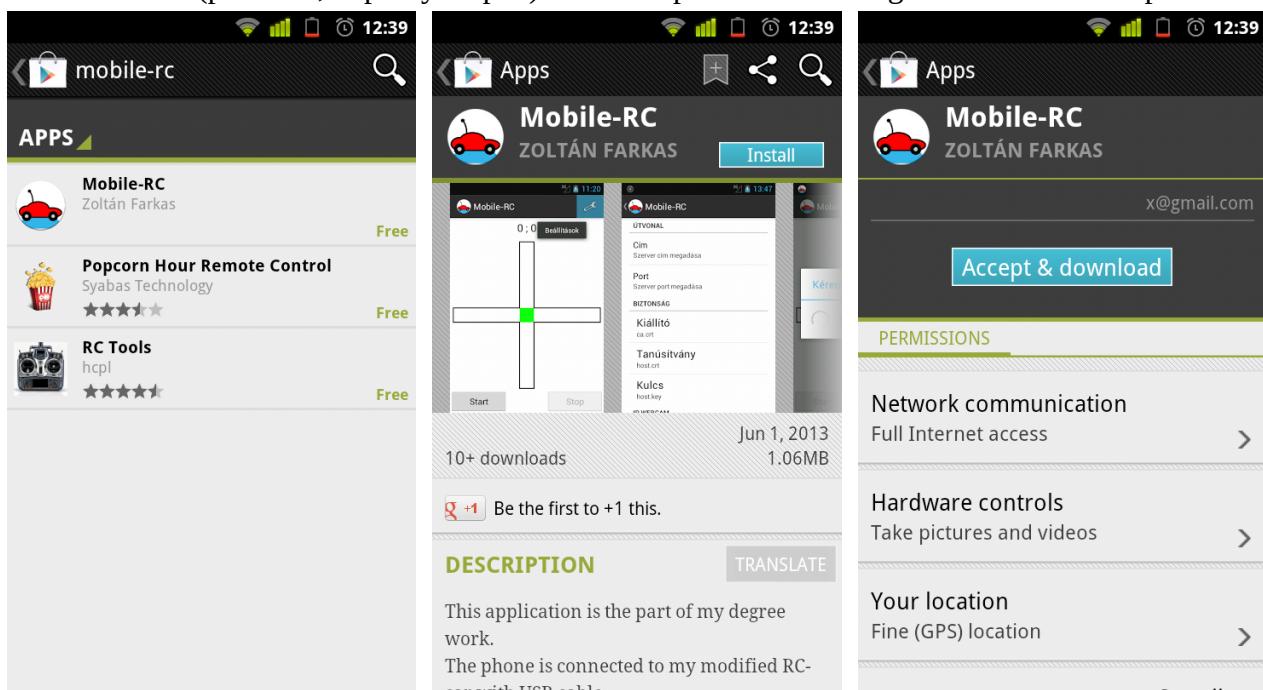
2.2. Beszerzés

A telefonra írt járműkliens telepíthető a Google Play-en keresztül, és számítógépen is megtekinthető a következő címen: <https://play.google.com/store/apps/details?id=org.dyndns.fzoli.rccar.host>

A Google Play-ben a „Mobile-RC” kifejezésre rákeresve jelenik meg a találatok között a program. A hídszerver és a vezérlőkliens alkalmazásokhoz telepítő készült minden operációs-rendszerre, melyek megtalálhatóak a diplomamunkámhoz mellékelt lemezen, de az Internetről is letölthető a legfrissebb verzió a [github.com oldalról: https://github.com/fzoli/RemoteControlCar](https://github.com/fzoli/RemoteControlCar)

2.3. Telepítés Android rendszerre

A Google Play-ben az alkalmazás megtalálása és kiválasztása után megjelennek az alkalmazás részletes adatai (pl. leírás, képernyőképek). Ezen a lapon van lehetőség az alkalmazás telepítésére.

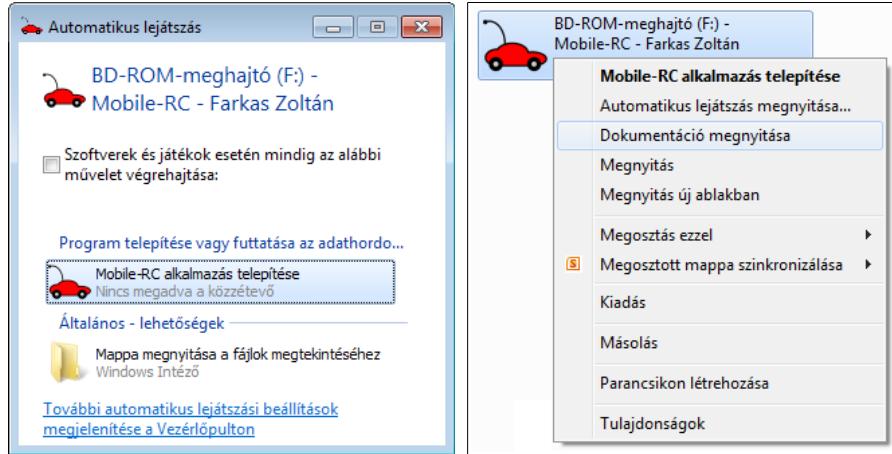


2. ábra: Google Play

A telepítést választva megjelennek azok a jogosultságok, melyeket az alkalmazás használ. Ezeket tudomásul véve az elfogadás után letöltődik és települ a járműkliens.

2.4. Telepítés Windows rendszerre

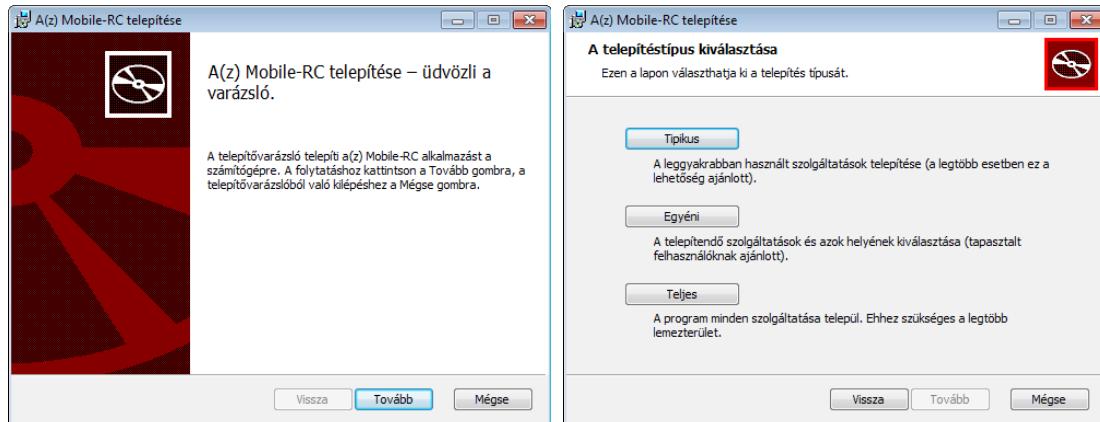
A mellékelt lemezt behelyezve a Windows automatikus lejátszás ablaka jelenik meg, ami fel is kínálja az alkalmazás telepítését. Az optikai meghajtó menüt előhívva a diplomamunkám PDF változatát is könnyen meg lehet tekinteni a Dokumentáció megnyitása opció választásával.



3. ábra: Automatikus lejátszás

Az automatikus lejátszás a telepítést választva a Setup.msi telepítőt indítja el. Windows XP SP2 vagy régebbi rendszereken nem biztos, hogy telepítve van a Windows Installer, ami szükséges a telepítő elindításához, ezért készítettem egy hordozható változatot is, melyről később még szó lesz.

Miután a telepítő sikeresen betöltődött, megjelenik az üdvözlő képernyő, ami után három telepítési mód közül lehet választani.

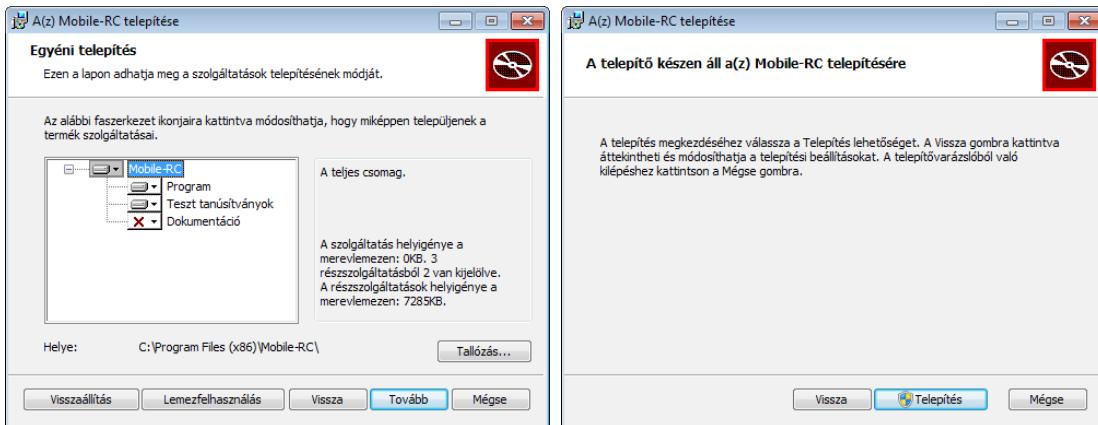


4. ábra: Windows telepítő - kezdőképernyő

A három telepítési mód a következő:

- Tipikus* telepítés esetén az alkalmazás teszttanúsítványokkal együtt települ, így könnyen kipróbálható az alkalmazás az előre beállított konfigurációval.
- Teljes* telepítés esetén az összes komponens – az alkalmazás, a teszttanúsítványok és a diplomamunkám PDF verziója – feltelepül.
- Egyéni* telepítés esetén ki lehet választani, hogy szükség van-e a teszttanúsítványokra vagy a PDF dokumentációra és a telepítés helye is módosítható. Szükség esetén megoldható, hogy csak az alkalmazás települjön, de a tanúsítványokat ekkor kézzel kell generálni, kivéve ha már van saját tanúsítvanya a felhasználónak éles rendszerre.

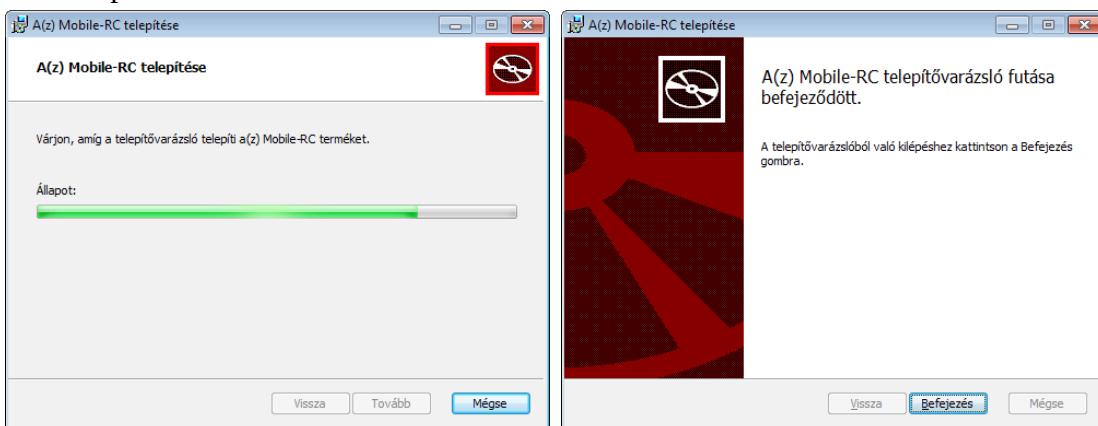
A telepítéstípus kiválasztása után a telepítő jelzi, hogy a telepítés készen áll, a Telepítés gombra kattintva megkezdődik a telepítés.



5. ábra: *Windows telepítő - testreszabás*

Windows Vista vagy újabb rendszereken a telepítő nem az indulása előtt, hanem a Telepítés gombra kattintás után kér csak rendszergazda jogot, ezért ha az UAC¹ be van kapcsolva, engedélyt kell adni a telepítőnek, hogy megkezdhesse a telepítést.

A telepítés engedélyezése után megkezdődik a telepítés és a befejeződése után a telepítő jelzi, hogy sikerült-e a telepítés.



6. ábra: *Windows telepítő - befejezés*

Sikeres telepítés után az asztalon és a Start menüben is megtalálható a kliens és a szerver alkalmazás. A Start menüben továbbá megtalálható az alkalmazás eltávolító és amennyiben telepítve lett, a dokumentáció is. Az alkalmazás eltávolítható a Vezérlőpulton keresztül is.

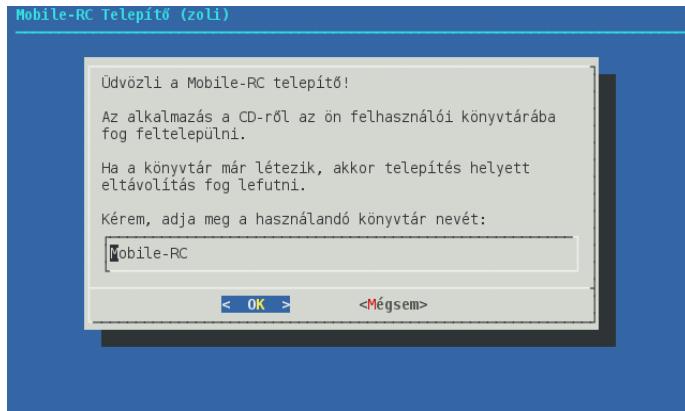
2.5. Telepítés GNU/Linux rendszerre

Linuxot főként szervereken használnak (pl. Debian), de léteznek otthoni használatra is készített disztribúciók, mint például az Ubuntu (aminek a Debian az alapja). Hogy minden felhasználók, minden rendszergazdák könnyelmesen használatba vehessék az alkalmazásokat, Linuxra konzol alapú, de párbeszéddobozos telepítőt gyártottam. Így a felhasználók is egyszerűen telepíthetik az alkalmazást, és azon szerver gépekre is telepíthető a program, melyeken nincs grafikus felület.

1 User Account Control, azaz felhasználói fiókok felügyelete.

Windows Vista óta elérhető funkció, mely az olyan kártevő programok ellen lett létrehozva, melyek rendszergazda jogosultsággal próbálnak kárt okozni az operációs-rendszerben. Az UAC segítségével a kártevő nem képes a háttérben lefutni, mivel a felhasználónak jogot kell neki biztosítani ahhoz, hogy rendszergazda jogosultsághoz jusson annak ellenére, hogy a felhasználó rendszergazda joggal bír.

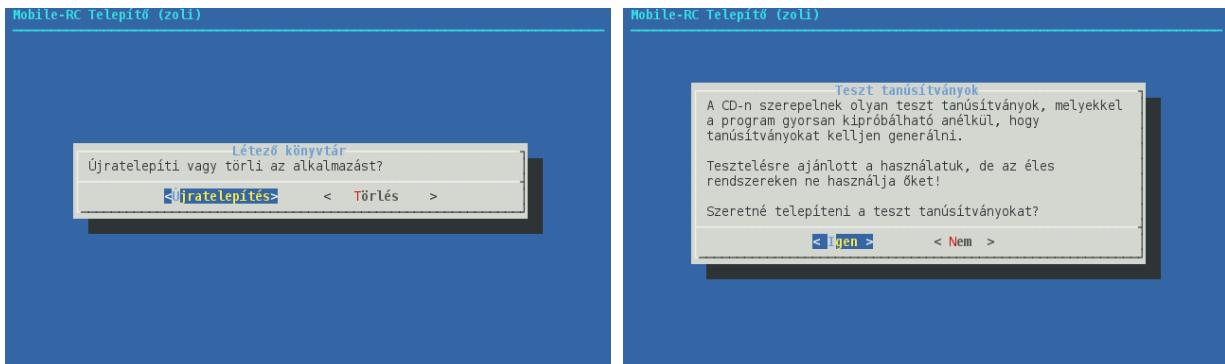
A telepítő elindítható a mellékelt lemesről a linux-installer.sh fájlt futtatva, vagy az Internetről letöltött linux-installer.bsx fájlt futtatva. Az első képernyőn megadható a könyvtár neve, amelybe települnek az alkalmazások és a teszttanúsítványok. A könyvtár a felhasználó home könyvtárába települ, de relatív útvonal beírása esetén mélyebb könyvtár is kiválasztható.



7. ábra: Linux telepítő - kezdőképernyő

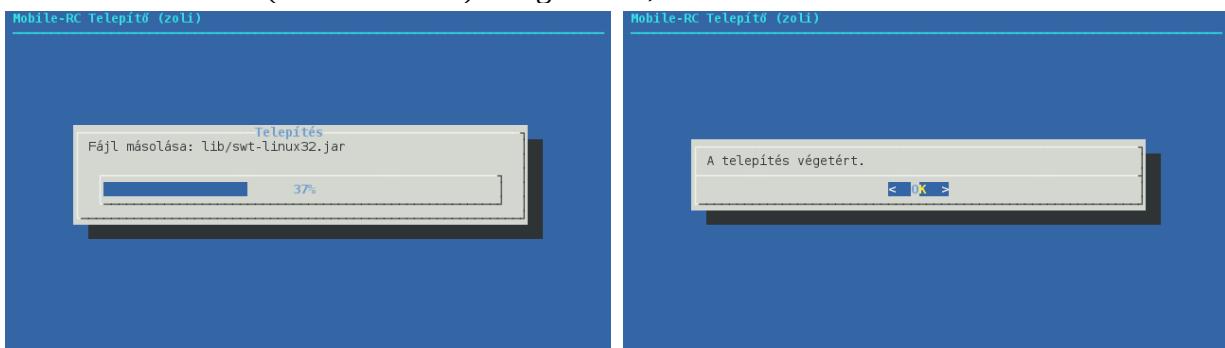
Ha a telepítő véletlen lett indítva, a Mégsem opcióval ki lehet lépni a telepítőből, a könyvtár megadása után az OK opcióval lehet a telepítőt folytatni. A telepítő folytatása után kilépésre nem lesz lehetőség, csak a varázsló befejeződése után.

A telepítési útvonal megadása után a telepítő ellenőrzi, hogy létezik-e az adott könyvtár. Ha létezik, megkérdei, hogy újratelepítés vagy törlés fusson-e le. A telepítő ez után megkérdei, hogy telepítse-e a teszttanúsítványokat.



8. ábra: Linux telepítő - válaszadás

A válaszadás illetve válaszadások után a telepítő átmásolja a fájlokat és létrehozza az indítóikonokat a menüben az Internet (illetve Hálózat) kategóriában, valamint az asztalra is készít hivatkozásokat.

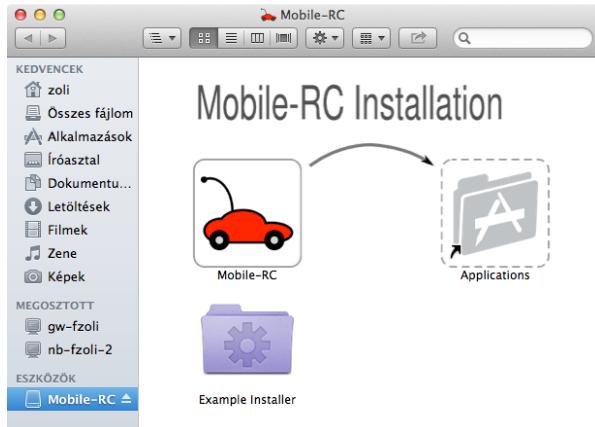


9. ábra: Linux telepítő - telepítés

A telepítés befejezése után az alkalmazások grafikus felület esetén legegyszerűbben a létrehozott indítóikonok segítségével indíthatóak el. GUI nélkül csak a Hídszerver futtatható, a telepítés megadott könyvtáron belül a server.sh fájlt indítva.

2.6. Telepítés Mac OS X rendszerre

Az Apple a telepítések egyszerűsítése érdekében létrehozott egy olyan képfájlt, mely jól tömöríti a benne lévő adatokat, így Internetről is gyorsabban letölthetők a DMG képfájlban lévő telepítők. OS X alatt a telepítés két-három egérmozdulatból áll. A mellékelt lemezen a mac-installer.dmg fájra duplán kattintva a rendszer felcsatolja (mount) a képfájlt és megjeleníti a tartalmát.



10. ábra: OS X telepítő

A megjelenő ablak mutatja, hogy a Mobile-RC nevű alkalmazás Drag & Drop módszerrel telepíthető az Applications (Alkalmazások) könyvtárba való egyszerű másolással. Ha a teszttanúsítványokra is szükség van, az Example Installer elindításával a háttérben felsolásnak a tanúsítványok és a konfigurációs fájlok. Miután befejeződött a háttérfolyamat, a rendszer jelzi, hogy sikerült a másolás, vagy hiba lépett fel a másolás közben.

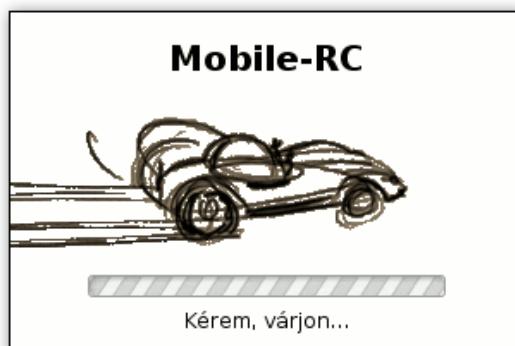
OS X alá is létezik, a Windows alatt már megszokott, szabványosított telepítő-varázsló, de mivel az alkalmazásom egyetlen könyvtárból áll és semmi egyedi háttérfolyamat futtatására nincs szükség, nem hoztam létre külön telepítőt. Több OS X alá letölthető alkalmazás is ezen módszert használja.

A telepítés után a képfájl lecsatolható és törölhető, a Mobile-RC, ahogyan a többi alkalmazás is, az Alkalmazások könyvtárból indítható és a letörléséhez is csak az indítóikon törlésére van szükség.

2.7. A vezérlőkliens használata

2.7.1. Az alkalmazás indulása

Az alkalmazás indulását követően egy töltőképernyő jelenik meg, ami az első látható ablak megjelenéséig látható marad és jelzi, hogy a program már el lett indítva, de még nem töltődött be teljesen. Erre azért van szükség, mert 10-20 másodpercet is igénybe vehet az alkalmazás elindítása. Ez az ára annak, hogy a felületi komponenseket az alkalmazás indulásakor előre legyártom, de cserébe a betöltés után a felület a felhasználó kéréseire gyorsan fog reagálni.

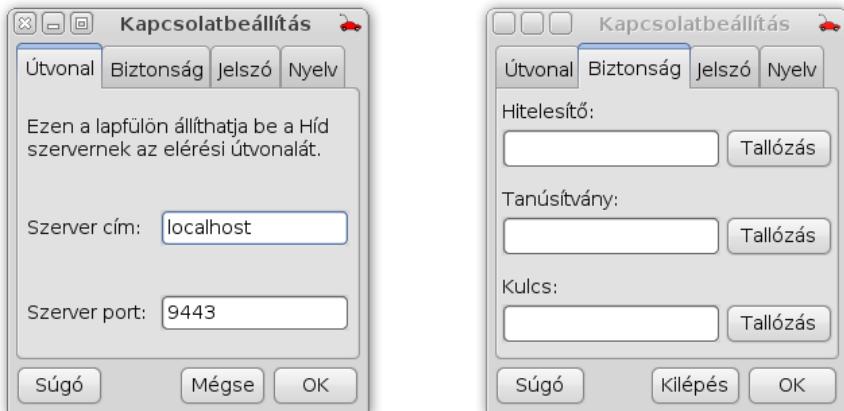


11. ábra: Töltőképernyő

A konfiguráció betöltődését követően az alkalmazás megpróbál kapcsolódni a Hídszerverhez, feltéve ha már be lett állítva a szerver elérhetősége és meg lett adva a titkosított kapcsolathoz használt tanúsítványfájlok helye.

2.7.2. Az alkalmazás beállítása

Ha szükség van a konfiguráció beállítására – pl. valaki áthelyezte a tanúsítványfájlokat, első indítás –, akkor megjelenik a kapcsolatbeállító-ablak.



12. ábra: Kapcsolatbeállító-ablak 1.

A bal oldali képen látható ablak a felhasználó kérésére jött fel az adatok utólagos módosítására, a jobb oldali képen látható ablakot az alkalmazás jelenítette meg az első induláskor. A két eset között annyi az eltérés, hogy az első induláskor kötelező az adatokat megadni, ezért vagy helyesen beállítja a felhasználó a kért adatokat, vagy kilép a programból. Az utólagos módosítás esetén a program nem áll le, csak bezárul az ablak.

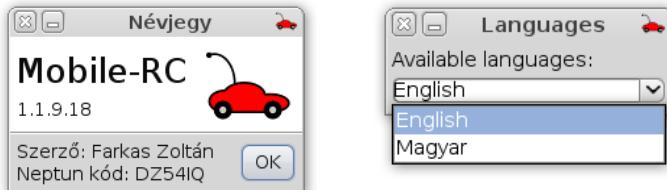
Az adatok megadása után az OK gombra kattintva lehet elmenteni az adatokat, viszont ha nem lett minden adat megadva, figyelmeztető üzenet jelenik meg és az ablak látható marad. A Mégse/Kilépés gombra kattintva a konfiguráció nem változik meg. Ha a felhasználó az ablakot bezárását az operációs-rendszerrel kéri, módosulatlan konfiguráció esetén bezárul az ablak egyébként egy Igen/Nem/Mégse dialógus jelenik meg megerősítést kérve a konfiguráció mentésére és a bezárásra.

A kapcsolatbeállító-ablakban további két opcionális beállítási lehetőség is elérhető. Ha a használt tanúsítvány jelszót igényel és a konfigurációban a jelszó el van mentve, a jelszó bármikor törölhető. Az utolsó lapfölön az alkalmazás nyelve állítható át. A nyelvet átállítva azonnal módosul a nyelv, nincs szükség újraindításra. A Mégse gombbal kilépve a nyelv visszaáll a beállítás előttire.



13. ábra: Kapcsolatbeállító-ablak 2.

Hogy a nyelv azok számára is könnyen átállítható legyen, akik nem értik az aktuális nyelvet, az értesítési területen elhelyeztem egy ikont, mely menüjéből előhozható egy angol nyelvű nyelvkiválasztó, de a kapcsolatbeállító-ablak és az alkalmazás névjegye is elérhető innen.

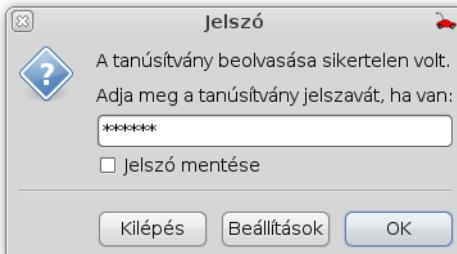


14. ábra: Névjegy és nyelvek

A konfiguráció beállítása után megkezdődik a kapcsolódás a szerverhez, illetve ha már van kialakítva kapcsolat és át lett írva a szerver címe, megkérdezi az alkalmazás, hogy kapcsolódjon-e újra, de ha itt véletlen a Nem lett kiválasztva, az újrakapcsolódás az értesítési ikon menüjéből bármikor kérhető.

2.7.3. Kapcsolódás a Hídszerverhez, jelszóbekérés

Mivel a tanúsítvány jelszavát nem kötelező és nem is ajánlott tárolni, a jelszót a kliens a kapcsolódás előtt kéri be. Ezen dialógusablakon kérhető a jelszó tárolása is a későbbi indítás gyorsítása érdekében, de felhívom a figyelmet arra, hogy a jelszó titkosítás nélkül tárolódik.

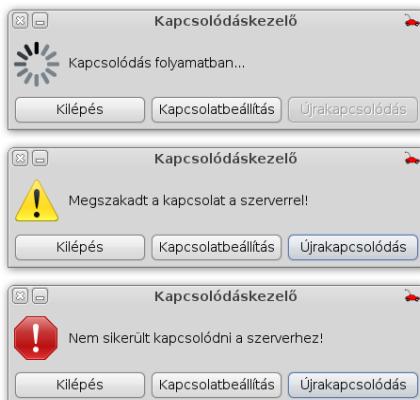


15. ábra: Tanúsítványjelszó bekérése

A jelszókérő ablak csak akkor jön elő, hogy ha jelszó nélkül nem sikerül megnyitni a titkos kulcsot (a key kiterjesztésű fájlt). Hibás jelszó megadása esetén újra megjelenik a jelszókérő ablak mindenkorábban, míg meg nem adják a helyes jelszót. Ha a felhasználó nem tudja a jelszót, kiléphet a programból, vagy átállíthatja a konfigurációt más tanúsítványra. Ha a tanúsítványhoz nem tartozik jelszó, valószínűleg megsérültetett a crt vagy a key fájl.

2.7.4. Kapcsolódáskezelő-ablak

A jelszó megadása után folytatódik a szerverhez való kapcsolódás. Mivel a töltőképernyő már eltűnt, a kapcsolódás tényét a kapcsolódáskezelő-ablakban jelzi az alkalmazás. Ez az ablak jelenik meg akkor is, ha valami gond van a kapcsolattal és az újrakapcsolódás is innen indítható el a leggyorsabban.



16. ábra: Kapcsolódáskezelő-ablak

A kapcsolódáskezelő-ablak több hibaüzenetet is képes megjeleníteni:

- *Nem sikerült kapcsolódni a szerverhez!*

Ezen hibaüzenet esetén a szerver nem érhető el. Ennek több oka is lehet. Lehetséges, hogy nem fut a szerver, vagy rossz cím lett megadva. Ha a szerver nem LAN-on belül van, lehetséges, hogy nincs Internetkapcsolat.

- *A szerver címe nem érhető el!*

Ez a hiba csak akkor jön elő, ha domén-név van megadva IP-cím helyett és a névfeloldás nem sikerült. Lehetséges, hogy a cím el lett írva vagy nincs Internetkapcsolat.

- *Időtúllépés a kapcsolódás közben!*

Ha a hálózat vagy az eszköz túlterhelt, előfordulhat hogy nem sikerül a kapcsolat kialakítása a tíz másodperces időkorlátban belül. Az újrakapcsolódást választva jó eséllyel sikerül a kapcsolat létrehozása.

- *Az SSL² kapcsolat létrehozása nem sikerült!*

Ez a hiba akkor jelentkezik, ha sikerült kommunikálni a szerverrel, de a tanúsítvány-kiállító (legtöbbször ca.crt fájl a neve) beállítása eltér a szervernél és a kliensnél vagy el lett írva a cím és teljesen más szerver van címezve. Megoldás a problémára a cím ellenőrzése és ha biztosan jó, egyazon kiállító által generált kliens- és szerver-tanúsítvány használata.

Ritkább hibalehetőség az, hogy a kliens olyan tanúsítványt használ, melyet szerverekhez állítottak ki. Ha valóban ez a hiba, a kapcsolódás pillanatában a szerver egyértelműen figyelmeztet az értesítési területen, de a szerver naplózásában utólag is látszik. Ez esetben le kell cserélni a tanúsítványt.

- *A szerver elutasította a kérést!*

Ezen figyelmeztetés esetén a kapcsolat létrehozása sikerült a szerverrel, de a kliens által használt tanúsítvány tiltólistára került a szerveren vagy már kapcsolódtak a szerverhez az adott tanúsítvánnyal. Megoldás másik tanúsítvány használata ha van, vagy a tiltás feloldása a szerver oldalán.

- *Megszakadt a kapcsolat a szerverrel!*

Ha a szerver és a kliens közti kommunikációban tíz másodpercnél hosszabb kimeradás jön létre, akkor minden oldal eldobja a kialakított kapcsolatokat és megszakad a kapcsolat a két fél között. Kisebb kimeradások esetén a kapcsolat nem zárul be. Nagy valószínűséggel a hálózat vagy az eszköz túlterhelt. Mobilinternet használata esetén előfordulhat, hogy gyenge a jel vagy a forgalomkorlát túl lett lépve és korlátozva van a sebesség.

- *A tanúsítványok beállítása nem megfelelő!*

A hiba egyértelműen a kliens konfigurációjában keresendő. Akkor fordul elő, amikor a három tanúsítvány-fájl ugyan be lett állítva és léteznek is, kiterjesztésük is megegyezik és valóban tanúsítványok, de nem egy tanúsítványt alkotnak a crt és a key fájlok, tehát eltérő tanúsítvány lett megadva a nyilvános és a titkos kulcsnak.

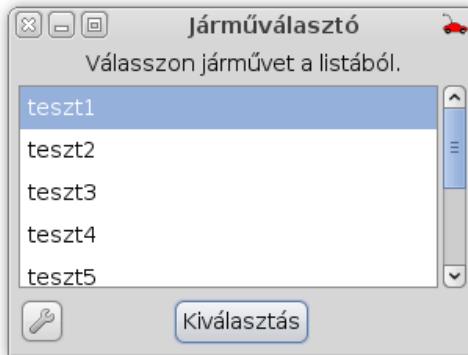
- *A szerver kliensekhez való tanúsítványt használ!*

A hiba egyértelműen a szerver oldalán keresendő. Akkor fordul elő, amikor a szervernek olyan tanúsítványt állítanak be, ami kliens oldalra lett kiállítva. Ezen hiba esetén a szerverhez egyetlen kliens se képes kapcsoldni.

2 Secure Socket Layer, azaz biztonsági alréteg. Az SSL egy protokoll réteg, amely a hálózati és az alkalmazási rétegek között van és mindenféle forgalom titkosítására használható.

2.7.5. A jármű kiválasztása

Sikeres kapcsolódás után ha az adott felhasználó még nem választott járművet, megjelenik a járműválasztó-ablak, amin az összes felhasználó által választható jármű látható; a felhasználó azok közül tud választani.



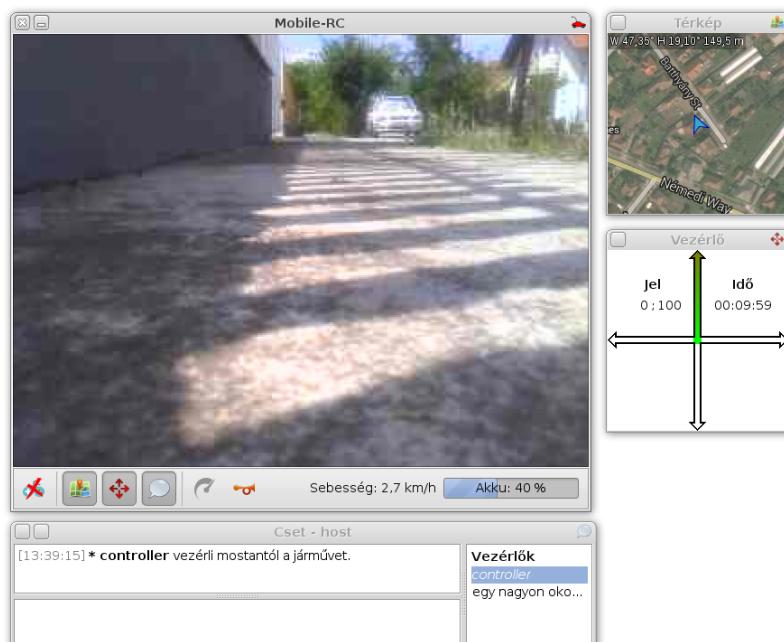
17. ábra: Járműválasztó-ablak

A kapcsolatbeállító-ablak innen is elérhető, mivel nem minden rendszeren van értesítési terület, ahonnan az alkalmazásikon menüjén keresztül meghívhatná a felhasználó az ablakot.

A lista frissül, amint kapcsolódik illetve lekapcsolódik egy jármű, de akkor is, ha a felhasználó jogot kapott az egyik járműhöz vagy éppen elvették azt. A listában nem szerepelnek offline járművek, így a kiválasztás gomb mindenkor inaktív marad, míg egyetlen jármű sincs kapcsolódva a szerverhez.

2.7.6. Járművezérlés, a főablak bemutatása

A jármű kiválasztása után előjön az alkalmazás főablaka, amihez társul még három kisebb dialógusablak is. A főablakon látható a 640x480 felbontású élő kép a telefon kamerájából, valamint itt kérhető/adható vissza a vezérlés és a dialógusablakok is itt tüntethetők el és jeleníthetők meg. Ezen kívül található még egy duda gomb is, amit csak az aktuális járművezérlő használhat és ha megnyomja, a telefon dudahangot játszik le. Ha a jármű precízen vezérelhető, akkor elérhető egy vezérlésmódosító gomb is, ami ha aktív, a gázadagolás folyamatosan történik.



18. ábra: Járművezérlő-főablak és dialógusablakai

A főablak jobb alsó sarkában található információ az autó energiaszintjéről és pillanatnyi sebességről. A sebessé csak akkor látszik, ha a telefonnak megfelelő GPS jele van ahoz, hogy meg tudja becsülni azt. Az akkumulátor-szint csak akkor látszik, ha a járműnek nincs vezérlőparancs kiadva. Erre azért van szükség, mert a kijelzett százalékos érték mindenkoránig mért feszültségszint alapján van kalkulálva és csak akkor mutat megbízható adatot, ha az áramkör alapállapotban van, tehát nem jár a motor.

A kamerakép közepén az alábbi tájékoztató-üzenetek jelenhetnek meg:

- **A jármű offline!**

Az egyetlen figyelmeztetőüzenet, ami csak akkor jelenik meg, amikor a jármű nincs kapcsolódva a hídszerverhez. A lekapcsolódás lehet szándékos, de az is lehet, hogy megszakadt a kapcsolat a szerverrel.

- **Várakozás az összeköttetésre.**

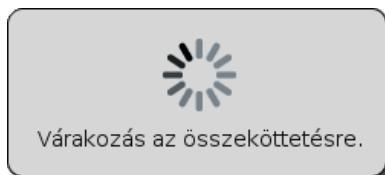
Akkor látható, ha a járműkliens kapcsolódva van a hídszerverhez, de nincs összeköttetésben – az USB kábelen keresztül – magával a járművel.

- **Várakozás a jármű kapcsolatára.**

Akkor látható, amikor a járműkliens kapcsolata instabil és több mint 1 másodperce nem vette fel a kapcsolatot a hídszerverrel. Ha a kapcsolat helyreáll, az üzenet eltűnik, de ha több mint 10 másodpercen keresztül nem válaszol a kliens, a szerver bontja a kapcsolatot és a jármű offline lesz, de mivel a járműkliens megszakadt kapcsolat esetén újra megkíséri a kapcsolódást, a hálózat helyreállása után a jármű visszakapcsolódik a szerverre.

- **Várakozás a Híd kapcsolatára.**

Ezen üzenet esetén is instabil a hálózati kapcsolat, de nem a jármű oldalán, hanem a vezérlőkliens oldalán. Itt is igaz, hogy a kapcsolat 10 másodperces időtúllépés után megszakad. Ebben az esetben megjelenik a kapcsolódáskezelő-ablak, ahonnan újra lehet kezdeményezni a kapcsolódást a szerverhez. Sikeres kapcsolatfelvétel után ha a szerver nem lett újraindítva, a főablak jelenik meg, tehát nem kell újra járművet választani.



19. ábra: Tájékoztató-üzenet

A járművet a főablak bezárásával lehet elhagyni. A jármű elhagyása után újra megjelenik a járműválasztó, mely bezárásával a kliensalkalmazás leáll. Ha a jármű offline, az alkalmazás megerősítést kér a jármű elhagyásakor, mivel a járműválasztóban már nem fog szerepelni a jármű, tehát a kilépés nem visszavonható művelet – legalábbis amíg a jármű nem érhető el.

2.7.7. Járművezérlés, a jármű irányítása

A vezérléskérő gombra kattintva (gyorsgomb: Shift + Enter) lehet kérni a jármű vezérlését. Két eset lehetséges: a kliens vagy várólistára kerül, vagy azonnal megkapja a vezérlést. A művelet mindenkoránig visszavonható, tehát le lehet iratkozni a várólistáról, valamint le lehet mondani a vezérlésről ugyan azt a gombot használva, amivel a vezérlés kérve lett (gyorsgomb: Shift + Backspace). A vezérlés átvétele után a nyilakkal vagy a W, A, S, D gombokkal lehet a járművet irányítani.

A vezérlésről való lemondást követően a várólistán elsőként szereplő kliens kapja meg a vezérlést. A várólista főként időrend alapján áll elő, tehát aki előbb kérte a vezérlést, előbb fogja megkapni, de a szerver precízen konfigurálható, hogy mely felhasználók milyen prioritással rendelkeznek adott járművenként. Így előfordulhat az is, hogy az egyik vezérlő elveheti a másiktól a vezérlést, ha annak nagyobb a prioritása. Erről bővebben a hídszerver beállításainál lehet olvasni.

2.7.8. Járművezérlés, a dialógusablakok bemutatása

A főablakhoz tartozik még három dialógusablak: térkép, vezérlő és cset. Ha a felbontás engedi, az összes dialógusablak látható, de ha túl kicsi a képernyő felbontása, az is lehet, hogy csak a főablak látható, mivel a dialógusok előhozásával a főablak takarásba kerülne.

A térkép csak akkor jelenik meg, ha a GPS be van kapcsolva a telefonon. Ha a pozíció ismeretlen, térkép helyett iránytű látható és amíg az irány sem ismert, az iránytű nem mutat irányt, kör alakot formál. Mivel a térképet a Google Map szolgáltatja, internetkapcsolat hiányában hibaüzenet jelenik meg, ahonnan elérhető az iránytű, valamint újra lehet próbálni a kapcsolódást a Google szerveréhez.



20. ábra: Térkép

A vezérlőpanel segítségével egérrel is irányítható a jármű, de fő célja az, hogy precízen vezérelhető legyen a jármű. Szűk helyeken előfordulhat, hogy nem elég pontos a billentyűzettel való irányítás. Ekkor az egérrel pontosan megadható, hogy mekkora sebességgel haladjon a jármű, valamint jobb egérgombbal sebességkorlát is beállítható, így a billentyűzetet használva is irányítható a jármű.

A csetablak segítségével a járműhöz kapcsolódott felhasználók beszélgethetnek egymással és a rendszerüzenetek is itt jelennek meg. Pl. felhasználó kapcsolódott a járműhöz, új járművezérlő, vezérléskérés és visszavonása. Az ablak jobb oldalán egy felhasználólista is látható. A listában dőlt betűvel szerepel a helyi felhasználó, valamint ha van járműirányító, ki van emelve a listában a neve.

2.8. A hídszerver használata

A hídszerver úgy lett megírva, hogy egyszerűen beállítható legyen és futtatható legyen azokon a rendszereken is, melyeken nincs grafikus felület, tehát kimondottan szervergépek. Ha a szerver tehát Linuxon a virtuális konzolból van indítva, minden információ a konzolra íródik ki.

2.8.1. Az alkalmazás indulása

Az alkalmazás indulásakor ugyan úgy megjelenik a töltőképernyő, ahogyan a vezérlőkliens esetén is. Amint a szerver elindult, a töltőképernyő eltűnik. Ha a szerver beállításait tartalmazó fájl nem létezik, az alkalmazás jelzi, hogy létrehozta az alapértelmezett beállításokkal és a program leáll. A következő induláskor ha a beállítások nem megfelelők, a program felsorolja a kijavítandó hibákat. Ha a konfiguráció megfelelő, a szerver elindul és ha szükséges, előtte bekéri a tanúsítvány jelszavát.

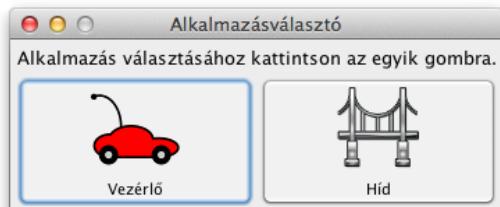


21. ábra: Szerver hibaüzenet

2.8.2. Az alkalmazás indítása Mac OS X alatt

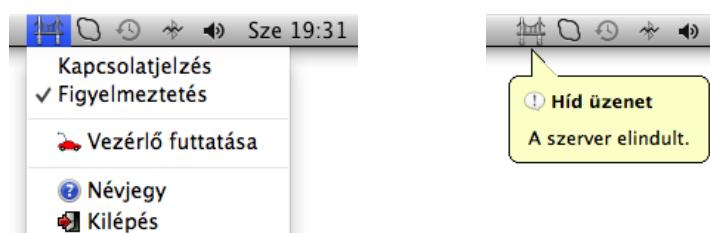
Figyelem: Mac OS X alatt az alkalmazás kizártlag az Apple virtuális gépével indul el!

OS X alatt az operációs-rendszer által biztosított felületről alkalmazásoknál csak egy folyamat (process) indítható. Mivel a Mobile-RC kliens és szerver alkalmazása ugyan azon csomag része, jár hozzá egy alkalmazás-választó is, hogy a felhasználó el tudja dönten, melyik alkalmazást akarja indítani.



22. ábra: Alkalmazásválasztó OS X alatt

Ha szükség van a szerver alkalmazásra is, az alkalmazás-választóban a Hídszervert kell indítani. A szerver indulása után a menüből indítható el a vezérlőkliens, így OS X alatt is egyszerűen indítható minden alkalmazás.



23. ábra: Értesítési terület OS X alatt

2.8.3. Az alkalmazás paraméteres indítása konzolból

Grafikus felület hiányában a kapcsolatjelzések és a figyelmeztetések futás közben nem állíthatóak, ezért a szerver három paramétert támogat, amik helyettesítik a grafikus menüt:

- v A figyelmeztetések engedélyezve vannak, de a kapcsolatjelzések nem.
- vv A figyelmeztetések és a kapcsolatjelzések is engedélyezve vannak.
- m A figyelmeztetések és a kapcsolatjelzések tiltva vannak.

A paramétereknek nincs hatása, ha a grafikus felület támogatva van, viszont ha nincs támogatva és több mint 1 paramétert ad meg a felhasználó, vagy a megadott paraméter nem egyezik a fenti három egyikével, az alkalmazás megjeleníti a konzolon a súgót majd leáll.

2.8.4. A Hídszerver konfigurációja

A Hídszerver konfigurációja az alkalmazás könyvtárában található, a fájl neve *bridge.conf*. Ebben a fájlban adható meg, hogy a szerver melyik porton figyeljen és hogy mely tanúsítvány-fájlokat használja az SSL kapcsolat létrehozásakor. A konfigurációban a privát kulcs jelszava is opcionálisan megadható, hogy a szerver beavatkozás nélkül is indítható legyen.

Példa a szerverrel kapcsolatos adatok megadására:

```
port 9443 # az a TCP port, amin a szerver figyel
ca test-certs-passwd/ca.crt # a tanúsítványokat kiállító CA tanúsítvány-fájl
cert test-certs-passwd/bridge.crt # a szerver tanúsítvány-fájl
key test-certs-passwd/bridge.key # a szerver titkos kulcsa
password asdfgh # a szerver tanúsítványának jelszava, ha van
```

A fájlból minden – nem üres – sor egy paraméter. A sor a paraméter nevével kezdődik, amit szóköz követ, majd a paraméter értéke. Ismételt paraméter esetén az utoljára megadott lesz érvényes. A fájlba megjegyzés is tehető a # jel használatával, ahogyan a példa is mutatja. A fájlok útvonalai lehet abszolút és relatív is, a példában relatív útvonal látható. Hibásan megadott opcionális paraméter esetén az alapértelmezett érték kerül használatra.

További opcionális paraméterek:

<i>timeout</i>	Percen belül megadott érték. A járművezérlők időkorlátja, mely tétlenség esetén lejár és a hídszerver megvonja a vezérlést a felhasználótól. Értéke 1-120 között érvényes. Alapértelmezetten 5 perc.
<i>strict</i>	Logikai érték, értéke <i>true</i> vagy <i>false</i> . Ha <i>true</i> , azok a vezérlők, melyek nem szerepelnek a fehérlistában, nem használhatják a szervert. Alapértelmezzen minden logikai érték <i>false</i> .
<i>quiet</i>	Logikai érték. Ha aktív, a program indulásakor az összes figyelmeztetés inaktív.
<i>hidden</i>	Logikai érték. Ha aktív, a program a háttérben fut és az alkalmazás ikonja nem jelenik meg az értesítési területen.
<i>lang</i>	ISO 639-1 nyelvkód (Pl. en, hu). Az alkalmazás nyelvét határozza meg. Alapértelmezett értéke a rendszer nyelve, de ha a rendszer nyelvéhez nincs fordítás, akkor az angol nyelv kerül használatba.

2.8.5. Engedélyek és prioritások

A hídszerver konfigurációja csak azokat a fix és szükséges adatokat tartalmazza, melyek feltétlen szükségesek az alkalmazás indításához. A program futása alatt, a konfigurációs fájl módosítása nincs hatással a szerverre egészen a következő indításig.

A konfigurációs-fájlon kívül létrehozható három lista is, melyek módosítása esetén nincs szükség a szerver újraindítására, így zavartalanul képes a szerver szolgáltatást nyújtani. Mindhárom lista más-más hatást fejt ki, de mindegyik jogosultságkezelés céljából lett létrehozva és hasonló a működésük. Mindhárom fájlból soronként egy felhasználónév szerepelhet.

A listafájlokat a szerver két helyen keresi: az egyik az a könyvtár, ahonnan a program indítva lett. Ha ott nem található a fájl, akkor az alkalmazás könyvtárában nézi meg és ha ott van, azt használja.

2.8.6. Tiltólista

Előfordulhat olyan eset, hogy a tulajdonos egy konkrét felhasználót ki szeretne vonni a forgalomból, mert mondjuk megszereztek a titkos kulcsát és többé nem megbízható a tanúsítványa.

Ekkor jön jól a tiltólista, mely *blocklist.conf* névre hallgat. A fájl tartalma egy egyszerű felsorolás. Azok a felhasználók, akik ebben a fájlból szerepelnek, nem tudnak kapcsolódni a szerverhez és ha éppen kapcsolódva vannak miközben belekerülnek a listába, a szerver bontja velük a kapcsolatot.

2.8.7. Fehérlista és feketelisták

A fehérlista és a feketelisták együttes használatával precíz jogosultságkezelés végezhető el. Járművenként adható illetve vehető el használati jog minden egyes vezérlő felhasználó esetén.

Mindkét listában járművenként alakíthatóak ki csoportok. Egy csoport a másik csoport végéig tart. A csoportkezdő jel a jármű felhasználójának / és / karakterekkel közbefogva. Ha a fájl nem csoportkezdő jellet kezdődik, akkor az alapértelmezett főcsoportba kerülnek a felsorolt felhasználók. A főcsoport felsorolása az összes specializált csoport alá bekerül, de a duplázódást elkerülve, az alcsoportban már szereplő felhasználók nem kerülnek újra a csoport végére.

A feketelisták általában erősebb, mint a fehérlisták, de csak akkor, ha a fehérlistában megegyező, vagy alacsonyabb prioritású beállítás van érvényben.

Az alábbi példa érhetőbbé teszi a két lista működési elvét.

whitelist.conf

```
controller1
controller2
[host]
controller2
controller3 [v]
```

blacklist.conf

```
controller3
```

A fenti példa alapján a host nevű járműhöz az alábbi sorrend érvényes:

1. controller2
2. controller3
3. controller1

Látható, hogy controller3 a feketelista alapértelmezett felsorolásában szerepel, ezért eredetileg nem láthatna a járműválasztóban egyetlen járművet sem, de mivel a fehérlistában is szerepel a host jármű csoportjában, ezért a host nevű jármű elérhető a számára.

A felhasználónevek sorrendjének szerepe van a jármű vezérlésének a kérések kor. Tegyük fel, hogy controller3 vezérli a host nevű járművet. Ha controller2 szeretné vezérelni a járművet, a hídszerver azonnal átadja az irányítás jogát neki, mivel magasabb ranggal rendelkezik az adott jármű esetén, viszont ha controller1 kéri a vezérlést, várólistára kerül.

A várólista kialakításában is szerepe van a rangsornak. A ranggal rendelkező felhasználók a rangjuknak megfelelően a lista elejére kerülnek besorolásra, a rang nélküliek őket követik a kérésüknek megfelelő idő szerint. Ha tehát controller4 kéri a vezérlést és éppen vezérli valaki a járművet, bekerül a várólista legaljára, mivel nem rendelkezik ranggal és ő az utolsó kérő.

Amikor a felhasználó lemond a vezérlésről és a várólista nem üres, a várólistán soron következő automatikusan megkapja a vezérlést és csak magasabb rangú felhasználó veheti el tőle a jogot.

A fehérlista tehát jogot ad a járműhöz való kapcsolódáshoz és rangsort is állít fel. A feketelista megvonja a járműhöz való kapcsolódás jogát, de csak akkor ha a fehérlistában nem szerepel erősebb utasítás. Ha minden két listában ugyan olyan erős utasítás szerepel, akkor a feketelista van előnyben részesítve.

A fehérlista viszont nem feltétlen ad teljes körű jogot a jármű használatához, mert ha a felhasználó neve után a [v] jelzés szerepel – ahogyan a példában is –, akkor a felhasználó csatlakozhat a járműhöz, használhatja a csetet és láthatja a jármű adatait, de nem kérhet és nem is kap vezérlést.

2.8.8. Naplázás

A szerveralkalmazás a naplózást is támogatja és alapértelmezetten aktív a szolgáltatás. A naplózás a *bridge.log* nevű fájlba történik és a szolgáltatás csak úgy kapcsolható ki, ha erre a fájlról nem ad a fájlrendszer írási jogot.

Naplózásra kerül a szerver indítása és leállítása, minden figyelmeztetőüzenet, a kliensek fel- és lekapcsolódása, valamint a járművezérlések átvétele és lemondása. A naplózás nyelve megegyezik a konfigurációban megadott nyelvvel.

Példa a *bridge.log* fájl tartalmára:

```
2013-07-26 21:04:35,591 INFO A szerver elindult.  
2013-07-26 21:04:41,234 INFO host (jármű) kapcsolódott a hídroz  
2013-07-26 21:04:43,361 INFO controller (vezérlő) kapcsolódott a hídroz  
2013-07-26 21:05:00,219 WARN Duplázott tanúsítvány a 192.168.10.1 címről.  
2013-07-26 21:05:03,948 INFO [host] => [controller]  
2013-07-26 21:05:06,039 INFO [host] <= [controller]  
2013-07-26 21:05:10,175 INFO controller (vezérlő) lekapcsolódott a hídról  
2013-07-26 21:05:34,562 INFO host (jármű) lekapcsolódott a hídról  
2013-07-26 21:06:04,007 INFO A szerver leállt.
```

A hibaüzenetek WARN jelzéssel kerülnek naplózás alá, a közönséges tájékoztatóüzenetek INFO jelzést kapnak. A járművezérlés átvételét => jel és lemondását <= jel jelzi. A példában tehát a controller nevű vezérlő megkapja a host járműhöz való vezérlés jogát és pár másodperccel később le is mond róla.

A naplózás hasznos lehet, ha a szervert megtámadják és tudni szeretnénk a támadó címét, vagy ha utólag valamiért ki kell deríteni, hogy egy bizonyos pillanatban ki irányította a járművet.

2.9. A járműkliens használata

2.9.1. Az alkalmazás használatának előfeltételei

Az Android operációs-rendszer nagy előnye, hogy nyílt forráskódú és bárki fejleszthet rá alkalmazást, amit aztán meg is oszthat az operációs-rendszert használó felhasználókkal. A gyári rendszer éppen ezért nem tartalmaz alkalmazást minden területre, hanem minden felhasználó a szükségleteinek megfelelően választhat és szerezhet be alkalmazást, ezzel a megoldással sok hely takarítható meg, mivel mindenkinél csak a számára hasznos alkalmazásokat telepíti fel.

A járműkliens teljes körű használatához szükség van egy olyan fájlkezelőre, ami képes együttműködni más alkalmazásokkal, valamint az *IP Webcam* nevű alkalmazásra. Én az *OpenIntents File Manager* nevű fájlkezelőt használom, főként azért, mert a rendszer ezt gyárilag tartalmazza. A fájlkezelő megléte ahhoz szükséges, hogy ki lehessen tallózni az SD-kártyáról a tanúsítványfájlokat. Az IP Webcam a kamera hatékony megosztására van használva.

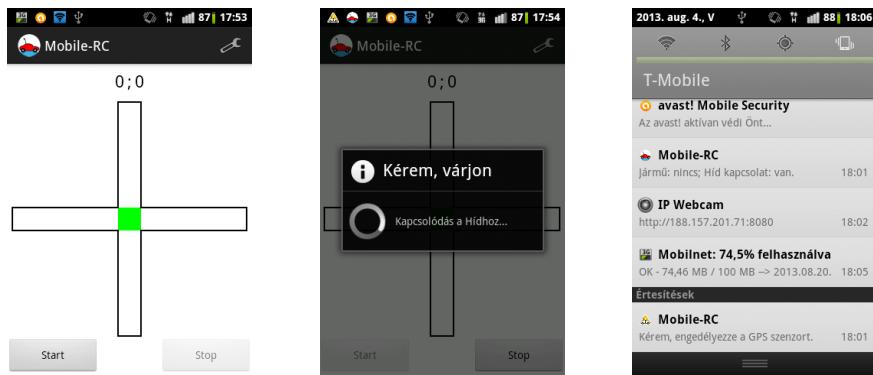
Ha az Android-rendszer verziója 3.1-nél régebbi, az *USB hibakeresés* engedélyezésére is szükség van, mivel az ennél régebbi rendszerek nem támogatják az USB portra köthető kiegészítő eszközöket, helyette a hibakeresőn keresztül folyik a kommunikáció a jármű áramkörével.

Az USB hibakeresés elérhetősége: Beállítások – Alkalmazások – Alkalmazásfejlesztés

2.9.2. Kapcsolódás a Hídszerverhez

A járműkliens oly módon biztosítja az összeköttetést a hídszerverrel, hogy indítása után mindaddig, míg nincs kapcsolat, időközönként megkíséri a kapcsolódást és ha a kapcsolat megszakad, akkor is így jár el. A kapcsolódást az alkalmazás főablakából lehet kezdeményezni és mindaddig aktív marad a kapcsolat, míg ugyan itt le nem állítja a felhasználó. A kapcsolat ideje alatt a program háttérfolyamatként fut, így a főablak bármikor bezárható és előhozható.

Ahhoz, hogy a jármű pontos helyzete elérhető legyen, a GPS szenzor engedélyezése is szükséges. Ha a felhasználó nem engedélyezi, a kapcsolódás kezdetekor az alkalmazás figyelmezteti a felhasználót.



24. ábra: A járműkliens

A hídhoz való kapcsolódás a *Start* gomb megnyomásakor kezdődik meg. A kapcsolódás előtt az alkalmazás ellenőrzi a konfigurációt és megfelelő beállítás esetén elindul a háttérfolyamat.

A háttérfolyamat kétféle üzenettípust képes megjeleníteni:

Figyelmeztetés esetén a háttérfolyamat tovább fut, mivel nincs szükség felhasználói beavatkozásra. *Hibaüzenet* esetén a háttérfolyamat megszakad és az üzenet elolvasása után leáll.

Hibaüzenet több esetben is jelentkezhet. Például ha az IP Webcam alkalmazás nincs telepítve, akkor olyan hibaüzenet jelenik meg, amire kattintva a Google Play áruház előjön és megjeleníti az alkalmazás részleteit, és a felhasználó könnyen telepítheti azt. Ha nem összetartozó tanúsítványok lettek beállítva vagy hibás tanúsítvány-jelszó lett megadva, akkor is hibaüzenet jelenik meg, amire kattintva a beállítások jelennek meg.

A hídhoz való kapcsolódás egyes eszközökön akár 1 percet is igénybe vehet, mivel a memóriakártyáról való tanúsítványok beolvasása időigényes, viszont a tanúsítvány beolvasása csak az első kapcsolódáskor történik meg, és mindaddig amíg az Android nem törli a memóriából, addig gyorsan fog kapcsolódni a kliens a szerverhez.

Sikeres kapcsolódás után megkezdődik a szenzoradatok továbbítása a híd felé, valamint elindul az IP Webcam alkalmazás is, hogy a kamerakép sugárzása is gyorsan indítható legyen szükség esetén. Sajnos az Android fejlesztői nem adtak arra lehetőséget, hogy a kameraképet anélkül kérje le a programozó, hogy azt valamilyen formában ne jelenítse meg a rendszer, ezért a program az indulásakor felhozza a főablakát, amit bármikor háttérbe lehet tenni, viszont bezárnival nem ajánlott, mert akkor leáll a program és a járműkliens a futása közben úgy is újra meghívja.



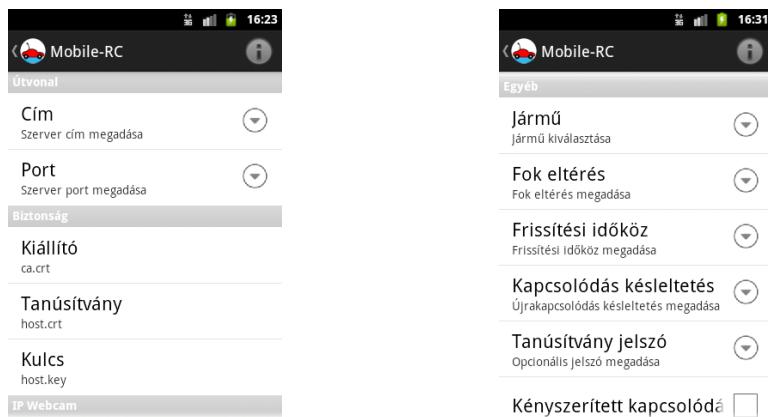
25. ábra: IP Webcam

Az adatforgalommal való takarékosság érdekében a szenzoradatok küldése tömörített csomagokban történik és ha nincs egyetlen vezérlőkliens se kapcsolódva a járműhöz, a kliens nem közvetíti a kameraképet a szerver felé. Az adatforgalommal való takarékosság fő oka a mai mobilinternet-tarifák adatforgalom-korláatos mivolta.

A háttérfolyamat a *Stop* gombbal állítható le, ekkor a kliens bezárja a szerverrel kiépített kapcsolatot és a főablak bezárásakor az IP Webcam is leáll.

2.9.3. A járműkliens konfigurálása

Ahhoz, hogy a hídroz kapcsolódni lehessen, a felhasználónak meg kell adnia annak pontos címét és a tanúsítvány-fájlok helyét. Ezen kívül még további hasznos beállítási lehetőségek is elérhetőek a főablakból előhozható Beállítások ablakból.



26. ábra: Járműkliens beállítások 1.

Mivel az IP Webcam szolgáltatása belső hálózaton bármelyik webböngészőből elérhető, lehetőség van felhasználónévvel és jelszóval levédeni a videószervert. Ezen adatokat a vezérlőkliens számára is meg kell adni. Ezen kívül megadható az is, hogy a szerver melyik porton figyeljen. Ha a szervert a vezérlőkliens indítja el, akkor az abban megadott beállítások lesznek az érvényesek akkor is, ha az IP Webcam konfigurációjában más szerepel.

Az akkumulátor-szint százalékos megbecsüléséhez szükség van a feszültséghatárok megadására. A maximum feszültség az a feszültség, mely az akkumulátorok teljes töltöttsége esetén olvasható le a főablakról. A minimum feszültség ezzel ellentétben az a feszültség, mely még éppen hagyék képes meghajtani a jármű motorját.

A program fel van készítve több jármű kezelésére is. Az egyszerűbb járművek, mint az általam elkészített prototípus is, nem rendelkeznek olyan erős motorral, hogy érdemes legyen őket precízen irányítani, de vannak olyan gyors járművek, melyeknél felmerülhet a gázadagolás lehetősége. A járműválasztóban jelenleg csak a *Prototípus* és a *PWM teszt* opciók közül lehet választani. Az utóbbi választása esetén megszabható a jármű sebessége az egérrel való irányításkor.

Ha a telefonban a mágneses-szenzor pontatlan, és tudható, hogy hány fokban tér el a megfelelő iránytól a kapott adat, akkor az eltérést kompenzálni lehet a *Fok eltérés* opció átállításával.

A Frissítési időköz megadásával takarékoskodni lehet az adatforgalommal. Ez megint csak mobilinternet használata esetén lehet hasznos funkció. Ha az érték nulla, akkor a szenzoradatok egyből küldésre kerülnek, amint változtak. Az értéket ezredmásodpercben kell megadni, tehát ha az érték 1000, akkor minden másodpercen egyszer frissülnek a szenzoradatok, ezáltal a jármű pozíciója és iránya.

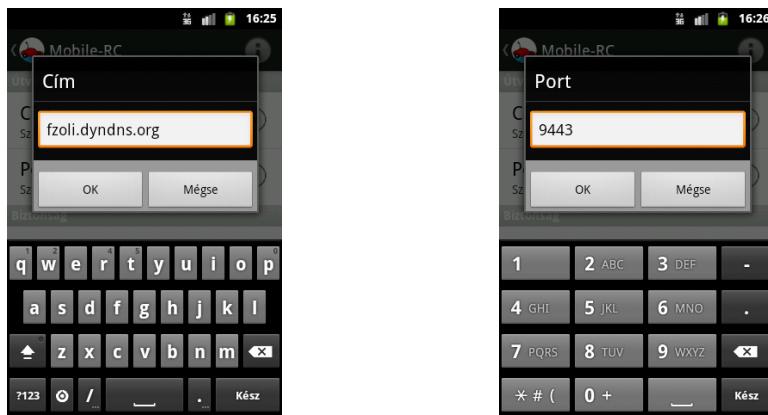
Ha a kapcsolat megszakad, a kliens megpróbál újra kapcsolódni a szerverhez. Hogy a kapcsolódási kísérletek ne terheljék a hálózati forgalmat és a processzort, a kapcsolat megszakadása után nem azonnal történik meg az újrakapcsolódás. A *Kapcsolódás késleltetés* opció beállításával testreszabható, hogy mennyi időközönként kísérelje meg a kapcsolódást a kliens.

Az alkalmazás alapértelmezetten meg se próbál kapcsolódni a szerverhez, ha úgy észleli, nincs aktív hálózati kapcsolat. A hálózatok működési elvében jártasabb felhasználók viszont használhatnak olyan alternatív módszereket, melyeket a program nem tud detektálni. A *Kényszerített kapcsolódás* opció aktiválásával kikapcsolhatják a hálózat elérhetőségének ellenőrzését. Például ha mobilhálózatot vagy Wi-Fi hálózatot használ a felhasználó, akkor ez az

opción maradhat kikapcsolva, de ha mondjuk csak az internetmegosztást használja és laptoppal kapcsolódik a telefonhoz, akkor szükség van ennek az opciónak a bekapcsolására.

A jármű irányításához a régebbi Android rendszereken szükség van az USB hibakeresés bekapcsolására, amit a fejlesztők röviden ADB-nek neveznek. Az ADB eredetileg egy hasznos hibakereső interfész, de használatával a teljes rendszer konfigurálható így a jármű vezérlése is megoldható rajta keresztül. Az ADB viszont nem kis potenciális veszélyt jelent, ha állandóan be van kapcsolva, mivel a felhasználó személyes adatait, jelszavait hozzá értő könnyen megszerezheti. Az ADB bekapcsolása ezért csaknél kizárolag rendszeralkalmazásoknak megengedett. Tehát az én alkalmazásomból nem lehet bekapcsolni az USB hibakeresést, mert nem rendszeralkalmazás. Létezik viszont egy AdbToggle nevezetű program, ami rendszeralkalmazásként telepíthető rootolt rendszereken. Az alkalmazásom támogatja ezt az alkalmazást, és ha telepítve van, induláskor automatikusan bekapcsolja az ADB-t, ha szükséges és le is állítja azt a leállásakor. A felhasználó az *ADB bekapcsolva marad* opciónnal megadhatja, hogy szeretné-e hogy bekapcsolva maradjon az USB hibakeresés az alkalmazás leállítása után, de ez az opción csak akkor érhető el, ha az AdbToggle telepítve van. Ha nincs telepítve és az ADB ki van kapcsolva, az alkalmazás az indulásakor figyelmezteti a felhasználót.

Az alkalmazás *offline üzemmódban* is képes működni. Offline üzemmódban a szolgáltatás indításakor a program nem kapcsolódik a hídszerverhez, csak a jármű áramköréhez. Offline üzemmódban a jármű könnyen tesztelhető. A főablakon található irányító panellel pontosan vezérelhető a jármű. Tehát ha a felhasználó vicces kedvében van, „megtáltathatja a kedvencét”, nem kell attól tartania, hogy elkóborol a járműve.



27. ábra: Járműkliens beállítások 2.

A felhasználói útmutató véget ért.

3. Betelekintés a háttérbe

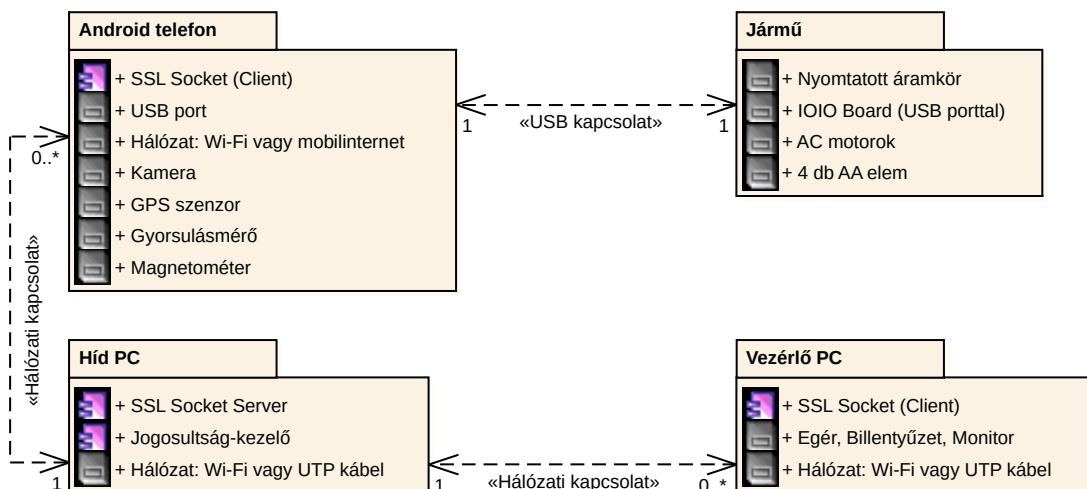
Sokat gondolkadtam azon, hogyan lehetne a diplomamunkám működésének elvét a legérhetőbb módon átadni az olvasónak. Végül arra jutottam, hogy leírom azt, hogyan terveztem meg az alkalmazásokat lépésről-lépésre. Ebben a fejezetben felvázolom az alapötletemet, majd ennek részeit mélyebb szinten kibontom és leírom azt is, hogy hogyan változott az elképzélésem és milyen lehetőségek közül választthattam és miért választottam azt, ami végül kivitelezve lett.

3.1. Eszközválasztás

Első körben azt kellett eldöntenem, hogy milyen eszközöket fogok felhasználni a kivitelezéshez. A jármű adott volt, mivel már rendelkeztem egyel, ami eredetileg a kiskutyák játékszere volt, prototípusnak tehát tökéletesen megfelelt. Mivel az Android fejlesztőeszközével már foglalkoztam a GDF diákműhelyében, tudtam, hogy Java nyelven könnyen lehet rá programokat fejleszteni és mivel az volt az első objektum-orientált nyelv, amit megtanultam, számonra egyszerű benne programozni. A diákműhely kedvet adott arra is, hogy vegyek egy okostelefont és mivel már telefonnal is rendelkeztem, az Apple okostelefonjai nem is kerültek számításba.

Az elején nem volt egyértelmű, hogy a telefont hogyan fogom az áramkörrel összeköttetésbe hozni. Választthattam USB illetve Bluetooth alapú kommunikáció között. Hamar eldöntöttem, hogy inkább USB kábelt fogok használni, mert egyszerűbb, megbízhatóbb és biztonságosabb is a támadások ellen, de mivel ez a terület teljesen új volt számonra, utána kellett járnom, hogyan valósíthatom meg az USB-porton keresztül az áramkör vezérlését és annak is, hogy hogyan tudom elérni, hogy a motorok mindenkorban mozgathatóak legyenek.

Az interneten két olyan elektronikai fejlesztőplatformot találtam, ami képes USB kapcsolaton kereszti kommunikációra és működik Android alapú telefonokkal is. Az első találatom az Arduino^a IDE volt, a második a IOIO for Android^b. Végül a IOIO Board mellett döntöttem, mivel kimondottan Android telefonokhoz lett tervezve, és nincs szükség a mikrovezérlő programozására.



28. ábra: Eszközök és kapcsolatuk

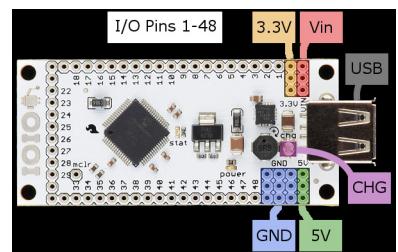
Az eszközválasztás után egy az ábrához nagyon hasonló terv állt össze a fejemben. A bevezetésben leírt észrevételeimet is ekkor írtam le. Tudtam, hogy a telefon szenzorait is fel fogom használni helymeghatározásra és azt is, hogy három alkalmazást fogok készíteni, és hogy a két klienst összekötő szerver fogja a teljes jogkezelést elvégezni. Meg voltam róla győződve, hogy képes vagyok megvalósítani a tervemet, de mivel eddig kizárálag szoftverfejlesztéssel foglalkoztam, eljött az ideje, hogy mélyebbre ássak az áramkörtervezésben is.

3.2. Az áramkör megalkotása

Miután eldöntöttem, hogy három alkalmazást fogok létrehozni és hogy mely eszközökön fognak futni, a szoftverfejlesztést kis időre félretettem és az elektronikai részt vettet elő. Azért döntöttem így, mert kliens-szerver alapú alkalmazásokat már nem egyszer írtam és biztos voltam benne, hogy a programozással nem lesz problémám, de abban már nem voltam biztos, hogy az áramkört is meg tudom valósítani. Úgy gondoltam, hogy ha még sem sikerül, még választhatok egy másik diplomamunka téma kört és viszonylag kevés idő fog rámenni erre a próbálkozásra, de azért bíztam a sikerben. Mivel a IOIO Board kizárolag Amerikából szerezhető be, hogy még kevesebb időre legyen szükségem, Interneten keresztül megrendeltem, hogy mire az áramkör terve elkészül, rendelkezésemre álljon az eszköz.

3.2.1. IOIO for Android

A IOIO (angol kiejtéssel yo-yo) kimondottan Android 1.5 és az felettes rendszerekhez lett tervezve. Gyártója a SparkFun Electronics. Alapértelmezetten USB-kapcsolaton keresztül kommunikál, de kiegészítő eszközzel képes Bluetooth hálózaton keresztül is kapcsolódni a telefonhoz. Vezérléséhez nincs szükség mikrovezérlőjének programozására. A gyártó oldaláról letölthető egy könnyen használható Java API, amit az Android alkalmazás projekthez hozzá lehet adni és onnantól kezdve Java nyelven lehet



29. ábra: IOIO Board

utasításokat adni a IOIO számára.

Az eszközön 48 vezérelhető tű (pin) van, nagy részük digitális ki- és bemenet attól függően, milyen célra használjuk fel, de a 31-től 46-ig számozott tűk kizárolag analóg bemenetnek használhatók fel. Az eszköz nem rendelkezik analóg kimenettel, de támogatja a PWM-et a digitális kimeneteken. A PWM-ről azaz az impulzusszélesség modulációról később még szó fog esni.

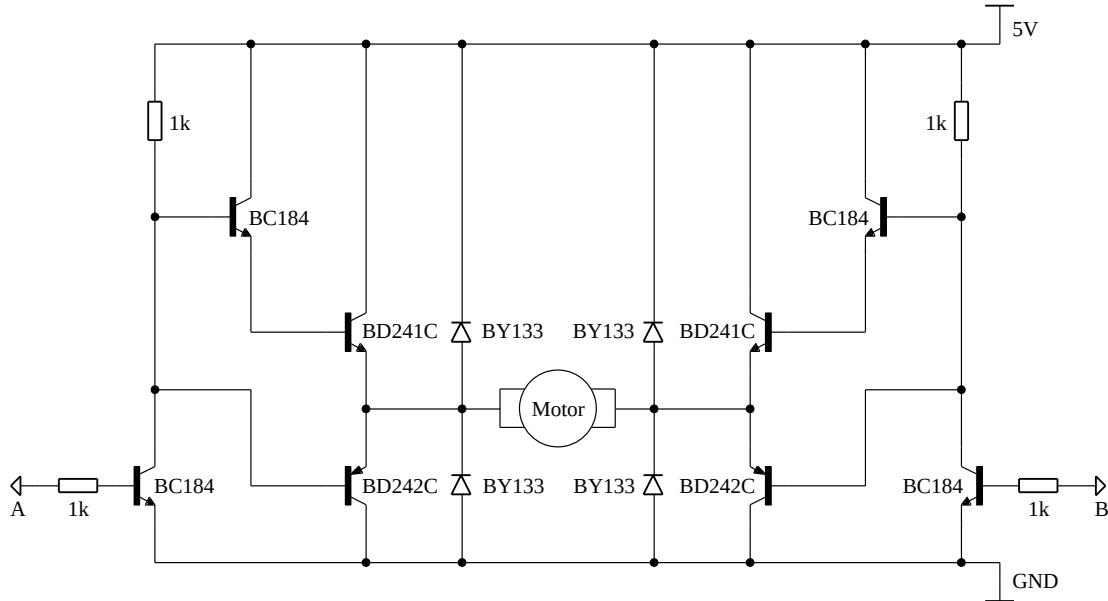
A telefonnal való kommunikációt az eszköz az ADB-n (Android Debug Interface) keresztül valósítja meg, amennyiben az USB hibakeresés engedélyezve van. Azokon a rendszereken, melyek már támogatják az Open Accessory protokollt (Android 2.3.4-től), nincs szükség az ADB engedélyezésére.

3.2.2. Kezdeti lépések

Korábban említettem, áramkört még soha nem terveztem önállóan, konkrét céllal. Szerencsére volt egy ismerősöm, aki otthon volt ebben a téma körben és tudtam tőle útmutatást kérni. Az első és legfontosabb kérdésem az volt – ismervén a IOIO eszköz Java nyelvre írt API-ját –, hogy hogyan tudok felhasználni két digitális kimenetet úgy, hogy egy egyenáramú motort mindenkor irányba el tudjak indítani. Hasonlóképpen, ahogyan én keresek megoldást konkrét problémára programozás közben, ő is keresett és talált az interneten egy olyan áramköri rajzot^c, amit kezdők is képesek megépíteni, mégis megbízhatóan működik.

A tervrajzon azonban amerikai eszköznevek voltak feltüntetve így meg kellett keresni az itthon kapható eszközök amerikai megfelelőit. A főiskolán tanult Micro-Cap alkalmazásban kipróbáltam az áramkört és azt tapasztaltam, hogy pont úgy működik, ahogyan szeretném. Ez után próbápanelen is megépítettem az áramkört, hogy tesztelni tudjam a valóságban is a jármű motorjaival.

Az áramkörhöz szükséges eszközök beszerzése után nekiláttam a tesztáramkör megépítésének, de hamar rájöttem, hogy nem olyan egyszerű az áramkör megépítése a valóságban, mint a szimulátorban.



30. ábra: BJT H-híd, feszültség-polaritásváltó áramkör

Az ábrán látható, hogy az áramkör főként bipoláris tranzisztorokból áll. A nehézséget az jelentette számomra, hogy a BC184 NPN-tranzisztor lábai pontosan C-B-E sorrendben helyezkednek el, ahogyan az ábrán is, viszont a BD241C és BD242C tranzisztorok lábai B-C-E sorrendben követik egymást, tehát a kollektor és a bázis fel vannak cserélve a két tranzisztor esetén. A másik ami megzavart, hogy a BD242C tranzisztor nem NPN, hanem PNP típusú tranzisztor, de ennek ellenére a specifikációban ugyan abban a sorrendben vannak a lábak felsorolva, mint az NPN típusú társánál, a BD241C tranzisztoronál. A diódákon szerencsére egyértelműen jelölve volt vonallal, hogy melyik a katód, és mivel az áramköri rajzokon is vonallal jelölik, nem volt gond a bekötésével. Azt is észrevettem, hogy az ellenállások jelölése teljesen szimmetrikus. Ez azért van, mert az ellenállásokat teljesen mindegy, milyen irányba kötjük be.

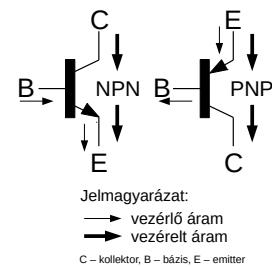
3.2.3. Az áramkör felhasználása

Ahhoz, hogy egy egyenáramú motort meg lehessen hajtani tetszőleges irányba, tulajdonképpen a motorra eső résznél tudni kell szabályozni a feszültség polaritását. A BJT H-híd nevű áramkör ezt bipoláris tranzisztorokkal (Bipolar Junction Transistor) oldja meg, innen is származik a neve.

A szoftverfejlesztők számára mellékelek egy képet, ami érthetőbbé teszi az előző részben emlegetett tranzisztorok lábait és azok típusait.

Ezeket a tranzisztorokat erősítőkben, szabályzó és kapcsoló áramkörökben használják. A BJT 3-kivezetésű elektronikai alkatrész. Három elektromosan szétválasztott réteggel rendelkezik, innen származik az NPN és a PNP megnevezés.

A három kivezetés neve: kollektor (C), bázis (B), emitter (E).



31. ábra: Bipoláris tranzisztor

A feszültség polaritásának szabályozása az áramkör szélein található két digitális feszültségforrás (jelük A és B) segítségével történik. Ha egyik feszültségforrás se aktív, a motor áll, viszont ha a kettő közül az egyik aktív, akkor a motor forog egyik, illetve másik irányba attól függően, melyik feszültségforrás lett aktiválva. Az áramkör zárlat ellen is védve van, így ha minden feszültségforrás aktív, akkor is áll a motor, de nem tanácsos a tüzzel játszani.

3.2.4. A nyomtatott áramkör megtervezése

3.2.4.1. Az első lépések

A tranzisztorok lábainak eltérő sorrendje azt jelentette számomra, hogy nem tudom pontosan úgy megépíteni az áramkört a valóságban, ahogyan a tervben szerepel, mert akkor sok helyen kéne áthidalni a vezetéseket. A próbapanelen az összekötőkábeleket úgy rendezhettem, ahogyan akartam így ott ez még nem is okozott gondot, de a végleges nyomtatott áramkör (NyÁK) esetében már nem lett volna szép megoldás az áthidalás.

Megrajzoltam egy olyan áramkört, ami megfelelt az eredeti áramkörnek, de már a felhasznált elektronikai alkatrészek kivezetései alapján lettek elhelyezve a vezetőrétegek. Az eredeti áramkör szimmetrikus felépítését is megtartottam és a könnyebb összeszerelhetőség érdekében úgy terveztem meg az áramkört, hogy minden tranzisztor egy irányba nézzen. Ezzel tulajdonképpen a NyÁK-terv egy részét alkottam meg.

Mivel a járműben két motor is van, tudtam, hogy a nyomtatott áramkörön az általam megrajzolt tervnek kétszer is szerepelnie kell és az IOIO eszközöt helytakarékosan el kell tudnom helyezni a lapon. Később jött még az az ötletem is, hogy a prototípus járműbe gyárilag beépített világító diódákat (LED) megtartom, de teszek a NyÁK-ra egy kapcsolót, amivel ki-be lehet kapcsolni őket.

3.2.4.2. NyÁK-tervező program választása

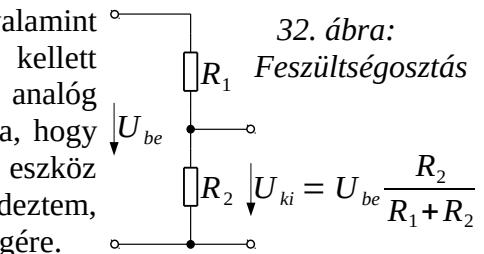
A nyomtatott áramkör egyszerű megtervezése érdekében kerestem egy Linux alatt is futó NyÁK-tervező programot. Több alkalmazást is találtam, de legjobbnak az EAGLE bizonyult a CadSoft-tól. Az alkalmazásban nekem az tetszett meg, hogy minden alkatrész méretarányos benne és támogatja a PDF-be való exportálást. A PDF fájlra azért volt szükségem, mert nem rendelkezem lézernyomtatóval, viszont a NyÁK-ra csak lézernyomtatóval készült terv vasalható rá. A PDF megtartja a méretarányokat, és a boltban, ahol nyomtatnak lézernyomtatóval, van PDF nézegető telepítve a gépre.

Ettől a ponttól kezdve tudtam, hogy meg fogom tudni valósítani az áramköri részt és míg várakoztam a postára, hogy meghozza az IOIO eszközt, elkezdtem a három alkalmazás tervezését és fejlesztését. Miután a posta megérkezett, és le tudtam mérni az IOIO és kivezetései pontos méretét, nekiláttam a végleges NyÁK megtervezésének.

3.2.4.3. A NyÁK kialakulása

A legkisebb helyfoglalás érdekében a két H-híd egymás alá került és el lettek forgatva 180 fokkal, hogy az IOIO digitális kimenetei egyszerűen beilleszthetőek legyenek és így a tápellátást is el tudtam vezetni az egyik ellenállás kivezetései között.

Miután elkészülttem a két H-híd és az IOIO összekötésével, valamint az áramellátással, már csak az akkumulátor-szint mérését kellett beleterveznem az áramkörbe. Az IOIO rendelkezik analóg bemenettel, de a dokumentációban^d felhívta a figyelmet arra, hogy ha a bemenetre 3,3 V feszültségnél több kerül, az eszköz meghibásodhat. Újra találkoztam az ismerősömmel és megkérdeztem, hogyan oldaná meg a feszültségmérést ügyelve az eszköz épsgére.

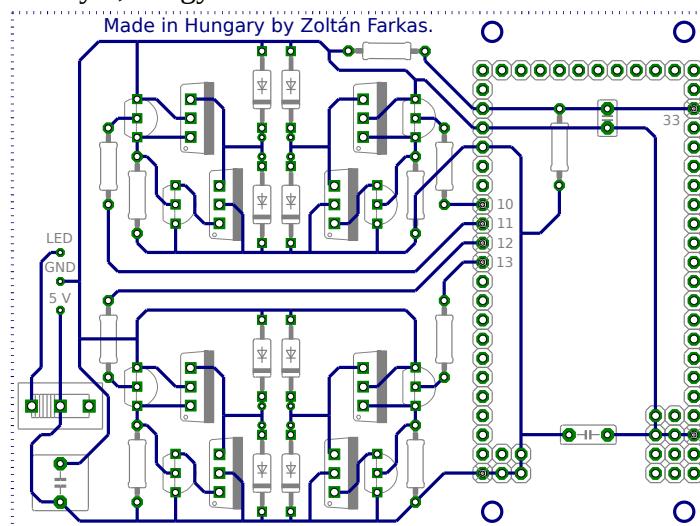


Két lehetőség merült fel. Zéner-dióda és ellenállások használatával megoldható, hogy eltérő feszültségforrások használata esetén se menjen a feszültség 3,3 V fölé a bemenetnél, de ha a tápellátás nem változik, akkor a feszültségesztás^e is alkalmazható. Mivel a járművet kizárolag négy darab Nikkel-Cink (NiZn) akkumulátorral táplálom, a feszültség soha nem megy $4 \cdot 1,6$ V, azaz 6,4 V fölé, tehát elég a két ellenállás használata, nincs szükség feszültségstabilizáló Zéner-diódára.

Az ismerősöm tanácsára az áramkörön elhelyeztem egy elektrolit kondenzátort és két kerámia kondenzátort. Az egyik kerámia kondenzátorra azért volt szükség, mert nagyon ingadozott a mért feszültség, és a kondenzátor rövidtávú energiatárolása nagyban csökkentette az ingadozást. A másik két kondenzátor biztonsági célból lett elhelyezve az IOIO tápellátásának segítésére.

Az IOIO áramkörhöz való kapcsolását nem közvetlenül terveztem, hogy bármikor levehessem a NyÁK-ról és más projektekben is felhasználhassam, ezért tüskesor is fel lett használva. Ez azért is fontos, mert az IOIO alá így befért több alkatrész is, egész pontosan egy ellenállás és két kerámia kondenzátor. Az elektrolit kondenzátort a kapcsoló mellett, az áramkör másik végén helyeztem el, mivel csak ott volt elegendő hely számára.

A kondenzátorok elhelyezésével és bekötésével a nyomtatott áramkör elnyerte végső formáját. Miután a panelen megjelöltetem az IOIO rögzítési pontjait maradt még egy kis szabad hely, ahol feltüntethettem a gyártás helyét, ahogyan azt szokás.



33. ábra: A nyomtatott áramkör

Miután végeztem a tervezéssel, az egész áramkört kilenc példányban elhelyeztem egy A4-es lapon és beállítottam az EAGLE-t hogy csak a vezetéseket és a furatokat jelenítse meg. Ez után fekete-féhérre PDF fájlba exportáltam a lapot, amit lézernyomtatóval kinyomtattattam és megkezdtem életem első nyomtatott áramkörének az elkészítését.

3.2.5. A nyomtatott áramkör kivitelezése

A NyÁK elkészítésének több útja is van. A legegyszerűbb a kézzel való megrajzolás, ám ez azért időigényes, mert minden kézzel kell kimérni, hogy pontosan minden furat a helyén legyen. Az áramkör tervezése közben arra gondoltam, hogy ha már úgy is készítek egy méretarányos, teljes tervet, megoldható-e otthoni körülmények között, hogy ezt a tervet a panelre vigyem. Két elkészítési módszert találtam: vasalásos^f- valamint fotótechnikás^g NyÁK készítés.

3.2.5.1. Fotótechnikás NyÁK készítés

Az eljárás folyamán nem hagyományos, hanem fényérzékeny lakkal bevont nyáklapot használunk. Ez a fényérzékeny réteg UV fény hatására olyan kémiai reakció megy át, aminek eredménye képpen ez a lakk réteg lúggal oldható lesz. Ahol a lakkot nem éri UV fény, ott a lakk ellenáll a lúgnak. Az előre megtervezett nyáktervet átlátszó fóliára nyomtatjuk, majd a fényérzékeny nyáklapunk elé helyezzük. A lézernyomtató festéke ellenáll az uv fénynek, így a mögötte lévő nyákfelület védett marad a fénnel szemben. Így lényegében a kinyomtatott nyáktervet a nyákról rávilágítjuk. Ahol UV fény érte a nyákat, ott lemosható lesz a védőlakk, ahol nem érte fény, ott ellenálló marad. Az oldható lakkot lúgoldatban lemoszuk, így ott tiszta réz felület lesz, amit a maratószer lemar. A védett réteg pedig megmarad, ami pedig maga a nyákrájz.

3.2.5.2. Vasalásos NyÁK készítés

Véleményem szerint ez az eljárás biztonságosabb és egyszerűbb fotótechnikás eljárásnál. Biztonságosabb, mert nincs szükség UV fényre, így a szem nincs sugárzásnak kitéve. Egyszerűbb, mert nincs szükség lúgoldatra, se átlátszó fóliára, csak egy sima talpú vasalóra és egy fényezett fotópapírra. Ezen okokból kifolyólag én ezt a módszert választottam.

Az eljárás során a lézernyomtató tintáját égetjük rá a nyáklapra, majd a lapot, amire az áramkör nyomtatva volt, leáztatjuk a nyáklapról. A nyáktervet tükrözve kell kinyomtatni, mivel a kinyomtatott lapot értelemszerűen a tintás oldalával lefelé vasaljuk rá a nyáklapra. A nyomtatáshoz a legtöbb 135 grammos műnyomópapírt ajánlanak, de fotópapírt is lehet használni, lényeg, hogy a lap tömege 130-150 g között legyen, hogy könnyen vasalható és áztatható legyen. Fontos, hogy a lap, amire nyomtatunk, fényes felületű legyen, hogy ne szívja magába a lézernyomtató festékét, és kizárolag lézernyomtatót használunk, mert a tintasugaras nyomatók víz alapú tintája soha nem fog a vasaló hőjétől megolvadni és átfolyni a nyákra. A lap fényes felét nem tanácsos kézzel megfogni és a nyáklapot is meg kell tisztítani víz alatt finom csiszolópapírral még a vasalás előtt. Ezzel a tinta nyákhöz való tapadását segítjük elő.

A tinta ráégetése a nyákra vasalóval történik, de akinek van, laminálógéppel egyszerűbben megteheti. A megfelelő hőmérséklete 200-220 °C között van, de ez változhat a papírlap vastagságától függően. A vasalás akkor fejeződik be, amikor az áramkör már a lap másik oldaláról is látható, ez után már csak le kell áztatni a papírlapot és középről kifelé haladva óvatosan leszedni. A papírlap leszedése után láthatjuk, hogy a teljes nyákrajz rákerült a nyákra.

3.2.5.3. A NyÁK maratása

Miután a fentebb bemutatott módszerek valamelyikével a nyákrajz elkészült, a rézréteg lemaratása következik. A maratáshoz vas-klorid marató oldatot használtam. Kerestem a háztartásban egy akkora műanyagedényt, amibe belefér a nyák, de nem is túl nagy, hogy ne kelljen sok vas-kloridot felhasználni. Fém alapú edényt nem használhattam, mert a maratószer azt is megtámadta volna.

Az edénybe belehelyeztem a nyákot és annyi vas-klorid oldatot öntöttem bele, hogy a nyákot elfedje. A maratás folyamatának gyorsításához a műanyagedényt forró vízbe nyomtam bele és az edény két szélét fel-le mozgattam, hogy az oldat mozogjon. Az edény mozgatásával azt is láttam, hogy hogy áll a maratás folyamata és a nyákot időben ki tudtam venni, még mielőtt a vas-klorid a tinta alapú védőréteg széleit is elkezdte volna marni.

A maratás után a festéket lakkbenzin hígítóval eltávolítottam a nyákról majd vízzel lemostam. Hogy a vezetőréteget védjem a korroziótól, kémiai ónozó oldatba helyeztem a nyákot, mely matt felülettel bevonta a rézfelületet.

3.2.5.4. A NyÁK befejezése

A maratás után a vezetőrétegek kialakultak, a furatok helyei is megvoltak, következhetett tehát a furatok elkészítése. Az itthon található fúrógépek egyike sem volt képes az 1 milliméternél kisebb átmérőjű fúrófejek befogására, ezért szinte az összes furat 1 mm átmérőjű lett.

A furatok elkészülése után jöhett az elektronikai alkatrészek beforrasztása. Itthon rendelkezésemre állt egy forrasztópisztoly, ami meg is felelt erre a célra, bár a forrasztásban semmi gyakorlatom nem volt, ezért elég sok forrasztó hegy tönkrement. A forrasztást nehezítette az is, hogy túl vékony vezetésekkel terveztem és volt, hogy feljött a rézréteg. Az áramkör vezetését multiméterrel ellenőriztem, és addig javítottam, míg megfelelően nem működött. Végül kész lett a nyomtatott áramkör.

3.2.6. A nyomtatott áramkör felhasználása

A kész nyákra csavarokkal rögzítettem az IOIO eszközt, és az IOIO alatt a nyákra fúrtam két lyukat is, hogy rögzítés céljából a járműhöz hozzá tudjam csavarni. Természetesen még mielőtt a nyákat rögzítettem, az eredeti nyákat eltávolítottam és a kábeleit felhasználtam a motorok bekötésére. A járművön lévő kapcsoló kábelét felhasználva elláttam árammal a nyákat és a LED-ek tápellátását a kapcsolóról áthelyeztem a saját áramköröm kapcsolójának kivezetésére.

Mire a nyomtatott áramkörrel elkészütem, az Androidra írt alkalmazásom is olyan szintre jutott, hogy tesztelni tudtam vele az áramkör működését. Ezt a funkciót meg is hagytam benne és elneveztem Offline módnak. Az áramkör megfelelően működött, így az elektronikai résszel el is készülttem, folytathattam a szoftverfejlesztést.

3.3. A szoftverek megalkotása

Általában a logikai tervet és a program forráskódját egyszerre szoktam írni. Ez azért van, mert valójában minden fejben tervezek meg. Ennek az elérésében sokat segít az objektum-orientált tervezés, mert nagyon jól részekre tudom bontani az alkalmazást és így minden csak az aktuálisan fontos részre kell koncentrálnom.

A legelején eldöntöm, mit is fog a program csinálni és mi szükséges hozzá. Ez alapján kigondolok egy általános tervet. Ekkor dől el, milyen részeiből fog összeállni a program és ez után megyek mélyebben bele az egyes részekbe.

Ebben a fejezetben bemutatom az írt alkalmazások fontosabb részeit, a hozzájuk felhasznált programkönyvtárakat (lib) és azt, hogyan változott a logikai terv az idő múlásával.

3.3.1. A hálózati kommunikáció felépítése

A szoftverfejlesztést a hálózati kommunikáció megtervezésével kezdtem. Mivel mindenkor alkalmazás Java nyelven íródott, a hálózati kommunikáció az egyetlen közös rész a három alkalmazásban és egyben a legfontosabb is, hiszen az alkalmazások használhatósága nagyban függ a hálózat képességeitől.

Régebben már foglalkoztam kliens-szerver architektúrával Java alkalmazásokban, így könnyen el tudtam dönten, hogy nem veszek igénybe külső programkönyvtárat, mint például az Apache HttpClient nevű projektje, mert jelen esetben a HTTP protokoll használata nem lenne kifizetődő.

A hálózaton keresztül egy időben zajlik a kamerakép és az üzenetek közvetítése. Mindez valós időben történik és szükség van a hálózat ellenőrzésére is, ezért saját magam építettem ki a hálózati kommunikációt TCP protokollt használó socketekre támaszkodva.

3.3.1.1. TCP Socket a Javaban

A Java SDK alapértelmezettben támogatja a socketek használatát. A szerver szerepét betöltő alkalmazás használja a *ServerSocket* nevű osztályt, melynek használatához elegendő megadni a konstruktőrben egy portot, amit lefoglal magának az alkalmazás. A kliens alkalmazás esetében a kapcsolat létrehozásához a *Socket* nevű osztályt kell példányosítani, mely két paramétert vár: cím és port. Amint egy kliens kapcsolódik a szerverhez, a szerver oldalon a *ServerSocket accept()* metódusa visszatér egy *Socket* objektummal, ami összeköttetésben áll a kliens oldal *Socket* objektumával. A *Socket* objektumtól elérhető a kimenő és a bejövő folyam referencia. Ettől a ponttól kezdve úgy használható az *InputStream* és az *OutputStream* objektum, ahogyan a fájlkezelésnél is.

3.3.1.2. Eszköz- és kapcsolataazonosító

Ahhoz, hogy többféle információt lehessen közvetíteni a hálózaton egy időben, a klienseknek nem elég egy kapcsolatot létrehozni a szerverrel. Tehát az MJPEG folyam és az üzenet objektumok külön csatornán kerülnek közvetítésre. Ez egyből felveti azt a problémát, hogy honnan fogja tudni a szerver, mi célból kapcsolódott hozzá egy kliens.

Az egyik megoldás lehetne a különböző port használata. Ez esetben két port lenne a szerveren lefoglalva és két ServerSocket lenne használva. Attól függően, hogy melyikhez kapcsolódik a kliens, megtudható, hogy mi a célja. Ezt a megoldást viszont elvetettem, mert szerettem volna úgy megoldani, hogy csak egy portot foglaljon le a szerver a takarékkosság és a könnyebb kezelhetőség érdekében.

Azt találtam ki, hogy kapcsolatazonosítót vezetek be, melyet a kliens küld el a szervernek attól függően, milyen céllal veszi fel vele a kapcsolatot. Tehát minden kapcsolatfelvétel úgy történik, hogy a kliens elküldi és a szerver fogadja az első bájtot, ami a kapcsolatazonosító és ez alapján cselekednek. Az azonosítót küldhette volna a szerver is, az alapján, hogy mely kapcsolatazonosító nincs még használatban a kliens által, de a kliens oldal jobb választásnak tűnt a kevesebb adminisztráció miatt.

A kapcsolatazonosító segítségével tehát a szerver már be tudta azonosítani a kapcsolódás célját, de azt még nem, hogy melyik alkalmazás kapcsolódott hozzá. A kapcsolatazonosító ugyan erre a célra való felhasználása szintén megoldható lett volna az eltérő azonosító használatával, de a könnyebb átláthatóság érdekében szerettem volna, hogy minden kliensalkalmazás ugyan azt az azonosítót használja, ha ugyan az a céljuk. Például ha a járműkliens a kameraképet akarja feltölteni, és a vezérlőkliens is a kameraképet akarja letölteni, akkor használják ugyan azt az azonosítót.

Ezen probléma megoldására bevezettem az eszközazonosítót is, amit szintén a kliens küld a szervernek még a kapcsolatazonosító előtt. Így végül a második bájt lett a kapcsolatazonosító és az első bájt az eszközazonosító.

Itt még mindig nem ért véget a történet. A szerver most már tudja az eszköz típusát és a kapcsolatteremtés célját, de nem tudja elkülöníteni az alkalmazásokat egymástól, vagyis nem tudja megmondani azt, hogy ki is az, aki kapcsolódott hozzá. Ezen okból is szükséges volt bevezetni a felhasználó-azonosítást. A felhasználónév alapján már elkölníthetők az alkalmazások, mivel a rendszert úgy alakítottam ki, hogy egy időben csak egy alkalmazás használhatja ugyan azt a felhasználónevét.

3.3.1.3. Titkosított kapcsolat és felhasználó azonosítás

Manapság az olcsó kategóriájú okostelefonok egészen jó árban vannak, viszont így is több tízezer Forintba kerülnek, tehát nem árt a telefonra vigyázni. Éppen ezért úgy döntöttem, hogy a kapcsolatot titkosítom és csak az általam kiállított tanúsítvánnyal rendelkező klienseket engedem fel a szerverre, így kerülve el a jogtalan hozzáférést vagy az alkalmazás megkerülését.

Első körben arra gondoltam, hogy felhasználónév-jelszó párral azonosítanám a felhasználókat, viszont titkosítatlan kapcsolaton könnyen elfogható a jelszó, de ha már titkosított a kapcsolat, nincs is értelme felhasználónevet kérni, hiszen a tanúsítvány tartalmazhatja azt a CN mezőben, amit webszerverek esetén a domain névre tartanak fest. Én felhasználónévnek használom.

Attól sem kell tartanom, hogy a tanúsítvány CN mezejét valaki megmásítja és más felhasználó nevében kapcsolódik a szerverhez, mivel a kiállított tanúsítvány rendelkezik egy aláírással, ami érvényességet veszti, ha utólag módosítják a fájlt, aláírni viszont csak a kiállító tud, feltéve ha csak ő birtokolja a titkos kulcsát. Jelszóvédett titkoskulcsok is kiállíthatóak, így ha valaki meg is szerzi a tanúsítvány-fájlokat, akkor sem képes használni őket a jelszó ismerete nélkül.

A tanúsítványok CN mezőjét az alábbi módon szerzem meg:

SecureHandlerUtil.java részlet

```
private static String getCommonName(Certificate cert)
    throws CertificateException, CertificateEncodingException {
    String certdata = getPrincipal(cert);
    int cnstart = certdata.indexOf("CN=") + 3; // "CN=" résztől ...
    int cnstop = certdata.indexOf(',', cnstart); // ... a vesszőig ...
    if (cnstop == -1) cnstop = certdata.length();
    // ... vagy ha nincs vessző, a végéig kértem a string tartalmát,
    // ami a tanúsítványban szereplő Common Name (CN)
    return certdata.substring(cnstart, cnstop);
}
```

A távoli gép tanúsítványa a Socket munkamenetéből kérhető le:

`socket.getSession().getPeerCertificates()[0]`

Alapértelmezés szerint titkosított kapcsolat nem hozható létre olyan tanúsítvánnyal, mely kiállítója nem szerepel a megbízható tanúsítványok között. Ugyan ezen okból keletkezik *SSLHandshakeException* akkor, amikor önalárt tanúsítványt használó webszerverhez kapcsolódunk. Ezt a kivételt könnyen el lehet kerülni, ha implementáljuk a *TrustHandler* interfészét, de valójában nálam ez a probléma fel se merült, csak egy korábbi projektemben.

Hogy a crt és key tanúsítvány-fájlokat egyszerűen használhassam, igénybe vettem az Apache *not-yet-commons-ssl^h* nevű API-ját, mely nem csak arra jó, hogy az adott tanúsítványt használja az alkalmazás az *SSLSocket*, illetve az *SSLSocketServer* használatakor, hanem a kiállító publikus kulcsát megadva az alapértelmezett eljárást felülbírálva ellenőrzi a tanúsítványt. Kivétel csak akkor keletkezik, ha a használt tanúsítványt nem a megadott kiállító állította ki.

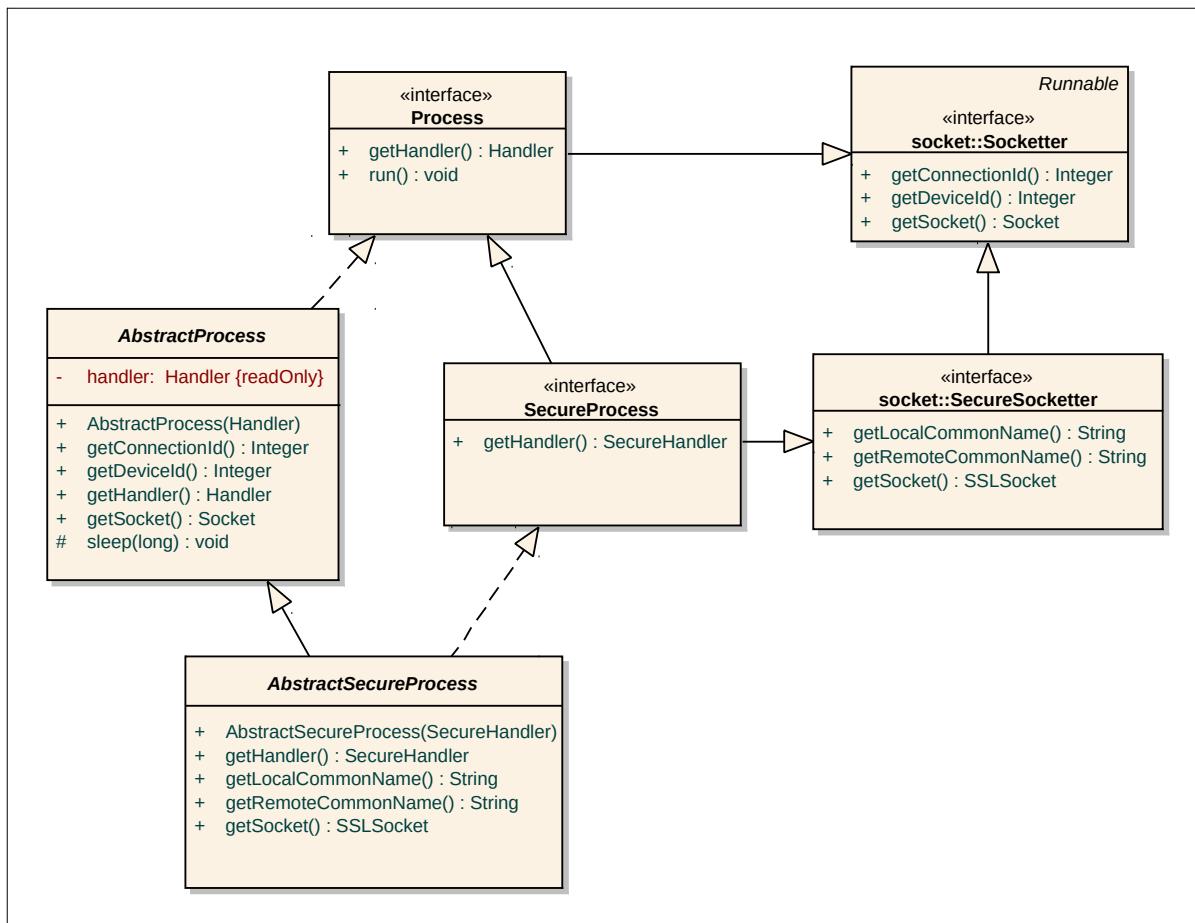
3.3.1.4. Az általános terv

Ahhoz, hogy a szerver képes legyen egy időben több kapcsolat kezelésére, egy előtesztelő ciklusban kell folyamatosan az *accept()* metódust futtatni, és amint visszatér egy *Socket* objektummal, a feldolgozást új szálban kell tovább folytatni, hogy a további kapcsolódó klienseket is fogadni tudja miközben tart a feldolgozás.

Mindent összevetve – eszközazonosító, kapcsolatazonosító, SSL és szálkezelés alkalmazásával – megalkottam a saját rendszeremet, mely alapja az általam írt *Handler* és *Process* interfész. A *Process* név használata nem bizonyult túl szerencsésnek, mert a JDK-ban van már egy ilyen nevű osztály, ráadásul importálni se kell, így kissé megtévesztő amikor hibás a kód, pedig tudja az ember, hogy nem kéne annak lennie, végül rájön, hogy nem ugyan arról az osztályról van szó. Jobb nevet viszont nem tudtam kitalálni és egy ponton túl már a *SecureProcess* és *SecureHandler* interfések vannak használva, így csak rövid ideig volt zavaró.

A Handler osztály implementálja a *Runnable* interfészét, így új szálban is futtatható a benne megírt *run()* metódus. Egyetlen feladata az eszköz- és kapcsolatazonosító kiértékelésével a megfelelő *Process* objektum létrehozása és futtatása.

A *Process* is implementálja a *Runnable* interfészét, de csak azért, mert ő is rendelkezik egy *run()* metódussal, amit a *Handler* hív meg. Az egyszerűbb felhasználás érdekében van pár getter metódus, melyet a *Handler* értékel ki, de elérhető a *Process* osztályban is. A *Process* osztály egy kapcsolatfeldolgozó. Ezzel lehet megvalósítani az eltérő viselkedést változó kapcsolatazonosító esetén és a szerepkört is meghatározza az eszközazonosító.



34. ábra: Process osztálydiagram

A `Handler` és a `Process` közös metódusai a `Socketter` interfészben találhatóak. Ő tartalmazza az azonosítók getter metódusait és a socket is elérhető tőle, hogy a kommunikáció megvalósulhasson. Titkosított kapcsolat esetén már azt is tudni lehet, hogy ki a helyi és a távoli felhasználó, így a `SecureSocketter` – melynek őse a `Socketter` – lehetővé teszi ezen adatok lekérését is és a `Socket` típusát is pontosítja `SSLocket` típusúra. Az `AbstractProcess` és az `AbstractSecureProcess` implementálja a getter metódusokat a `handler` objektum adataira támaszkodva, az egyetlen kifejtendő metódusuk a `run()` metódus.

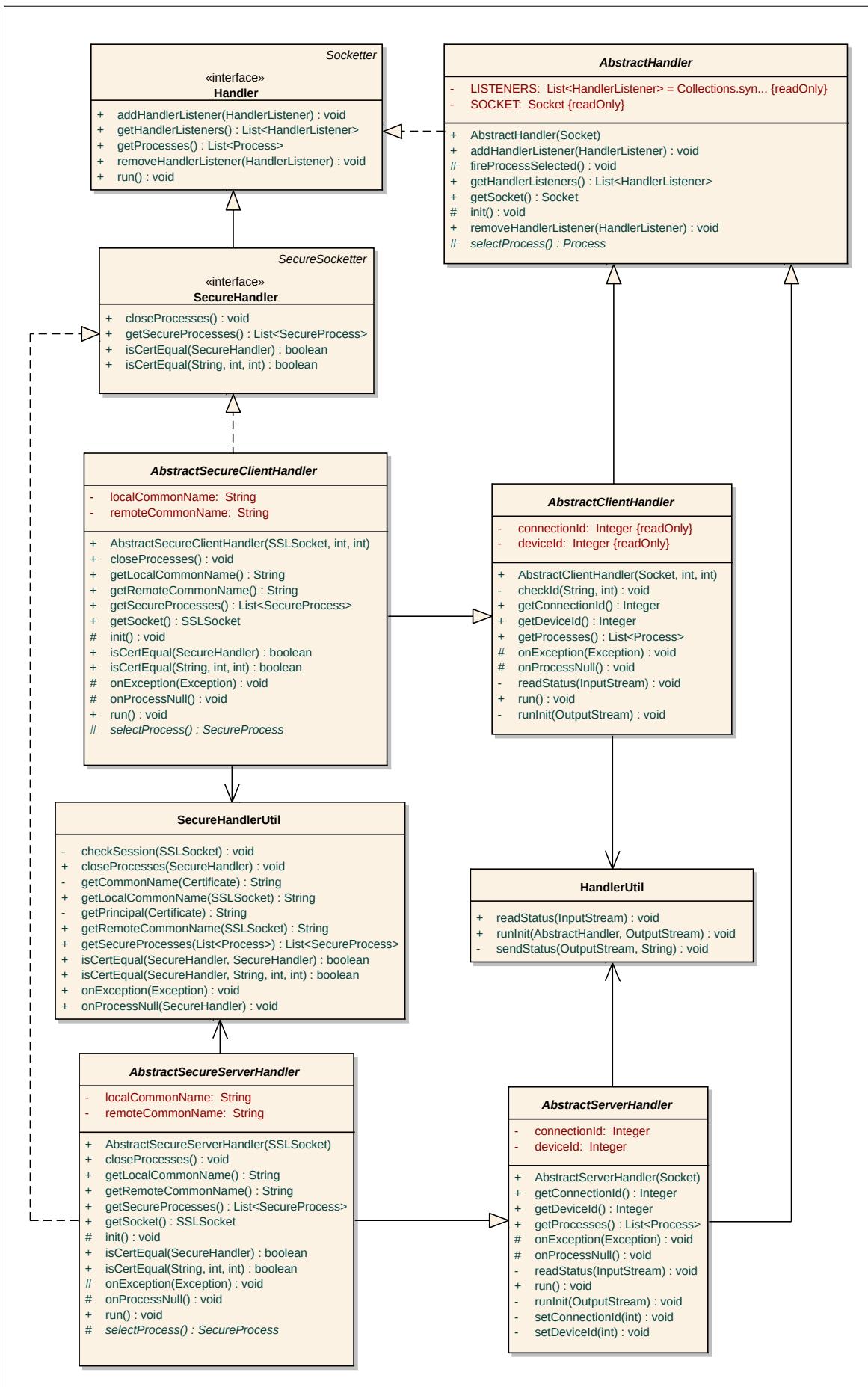
A `Handler` működésének megvalósítása valamivel összetettebb folyamat volt. Ennek az az oka, hogy a kliens és a szerver oldal eltérő módon viselkednek, mivel a kliens küldi a kapcsolatazonosítókat és a szerver fogadja őket. Ez után már minden fél ugyan úgy működik: létrehozza a megfelelő `Process` objektumot, majd meghívja a `run()` metódust.

A Handler osztálydiagram a következő oldalon található a nagy mérete miatt.

Az `Handler` interfések hierarchiája megegyezik a `Process` interfések hierarchiájával, de az eltérő viselkedés miatt itt két absztrakt osztály lett írva a két oldalhoz. Az `AbstractClientHandler` konstruktora paraméterben várja a két azonosító értékét, de az `AbstractServerHandler` nem vár ilyen értéket, mivel ő a kliens oldaltól kapja meg ezen adatokat.

Az `AbstractSecureClientHandler` és az `AbstractSecureServerHandler` osztályok esetén többszörös öröklésre van szükség. Egyrészt öröklődniük kell a megfelelő absztrakt `Handler` osztályból, másrészt saját közös metódusaik is vannak.

A Java szándékosa nem támogatja a többszörös öröklést, mert a tervezői szerint több bajjal jár, mint, amit megold. Tisztább megoldás helyette egy interfész és egy közbülső osztály használata, ahogyan az ábra is mutatja a következő oldalon.



35. ábra: Handler osztálydiagram

3.3.1.5. Az általános terv felhasználása

3.3.1.5.1. A kapcsolat ellenőrzése

A jármű biztonságos vezérlése érdekében a szerverrel kiépített kapcsolatot rendszeres időközönként ellenőrizni kell. A járműkliens mindenkor az utoljára kiadott parancsot hajtja végre mindaddig míg nem érkezik másik parancs. Ha a kapcsolat megszakad miközben az utolsó parancs az, hogy egyenesen előre menjen a jármű, ellenőrizetlen kapcsolat esetén a jármű megállíthatatlan lenne és biztosan ütközne valamivel.

A kapcsolat instabilitását csak akkor lehet detektálni, ha az egyik oldal ír a kimenő folyamba, a másik oldal meg olvassa a bejövő folyamot. Azon az oldalon, melyen az olvasás történik, instabil vagy megszakadt kapcsolat esetén időtúllépés detektálható. Ha minden oldal kölcsönösen felváltva ír és olvas, akkor minden oldalon detektálható az időtúllépés.

Ennek megfelelően létrehoztam két *DisconnectProcess* osztályt kliens és szerver oldalra. Működésük annyiban tér el, hogy mindenkor ír, a másik olvas és mindenkor addig csinálják, míg a kapcsolatot be nem zárják vagy meg nem szakad. Az én esetemben viszont nem elég a kapcsolat megszakadásának a detektálása, azt is észre kell venni, ha pillanatnyi instabilitás van a hálózatban, ezért két lépcsős időtúllépést vezettem be.

A `read()` metódus időtúllépését 1 másodpercre állítottam be, ez felel meg a pillanatnyi instabilitásnak. Időtúllépéskor elindul egy időzítő, ami 10 másodperctől számol vissza és ha ezen időn belül sem érkezik válasz, akkor a program úgy értelmezi, hogy a kapcsolat megszakadt. Ha 10 másodpercen belül válasz érkezik, az időzítő leáll, a hálózat újra stabilnak tekinthető és a kapcsolat ellenőrzése tovább folytatódik.

Az első időtúllépés esetén a jármű utolsó parancsa eltárolódik, majd a jármű megállításra kerül. A kapcsolat helyreálltával az utolsó parancs lép érvénybe és a jármű újra elindul, ha előtte is ment. Az időtúllépés közben küldött esetleges parancsokat a jármű sorban megkapja, így ha az időtúllépés alatt leállítás lett kérve, a jármű megáll.

3.3.1.5.2. Üzenetküldés és fogadás

A Java rendelkezik egy *Serializable* nevű interfésszel. Azon objektumok, melyek ezt implementálják, serializálhatóvá válnak feltéve, ha tulajdonságaik is serializálhatóak. Objektum fájlba való sserializálásakor az objektum összes tulajdonsága tárolásra kerül. A program leállta után, újból induláskor a fájl tartalmának deserializálásával az objektum visszatölthető a memoriába.

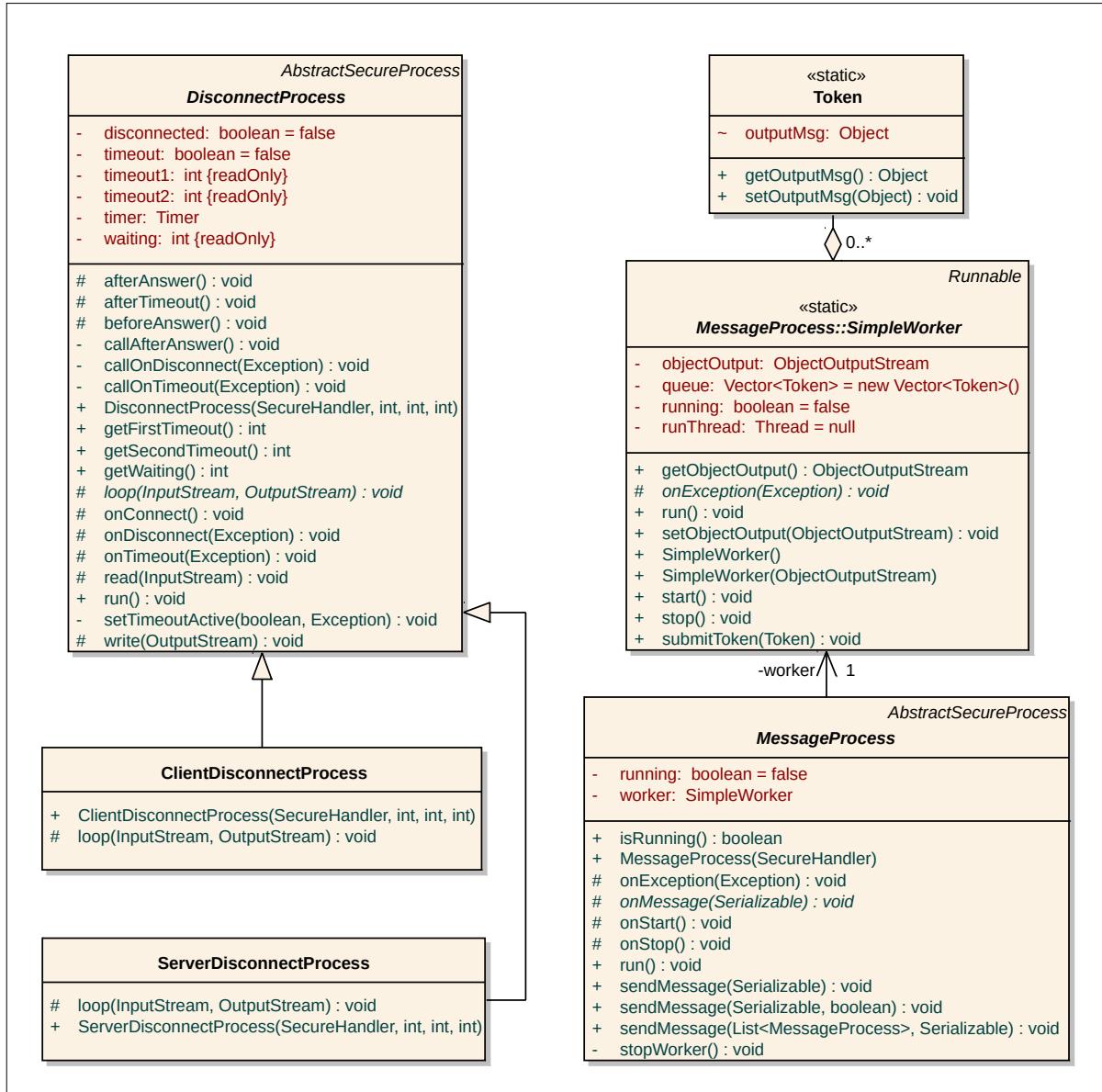
Fájlba való sserializáláshoz szükség van egy *FileOutputStream* objektumra, ami fel van ruházva egy konkrét fájlba való írás képességevel és egy *ObjectOutputStream* nevű objektumra, mely a *FileOutputStream* objektumot felhasználva képes sserializálni. Deserializáláskor ugyan ez a folyamat zajlik le, csak írás helyett olvasás, tehát *FileInputStream* és *ObjectInputStream* kerül felhasználásra.

A Java *ObjectOutputStream* és *ObjectInputStream* osztályát arra használtam fel, hogy üzenetobjektumokat küldjek a socketkapcsolaton keresztül. Az elv ugyan az, mint a fájlba való sserializálás, csak *FileOutputStream* helyett a socket *OutputStream* példányát adom át az *ObjectOutputStream* példányosításakor. A socket másik oldalán megtörténik a deserializálás, ezzel egy konkrét objektum átkerült egyik alkalmazásból a másikba.

Ezen elvnek megfelelően írtam meg a *MessageProcess* nevű osztályt, mely létrejötte után elindít egy szálat, melyben szálkezelt üzenetküldés megy végbe, és a szál indítása után ciklusban történik meg az objektumok fogadása.

Az üzenetfogadás nem szálkezelt, mivel egyszerre csak egy objektum érkezhet, de a küldéskor szükség van szálkezelésre, hogy két objektum egyszerre ne íródhasson ki a folyamba, hiszen ha két külön objektum összekeveredne, képtelenség lenne értelmes adatot nyerni a folyamból.

A szálkezelt üzenetküldésre találtam egy jó megoldástⁱ az interneten, melyet fel is használtam. Az elve viszonylag egyszerű. A küldendő üzenetet Tokenbe kell helyezni, majd megkérni a szálkezelt feldolgozót, hogy küldje el. A feldolgozó a tokeneket egy vektorba teszi és egymás után elküldi őket, amint sorra kerülnek.



36. ábra: DisconnectProcess és MessageProcess osztálydiagram

3.3.1.5.3. Üzenetobjektumok: adatok, részadatok

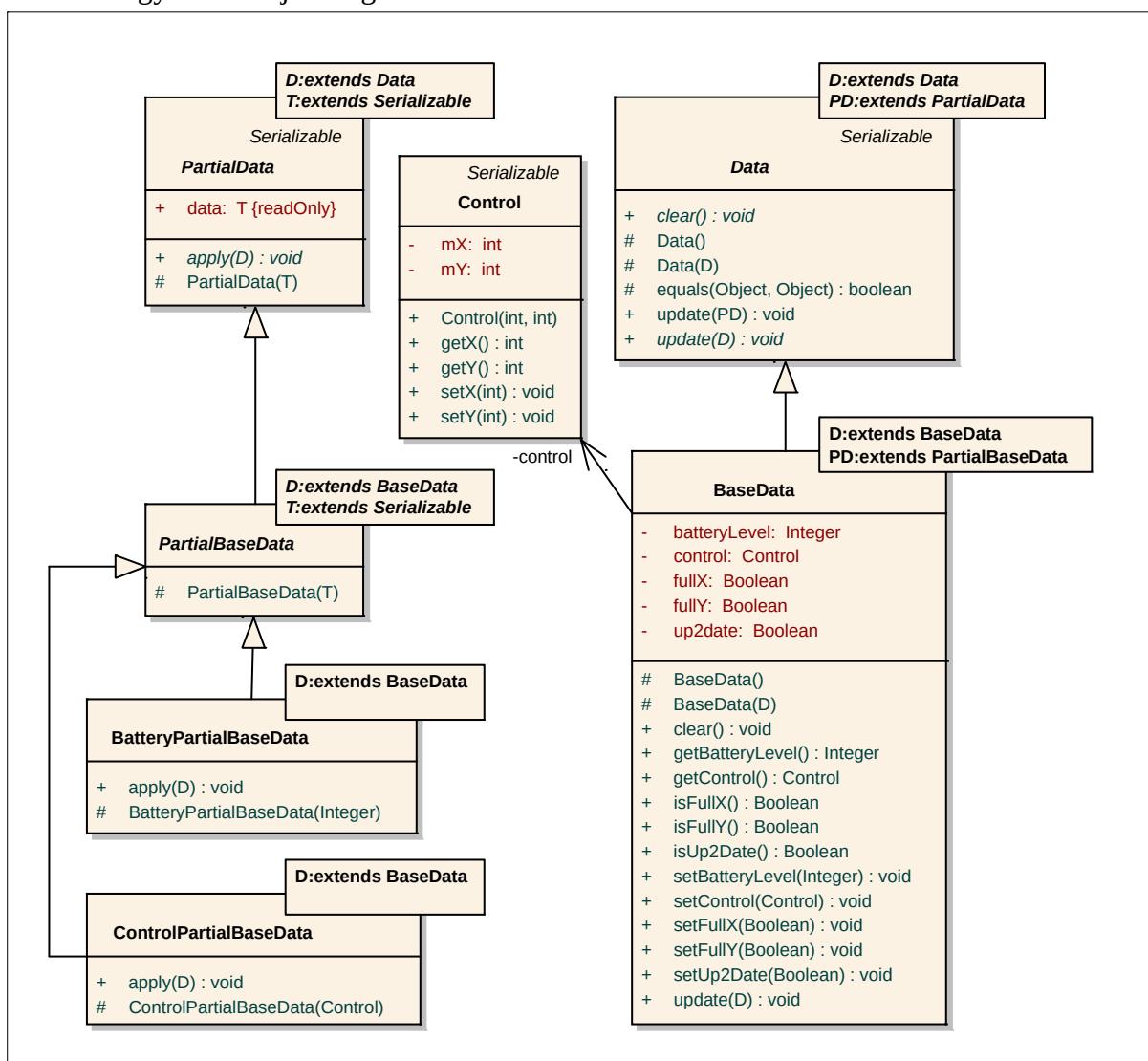
Ebben a fejezetben már szó esett arról, hogy az üzenetküldést hogyan valósítottam meg, de arról még nem beszéltem, mi is az, amit küldök ezen a csatornán keresztül.

Az üzeneteket két részre osztottam fel: az egyik részbe tartoznak az üzenetek, melyek a vezérlőkliens és a hídszerver között zajlanak, a másik részbe a járműkliens és a hídszerver között zajló üzenetek tartoznak.

A szétválasztásra elsősorban azért volt szükség, mert deserializálni csak azokat az objektumokat lehet, melyeket az alkalmazás ismer, tehát az osztálybetöltőnek el kell tudnia érni ezen osztályok bájtkódját. Már részről teljesen különböző információcsere zajlik egy járműkliens és egy vezérlőkliens esetén. A vezérlőkliens például küldhet és kaphat csetüzenetet, járművezérlő kérelmet, és a jármű nyers adatai helyett is csak a számára hasznos, átalakított üzenetet kap. A járműkliens viszont csak nyers szenzoradatokat közvetít és a jármű vezérlőjelét fogadja.

Valójában az üzenetobjektumok soha sem utasítások, mindig adatmódosulás jelzés történik, és erre reagálnak az alkalmazások. Például, ha a vezérlőjel megváltozik, a járműkliens módosítja az IOIO digitális kimeneteit és így a jármű viselkedése megváltozik. Felmerül a kérdés, hogy honnan tudják a kliensek, hogy az adott üzenet milyen adatmódosulást jelent és hogy a módosulást hogyan dolgozzák fel.

Mivel valójában adatküldés zajlik, ezen adattároló osztályokat nem üzenetnek, hanem adatnak és részadatnak neveztem el. minden *Data* osztályhoz több *PartialData* osztály tartozhat. Az adatosztály minden jelenlegi állapotát tárolja annak minden jellemzőjével. A részadatosztály viszont csak egyetlen tulajdonság módosulását tartalmazza.



37. ábra: *Data* és *PartialData* osztálydiagram

Amikor egy kliens kapcsolódik a hídszerverhez, megkapja/elküldi a neki megfelelő adatosztályt, így minden kezdőadat a rendelkezésre áll és a teljes felület lefrissül. Ettől a ponttól kezdve már csak részadatot kap, hiszen a többi adat elküldése fölösleges, mert nem változik meg.

A Data és a PartialData osztály két típusos paraméterrel rendelkezik. A Data első paramétere D, mely meghatározza milyen adatról van szó. Ez a típusparaméter megegyezik az osztály típusával. Célja, hogy korlátozza az update(D) metódus használhatóságát.

Hasonló elven lett megírva többek között a Boolean osztály is a Javaban:

public final class Boolean implements Comparable<Boolean>

Boolean esetén ezáltal a compareTo metódus szignatúrája: **public int compareTo(Boolean b)**

Ez azt eredményezi, hogy egy Boolean csakis Boolean típusú objektummal hasonlítható össze.

A Data második paramétere PD, mely a részadat típusát határolja be. Ez a paraméter már nem egy véleges, utódok nélküli osztály típusát fogja tartalmazni az utódokban, hanem egy általánosabb típust. Ilyen például a *PartialBaseData* osztály, mely a *BaseData* osztály részadata.

A PartialData is két típusparaméterrel rendelkezik. A D paraméter megadja azt, hogy a részadat pontosan milyen típusú Data osztályhoz tartozik és a T paraméter a megváltozott adat típusát adja meg. Itt is igaz, hogy a metódus használata konkrét adattípusra be van határolva, de itt első sorban a generikusság előnyét használtam ki, hogy ne kelljen kasztolnom az utód osztályokban.

Ugyanis a PartialData update(D) metódusa az, ami végrehajtja az adatmódosítást a Data objektumban, melyet paraméterben kell neki átadni. Ezzel a trükkkel nem kell if-else-if szerkezetet használni minden üzenetre odafigyelve, hanem maga az elküldött üzenetobjektum meghatározza azt is, hogyan kell kezelni az adat módosulását.

Az osztálydiagramon látható két részadat. A BatteryPartialBaseData egy Integer számot tárol, mely az akkumulátor-szint új értéke százalékban megadva és az apply(BaseData) metódusa meghívja a BaseData objektum setBatteryLevel(Integer) metódusát átadva az új értéket. A ControlPartialBaseData a jármű vezérlőjelét tartalmazza és a setControl(Control) metódust futtatja.

A setter metódusokat az utódban felül lehet definiálni, így be lehet tenni olyan utasításokat, melyeknek az adat módosulására le kell futniuk, így frissíthető például a vezérlőkliensen a vezérlőpanel vagy az akkumulátor-szint.

Ez elsőre kicsit úgy tűnik, szembemegy az eddig megszokott eseménykezeléses eljárással, de megvan az az előnye, hogy nagyon egyszerűen detektálható mindenféle adatmódosulás anélkül, hogy Listener interfészeket kelljen létrehozni mindenféle adattípushoz, ami jelentősen növelné a kód méretét fölöslegesen. Helyette mindenhol alkalmazás leörökli a BaseData osztályt és a setter metódusokat kibővíti úgy, hogy fel tudja dolgozni az adatok módosulását.

3.3.1.6. A kamrakép közvetítése

A kamerakép közvetítésére nem lehetett általános tervezet készíteni, mivel mindenhol alkalmazásnak más feladata van az MJPEG folyammal. Ezért mindenhol alkalmazás saját *VideoProcess* osztállyal rendelkezik.

A telefonon futó járműkliensen miután létrejött a híddal való kapcsolat és példányosítódott az osztály, kapcsolódni próbál a telefonon futó IP Webcam alkalmazás szerverével. Ha nem érhető el, elindítja a szervert, majd megvárja, hogy elérhető legyen. Miután kapcsolódott hozzá, egyszerűen elkeri a bejövő folyamot, valamint a hídszerverhez vezető kimenő folyamot és ciklusban egy kilóbájtos egységekben továbbítja az IP Webcam által generált MJPEG folyamot a hídszerver felé.

A hídszerver járműkliens felőli feldolgozója viszont egyszerűen elkezdi olvasni a bejövő folyamot, dekódolja az MJPEG folyam képkockáit, majd az utolsó képkockát eltárolja. A hídszerver vezérlőkliens felőli feldolgozója hozzáfér a képkockákat tároló objektumhoz és a kapcsolat létrejöttekor az MJPEG szabványnak megfelelően generál egy fejlécet, majd elküldi. Ha van a választott járműhöz eltárolva képkocka, elküldi. Ez után vár a képkocka megváltozására és változás esetén kiküldi az új képkockát. minden járműhöz külön tárolódik el az aktuális képkocka.

A vezérlőkliens kapcsolatfeldolgozója ugyan úgy dekódolja az MJPEG folyamot, ahogyan a hídszerver, viszont ő a képkockákat már nem tárolja, hanem átadja a főablak JLabel komponensének, ami megjeleníti az aktuális képkockát.

Ezzel a módszerrel tulajdonképpen hasonló funkció lett megvalósítva, mint a proxyk esetében azzal a különbséggel, hogy egy adatfolyamot több felé szór a hídszerver.

3.3.1.7. Összegzés

Ebben a részben bemutattam, a hálózati kommunikáció működési elvét. Láthattuk, hogy hogyan valósul meg a kialakított kapcsolat ellenőrzése, hogy hogyan küldenek és fogadnak üzenetet, és hogy ezek az üzenetek mik is valójában. Az utolsó fejezetben felvázoltam a kamerakép közvetítésének a módját is a különböző alkalmazások esetében.

Ezzel a teljes hálózati részbe sikerült betekintést nyerni. A kapcsolatot tehát a kliensek kezdeményezik a szerver felé és minden alkalmazás három kapcsolatot alakít ki a szerverrel. Elsőnek létrejön a kapcsolat ellenőrző, ezt követi az üzenetküldő és fogadó, végül az MJPEG folyam közvetítésére vagy fogadására jön létre a harmadik kapcsolat.

Hangot már nem közvetíték a telefonról, mert már így is a szokásosnál nagyobb terhelést kell neki elviselni az MJPEG folyam megalkotásakor, de a fenti elvet követve megvalósítható lenne egy negyedik kapcsolat, melyben a hang lenne közvetítve valamely kodek segítségével.

3.3.2. A vezérlőkliens fontosabb elemei

A vezérlőkliens programozását a grafikus felülettel kezdtem. Ezt követte a hálózati kommunikáció megvalósítására írt osztályaim felhasználása. Lényegét tekintve ezen két lépés után az alkalmazás elnyerte végső formáját, már csak apróbb simítások kellettek, hogy minden operációs-rendszeren a lehető legstabilabban fusson a program.

3.3.2.1. A grafikus felület felépítése

A grafikus felület főként Swing osztályokon alapszik, de natív SWT komponenseket is felhasználtam, melyekről még részletesebb leírást adok ebben a fejezetben.

3.3.2.1.1. Look And Feel (LAF)

A Swing komponensek nagy előnye az AWT komponensekkel szemben, hogy kinézetük központilag módosítható, így megoldható az alkalmazás kinézetének igazítása az operációs-rendszer felületéhez. A Javában erre a célra lettek megalkotva a Look And Feel (LAF) osztályok. A különböző operációs-rendszerekhez kiadott Java virtuális gépek tartalmazzák az adott rendszerre írt LAF osztályt, melyet még a Swing komponensek használata előtt alkalmazva a natív komponensekhez hasonlító kinézet állítódik be a Swing komponenseken.

Létrehoztam egy util osztályt, mely tartalmazza a grafikus felületeknél gyakran használatos eljárásokat. Ez az osztály tartalmazza többek között a rendszerhez igazított Look And Feel osztály beállítását is oly módon, hogy Linux, Windows és OS X rendszereken is megfelelően működjön.

A LAF beállító metódus első körben a Linux rendszereken használatos GTK Look And Feel osztályt próbálja alkalmazni, mert nem mindegyik Linux rendszeren tér vissza a getSystemLookAndFeelClassName() metódus a GTK LAF osztály nevével. Viszont, ha az operációs-rendszer nem Linux, akkor kihagyja a GTK LAF beállítását elkerülve az OS X rendszereken való alkalmazását. Nem Linux rendszereken a System Look And Feel megfelelően alkalmazható, így utolsó lépésként a rendszerhez tartozó LAF beállítása történik meg, kivétel keletkezése esetén nem változik az eredeti Look And Feel.

UIUtil.java részlet

```
/*
 * Az adott rendszer alapértelmezett kinézetét alkalmazza feltéve, ha
 * elérhető a grafikus felület.
 */
public static void setSystemLookAndFeel() {
    // csak akkor fut le, ha van grafikus felület támogatás
    if (!GraphicsEnvironment.isHeadless()) {
        try {
            // Linuxra GTK LAF beállítása
            // az OS ellenőrzés szükséges, mert Mac-en is van GTK LAF
            if (System.getProperty("os.name").toLowerCase().contains("nux")) {
                UIManager.setLookAndFeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel");
            }
            else {
                // nem szép dolog, de így legalább nem kell a catch blokkot külön
                // metódusra tenni
                throw new Exception();
            }
        }
        catch (Exception ex) {
            // Ha nem Linuxon fut a program, rendszer LAF beállítása
            try {
                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            }
            catch (Exception e) {
                ;
            }
        }
    }
}
```

3.3.2.1.2. Standard Widget Toolkit (SWT)

Az SWT^j egy Java nyelvre írt – Java Natív Interfészre (JNI) támaszkodó – komponensgyűjtemény, mely segítségével az operációs-rendszer által biztosított natív felületi komponenseket használva lehet grafikus felhasználói felületeket létrehozni. Eredetileg az IBM fejlesztette ki, de jelenleg az Eclipse Foundation végzi rajta a fejlesztést és a karbantartást, mivel az első SWT-t használó alkalmazás az Eclipse IDE volt.

Az SWT nem része a szabvány Java API-nak, csupán egy alternatíva az AWT és a Swing helyett. Előnye, hogy gyorsabb a Swing alkalmazásoknál és nincs szükség Look And Feel alkalmazására., mivel natív komponenseken alapszik. Hátránya, hogy sok helyet foglal, az alkalmazások hordozható, platformfüggetlen kiadásaiban, mert minden operációs-rendszerhez külön jar fájl lett kiadva, és minden rendszeren az annak megfelelő jar fájlt kell használni.

3.3.2.1.3. A térkép kivitelezése, Google Map

A vezérlőkliens térképet megjelenítő dialógusablaka a Google Map szolgáltatásán alapul. A Google írt a térképszolgáltatásához egy API-t, ami elérhető JavaScript nyelven a webes alkalmazásokban, Androidra írt alkalmazásokban és iOS rendszeren is.

Annak ellenére, hogy Androidra van támogatása, az asztali számítógépeken futó Java virtuális gépekre nem tettek közzé alkalmazásprogramozási interfést. Találtam egy fórumot, amin már ez az igény felvetődött, de a Google fejlesztői azt tanácsolták, hogy használjuk a JavaScriptre kiadott verziót, tehát weboldalt jelenítsünk meg.

Java nyelvre több webböngésző támogatás is létezik, de a legfejlettebb ezek közül az SWT által biztosított natív támogatás. Az SWT nem emulálja a webböngészőt, hanem ténylegesen a rendszerre telepített alapértelmezett böngészőt használja fel a weboldalak megjelenítésére, ezért a JavaScript alapú kód is tökéletesen futtatható segítségével.

A fő oka az SWT használatának tehát az, hogy meg tudjam jeleníteni a Google által biztosított térképet.

3.3.2.1.3.1. Swing versus SWT

Az SWT készítői felhívják a figyelmet arra, hogy azokban az alkalmazásokban, melyekben az SWT használva van, Swing és AWT osztályok nem használhatóak felület megjelenítésére, mert együttes használatuk az egész alkalmazás összeomlását is eredményezheti és magára a grafikus felületre is kedvezőtlen hatással lehet. Debian alatt tesztelve a GNOME valóban kifagyott az SWT leállítása után és a Java alkalmazás se volt hajlandó leállni. Amint konzolból kilőttem az alkalmazást, a GNOME újra megfelelően működött.

Megoldás lehetett volna a Swing és AWT elhagyása, teljes SWT alapú felület írásával, de inkább a Swing felületnél maradtam, mert nem igényel külön jarfájt és kerestem erre a problémára egy megoldást. A tervem az volt, hogy mellékelem az SWT fájlokat és így elérhető marad a térkép szolgáltatása, de SWT nélkül továbbra is működni fog a teljes alkalmazás a térkép szolgáltatás inaktiválásával.

3.3.2.1.3.2. Native Swing és bővítése

Christopher Deckers a DJ projekt keretében elkészítette a Native Swing^k nevű programkönyvtárt. Célja a Swingben nem támogatott, de az SWT-ben elérhető funkciók integrálása Swing komponensekhez. Így Swing alapú alkalmazásokban is elérhető a webböngésző támogatás, de a VLC médialejátszó vagy a Flash lejátszó is támogatva van.

Christopher azt a trükköt alkalmazta a projektben, hogy létrehozott egy natív interfész, mely inicializálásakor egy új Java process indul el a háttérben, ami az SWT komponenseket kezeli. Így valójában az alkalmazás, mely a natív interfész használja, valójában nem rendelkezik SWT komponensekkel és az új folyamatban indított Java alkalmazás is csak SWT komponensekkel rendelkezik, tehát minden program biztonságosan üzemel.

Ezt az információt nem az internetről szereztem, mivel a mögöttes működésről a szerző nem tett említést. Onnan tudom, hogy a programkönyvtárat tovább bővítettem és ehhez meg kellett értenem a működésének az alapját.

Az általam írt bővítés lehetővé teszi az SWT alapú értesítési ikonok használatát, tehát a régmídi AWT alapú TrayIcon osztályt váltja fel. Az SWT alapú TrayItem lehetővé teszi a képek használatát a lenyíló menü elemeihez, Linuxon átlátszó ikon jelenik meg és maga a lenyíló menü is jobban illeszkedik a rendszer felületéhez, mivel a LAF nem hat az AWT komponensekre.

Ezen kívül létrehoztam egy adapter osztályt, mely az általam Native Swingre írt JTray osztályt használja, ha az SWT elérhető és az AWT SystemTray osztályát akkor, ha nem érhető el az SWT. Így az értesítési ikonok akkor is megjelennek, ha az SWT nem érhető el.

3.3.2.1.3.3. Az SWT betöltése, SWTJar

Ahhoz, hogy a térkép és a natív értesítési ikon elérhető legyen, az osztálybetöltőnek meg kell adni az SWT operációs-rendszerhez illő elérhetőségét, csak ez után használható a Native Swing projekt.

Alapesetben erre a bevett eljárás a classpath változóban való felsorolása azoknak az osztályoknak, melyeket az alkalmazás használ, de mivel az SWT minden rendszerhez és ezen belül is architektúrához külön lett kiadva, több jar fájlt kéne kiadni, ami nehezítené a hordozhatóságot.

A megoldást az SWTJar¹ használata jelenti. Segítségével az SWT alapú alkalmazások könnyen csomagolhatóak egyetlen jar fájlba. A végeredményben kapott fájl tartalmazza az SWT Windowsra, Linuxra és OS X rendszerre kiadott változatát minden architektúrához – tehát 6 jar fájlt –, valamint tartalmazza az összes osztályt az ant taskban megadott jar fájlok ból. A jar fájl mindenkor rendszeren a megfelelő SWT könyvtárat tölti be jar-in-jar osztálybetöltő segítségével, így csupán egyetlen jar fájl szükséges az alkalmazás indításához.

Az ant task tartalma a következő:

```
<project name="RemoteControlCar" basedir=".">
<description>Package cross platform SWT Jar</description>
<taskdef name="swtjar" classname="org.swtjar.ant.SWTJarTask"
    classpath=".:/swtbuild/swtjar.jar"/>
<swtjar jarfile=".:/dist/ui.jar"
    targetmainclass="org.dyndns.fzoli.rccar.Main"
    swtversion="4.3M5a">
    <!-- Application Classes -->
    <fileset dir=".:/build/classes" includes="**/*.class" />
    <!-- Library Classes -->
    <zipfileset excludes="META-INF/*.MF" src="lib/common/log4j-1.2.17.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="lib/common/not-yet-commons-ssl-0.3.11.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="lib/controller/imgscalr-lib-4.2.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="desktop/BrowserTest/lib/DJNativeSwing.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="desktop/BrowserTest/lib/DJNativeSwing-SWT.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="desktop/BrowserTest/lib/jna.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="desktop/BrowserTest/lib/platform.jar"/>
    <zipfileset excludes="META-INF/*.MF" src="desktop/MacApplication/bin/MacApplication.jar"/>
    <!-- SWT Jars -->
    <fileset dir=".:/desktop/BrowserTest/lib/swt" includes="swt-*-4.3M5a.jar" />
</swtjar>
</project>
```

Az swtjar.jar fájl tartalmazza a jar-in-jar osztálybetöltőt és az SWTLoader nevű osztályt, mely kiválasztja a megfelelő SWT jar fájlt és betölti azt. Az ant task a dist könyvtárba egy ui.jar nevű fájlt állít elő, ami maga a csomagolt alkalmazás.

Az SWTJar példájára írtam egy metódust, ami hozzáadja a classpath változóhoz a megfelelő SWT jar fájlt és kivettem az összecsomagolt fájlból az SWT-t, mert túl nagy lett az így kapott fájl mérete. Helyette a lib könyvtárban kapott helyet, így a különböző rendszerre kiadott telepítők mérete jóval kisebb lett, mivel a más rendszerekhez tartozó SWT támogatás nem foglalta a helyet.

Az SWT betöltését csak akkor végzem el, ha az SWT osztály – mely általános változókat deklarál – nem érhető el.

Ha a `Class.forName("org.eclipse.swt.SWT", false, Main.class.getClassLoader());` utasítás nem dob `ClassNotFoundException` kivételt, nincs szükség az SWT betöltésére, mert már elérhető.

Kivétel keletkezése esetén szükség van a classpath kibővítésére, mely az URLClassLoader osztály `addURL(URL)` privát metódusával valósítható meg, de ehhez a privát metódust előbb elérhetővé kell tenni:

A privát láthatóságú URLClassLoader.addURL(URL) metódus hívása:

```
URLClassLoader sysloader =
    (URLClassLoader) ClassLoader.getSystemClassLoader();
Method method = URLClassLoader.class.getDeclaredMethod("addURL",
    new Class[] {URL.class});
method.setAccessible(true);
method.invoke(sysloader, new Object[] {swtJarFile.toURI().toURL()});
```

3.3.2.1.4. Töltőképernyő (Splash screen)

Az alkalmazás indulása 5-20 másodpercet is igénybe vehet, ezért készítettem egy töltőképernyőt, ami a Java indulása után egyből megjelenik. A hosszú betöltési időt főként a Native Swing projekt okozza, mert addig nem jelenik meg a kezdőképernyő, míg a natív interfész be nem töltődik.

A töltőképernyő^m gyors megjelenését maga a Java biztosítja.

A *SplashScreen-Image*: *org/dyndns/fzoli/rccar/resource/splash.gif* attribútumot beállítottam a jar fájl META-INF/MANIFEST.MF fájlban. Ezzel értem el, hogy még az alkalmazásom main metódusának hívása előtt megjelenik egy animált töltőképernyő.

A Java által megjelenített töltőképernyő futási időben kezelhető a SplashScreen osztályon keresztül. A SplashScreen.getSplashScreen() metódus null referenciával tér vissza, ha a megadott gif fájl nem létezik vagy nem lett megadva a SplashScreen-Image paraméter, tehát ha nincs töltőképernyő megjelenítve. Ha a töltőképernyő megjelent a SplashScreen.createGraphics() példánymetódussal készíthető egy grafikus objektumot, melyre rajzolva információ jeleníthető meg a töltőképernyőn.

Az így kapott Graphics2D objektumot használom fel arra, hogy szöveget jelenítsek meg:

```
public static void setSplashMessage(String s) {
    if (g != null && s != null && splash.isVisible()) {
        int y = 185; // a feliratok pozíciója vertikálisan a mozgó csík alatt
        g.setComposite(AlphaComposite.Clear); // rajzolás helyett radírozás
        g.fillRect(1, y - 10, splash.getSize().width - 2, 20); // felirat törlése
        g.setPaintMode(); // radírozás helyett újra rajzolás
        printString(s + (s.isEmpty() ? "" : "..."), y); // új felirat kirajzolása
                                                       // horizontálisan középre
        splash.update(); // a töltőképernyő frissítése
    }
}

private static void printString(String s, int y) {
    int len = (int) g.getFontMetrics().getStringBounds(s, g).getWidth();
    int start = splash.getSize().width / 2 - len / 2;
    g.drawString(s, start, y);
}
```

A töltőképernyő eltűnik, amint az első ablak megjelenik, de a close() metódus hívásával is eltűnik. Fontos megemlítenem, hogy amint a töltőképernyő eltűnik, nincs lehetőség az újbóli megjelenítésre.

3.3.2.1.5. A kamerakép megjelenítése, átméretezés (imgscalr)

A Motion JPEG stream valójában JPEG képkockák sorozata, melyeket egy előre meghatározott jel választ el egymástól. A folyamot olvasva így egyszerűen két határolójel köti rész felel meg egy képkockának. A határolójelek közti részt bájt tömbben tárolva, majd ezen tömb használatával egy *ByteArrayInputStream* objektumot létrehozva az *ImageIO.read(InputStream)* metódus használatával létrehozható egy *BufferedImage* objektum.

A megjelenítés előtt azonban a képet 640 x 480 méretűre át kell méretezni, hogy a felülethez illeszkedjen. Erre azért van szükség, mert a telefon 320 x 240 méretű képeket követít az adatforgalom és a processzor terhelésének minimalizálása érdekében.

A kép átméretezésére a Java Image Scaling Library-t használom, röviden imgscalr. Segítségével az *BufferedImage* objektumok könnyen átméretezhetőek. Nagy előnye, hogy gyors, testreszabható az átméretezés sebessége és minősége, valamint rendelkezik élsimítással is.

A *JLabel* nem csak szöveget, hanem képet is képes megjeleníteni, így az átméretezett kép a *JLabel.setImage(Image)* metódus használatával jelenik meg a felületen.

3.3.2.1.6. Az alkalmazás lokalizálása (ResourceBundle)

Mindhárom alkalmazás támogatja az angol és a magyar nyelvet. Ezen bekezdésben a hídszerver és a vezérlőkliens lokalizálásáról lesz szó, az Android rendszerre írt alkalmazások lokalizálása másképpen valósíthatók meg.

A Java nyelvben az elterjedt és javasolt módszer a lokalizációraⁿ a *ResourceBundle* osztály alkalmazása. minden nyelvhez külön properties kiterjesztésű fájlt kell létrehozni. Ezen fájlokat is a forráskódok közé kell betenni és csomagon belül is szerepelhetnek. A fájlok kulcs-érték párokat tartalmaznak. A nyelvi fájlokban a kulcsok segítségével lehet lekérni az adott nyelvhez tartozó szöveget. A *ResourceBundle.getBundle(String, Locale)* metódus első paramétere a fájl helyéből és nevéből származik, a második a kérty nyelvet definiálja. A metódus a megadott Locale alapján visszatér egy *ResourceBundle* objektummal, ami tartalmazza a helynek megfelelő nyelv kulcs-érték párokat. Ha az adott nyelvhez nem tartozik properties fájl, akkor az alapértelmezett fájl kerül használatba.

Az alkalmazásomhoz három különböző nyelvi fájlt hoztam létre. Mind a org.dyndns.fzoli.rccar.l10n nevű csomagban foglalnak helyet. A *chooser* properties fájlok tartalmazzák az alkalmazás választó ablakhoz használt szövegeket, a *bridge* a hídszerver lokalizációja és a *controller* a vezérlőkliensé.

Ennek megfelelően az alkalmazás választó a következő utasítást használja a nyelvi fájl betöltéséhez: *ResourceBundle.getBundle("org.dyndns.fzoli.rccar.l10n.chooser", Locale.getDefault())*

A *Locale.getDefault()* az operációs-rendszerben megadott helybeli tér vissza, így Magyarországon az alkalmazás választó nyelve magyar lesz, másutt viszont angol. A vezérlőkliens és hídszerver esetén az alapértelmezés szintén a tartózkodási hely nyelve, de a konfigurációban ez bármikor átállítható.

A lokalizát szövegek a *ResourceBundle.getString(String)* metódussal kérhetők le a paraméterben a kulcsot átadva. Példa a properties fájlok nevére: *bridge_en.properties*, *bridge_hu.properties*.

A properties fájlokban a kulcs-érték párokat az első szóköz vagy az egyenlőségjel választja el egymástól. Példa: *please_wait = Kérem, várjon*

Hogy ne kelljen az alapértelmezett nyelvi fájlt létrehozni, helyébe az angol nyelvet állítottam:

```
final ResourceBundle lng = ResourceBundle.getBundle(baseName, locale);
final ResourceBundle def = locale == Locale.ENGLISH ?
    null : ResourceBundle.getBundle(baseName, Locale.ENGLISH);
final ResourceBundle res = def == null ? lng : new ResourceBundle() {

    @Override
    protected Object handleGetObject(String key) {
        try {
            return lng.getObject(key);
        }
        catch (Exception ex) {
            try {
                return def.getObject(key);
            }
            catch (Exception e) {
                return null;
            }
        }
    }
};

return res;
```

A *ResourceBundle.handleGetObject(String)* metódus felüldefiniálásával elértem, hogy az angol nyelvi fájl legyen használva akkor, ha a választott nyelvhez nem lett definiálva egy kulcs.

3.3.2.2. A konfiguráció tárolása

A vezérlőkliens konfigurációja ellentétben a hídszerverrel nem szöveges fájl. Ennek fő oka, hogy a vezérlőkliens rendelkezik grafikus felülettel, amin egyszerűen be lehet állítani minden paramétert.

A konfiguráció tárolására írtam egy Config nevű osztályt a JavaBeans^º tervezési mintát követve. A bab osztályok fő feladata az adatok tárolása. Eredetileg GUI komponensek adatreprezentálására lett kifejlesztve.

Új konfigurációt a Config osztály példányosításával lehet létrehozni. A hídszerverhez való kapcsolódáshoz szükséges adatokat ez az objektum tárolja. Amikor a felhasználó elmenti az adatokat, ezen objektum szerializálásra kerül egy fájlba. Ha ez a fájl létezik és érvényes objektumot tartalmaz, akkor a program indításakor betölöttök a memoriába és nem jön létre új konfiguráció.

A Config objektumot tároló controller.ser nevű fájl felhasználónként külön kerül tárolásra, hogy a felhasználók egymás beállításait ne írják felül. minden operációs-rendszeren van az alkalmazásoknak felhasználónként fenntartva egy könyvtár, ahol a konfiguráció menthető. Linux rendszeren ez a felhasználó könyvtárában a .config nevű könyvtár. Linuxon ez alapján a valaki nevű felhasználó konfigurációs állománya a /home/valaki/.config/Mobile-RC/controller.ser helyen található. Unix alapú rendszereken a felhasználó könyvtára a ~ jelkel is helyettesíthető. OS X alatt a ~/Library/Application Support/Mobile-RC könyvtár van használva a végleges és ideiglenes adatok tárolására.

Windows rendszereken nem mondható meg egyértelműen, melyik az a könyvtár, ahol ezen adatokat célszerű menteni, mert verziónként eltérő hely van definiálva. A Windowsban viszont van egy SHGetFolderPath(HWND, int, HANDLE, DWORD, LPTSTR) szintaktikájú C++ metódus, ami a shell32.dll nevű fájlból található meg. Ezzel a metódussal megtudható, melyik az a könyvtár, mely az adott rendszeren az alkalmazásoknak van fenntartva. A metódus futása után az LPTSTR típusú változó tartalmazza ezt a könyvtárat.

3.3.2.2.1. Java Native Access

A Java Native Access – röviden JNA – egy API, mely segítségével könnyen hozzá lehet férni a natív megosztott könyvtárakhoz, ez által Windowson a dinamikus hivatkozású könyvtárakhoz is (DLL).

A JNA használatához nincs szükség C/C++ kód írására, ahogyan a Java Native Interface (JNI) esetén. Egyszerűen definiálni kell egy interfész, ami megfelel a C++-ban írt fejlécnek. Ez után a JNA segítségével létrehozható egy objektum, mely implementálja a megírt interfész és a megadott dll fájl metódusait futtatja.

A shell32.dll fájl SHGetFolderPath metódusának a hívásához definiált interfész:

```
private static class HANDLE extends PointerType {}

private static class HWND extends HANDLE {}

private static interface Shell32 extends Library {

    public static final int MAX_PATH = 260;
    public static final int CSIDL_LOCAL_APPDATA = 0x001c;
    public static final int SHGFP_TYPE_CURRENT = 0;
    public static final int SHGFP_TYPE_DEFAULT = 1;
    public static final int S_OK = 0;

    public int SHGetFolderPath(final HWND hwndOwner, final int nFolder,
        final HANDLE hToken, final int dwFlags, final char pszPath[]);
}
```

A Shell32 interfész megvalósítása és példányosítása a JNA segítségével és a könyvtár megszerzése:

```
String path;
Map<String, Object> options = new HashMap<String, Object>();
options.put(Library.OPTION_TYPE_MAPPER, W32APITypeMapper.UNICODE);
options.put(Library.OPTION_FUNCTION_MAPPER, W32APIFunctionMapper.UNICODE);
HWND hwndOwner = null;
int nFolder = Shell32.CSIDL_LOCAL_APPDATA; // a kért könyvtár definiálása
HANDLE hToken = null;
int dwFlags = Shell32.SHGFP_TYPE_CURRENT;
char pszPath[] = new char[Shell32.MAX_PATH];
Shell32 instance = // Shell32 példányosítás a shell32.dll használatával
    (Shell32) Native.loadLibrary("shell32", Shell32.class, options);
int hResult =
    instance.SHGetFolderPath(hwndOwner, nFolder, hToken, dwFlags, pszPath);
if (Shell32.S_OK == hResult) {
    path = new String(pszPath); // a karakter tömb szöveggé alakítása
    path = path.substring(0, path.indexOf('\0')) // vág az adat vége
karakterig
}
else {
    // ha a Shell32 nem működne, az alapértelmezett útvonal használata
    path = System.getProperty("user.dir");
}
return path;
```

3.3.2.3. Összefoglalás

Ebben a fejezetben – a teljesség igénye nélkül – bemutattam a vezérlőkliens fontosabb elemeit. Szó volt a felületről, ezen belül a megjelenésért felelős Look And Feel osztályról, a webböngésző támogatásról, melyhez az SWT lett felhasználva és a natív interfészről, mely segítségével a Swing és az SWT egyazon alkalmazásban igénybe vehető. Ezen kívül bemutattam, hogyan hozható létre többnyelvű alkalmazás asztali környezetben Java alatt és hogy mi a leggyorsabban működő töltőképernyő létrehozásának a módja. Végezetül bemutattam a JNA API-t, amivel egyszerűen lehet a megosztott könyvtárakban szereplő metódusokat használni.

3.3.3. A járműkliens fontosabb elemei

Ebben a fejezetben az Android fejlesztői környezet főbb elemeit fogom bemutatni a járműkliens alkalmazásban részt vevő szerepe szerint.

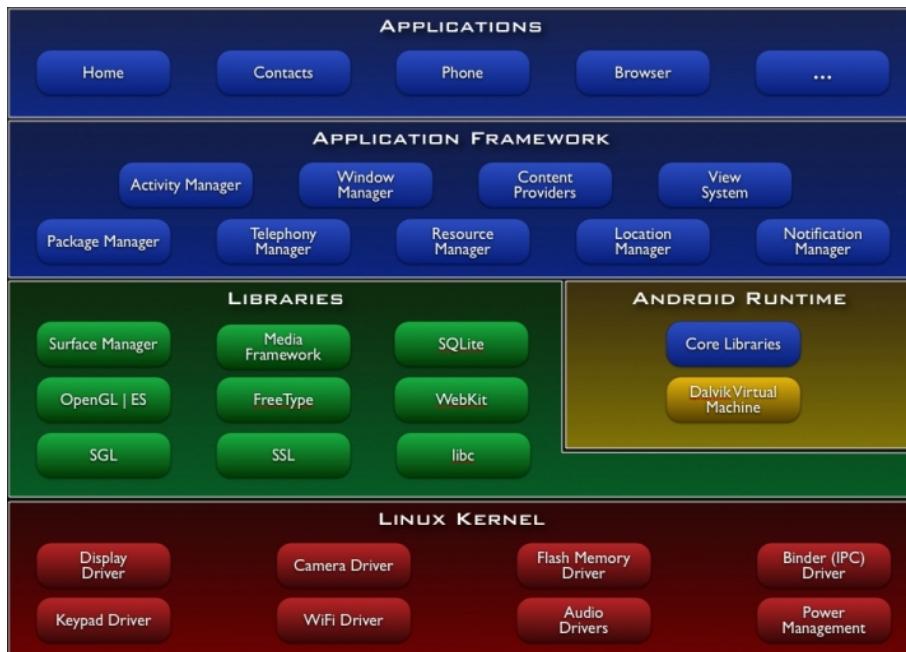
A járműkliens három fő részből tevődik össze. A felületen a főablak mutatja a vezérlőjelet és a feszültségszintet és itt lehet elindítani és leállítani a háttérfolyamatokat. Az alkalmazáshoz tartozik egy tárhely is, ahol a beállítások tárolódnak. A háttérfolyamatokat egy szolgáltatás biztosítja. Továbbiakban ezen részek felépítéséről lesz szó.

3.3.3.1. Az Android platform

Az Android^p egy Linux kernelt használó operációs-rendszer. A Linux kernel fölé egy Java virtuális gép van helyezve, melynek neve Dalvik virtuális gép. A virtuális gép felelős a felhasználói felület kezeléséért és az alkalmazások futtatásáért.

A platform alapját a Linux kernel adja, amely tartalmazza a hardver által kezelendő eszközök meghajtó programjait. Ezeket azon cégek készítik el, amelyek az Android platformot saját készülékükön használni kívánják, hiszen a gyártónál jobban más nem ismerheti a mobil eszközbe integrált perifériákat. Ez a kis méretű kernel adja a memória kezelését, a folyamatok ütemezését és az alacsony fogyasztást elősegítő teljesítmény-kezelést is.

A kernel szolgáltatásait használják a Linux rendszerekben meglévő különféle programkönyvtárak, mint a libc, az SSL vagy az SQLite; ezek C/C++ nyelven vannak megvalósítva, és a Linux kernelen futnak közvetlenül. Részben ezekre épül a Dalvik virtuális gép, amely egyáltalán nem kompatibilis az Oracle virtuális gépével, teljesen más az utasítás készlete, és más bináris programot futtat. A Java programok nem egy-egy .class állományba kerülnek fordítás után, hanem egy nagyobb Dalvik Executable formátumba, amelynek kiterjesztése .dex, és általában kisebb, mint a forrásul szolgáló .class állományok mérete, mivel a több Java fájlban megtalálható konstansokat csak egyszer fordítja bele a Dalvik fordító. A virtuális gép más, mint a Java alatti megszokott virtuális gép, vagyis a Java csak mint nyelv jelenik meg!



38. ábra: Az Android rendszer felépítése

A kék színnel jelölt részekben már csak Java forrást találunk, amelyet a virtuális gép futtat, s ez adja az Android lényegét: a látható és tapintható operációs rendszert, illetve a futó programokat. A virtuális gép akár teljesen elrejti a Linux által használt fájlrendszerét, és csak az Android Runtime által biztosított fájlrendszer láthatjuk.

3.3.3.2. Az Android SDK

A fejlesztőkörnyezetet az Eclipse IDE biztosítja. Az Androidra való szoftverfejlesztés előtt le kell tölteni egy plugint az Eclipse-hez és magát az Android SDK-t is. A plugin telepítése után a hozzá tartozó beállításokban meg kell adni az Android SDK helyét. A Window menüből ez után elindítható az SDK manager, ahonnan letölthető a különféle Android rendszerekhez kiadott API. Ez után a Virtual Device Manager segítségével virtuális telefont is létre lehet hozni, így az egyszerű alkalmazások teszteléséhez nincs szükség valódi telefonra.

Az Android SDK sajátos MVC modellel rendelkezik. A Java nyelven kívül az XML-nek is nagy jelentősége van az Android alkalmazások fejlesztésekor. A nézeteket is XML nyelvben definiálják. A nézetet az Activity osztály jeleníti meg és a nézet és vezérlő ezen osztályban köthető össze az eseménykezelők használatával. A beállítások tárolására is lehetőség van XML fájlban definiált struktúra használatával, melyhez olyan nézet is rendelhető, mely a struktúra alapján létrehozza a felületet és segítségével a felhasználó módosíthatja a beállításokat. Maga a manifest fájl is XML formátumú.

3.3.3.3. AndroidManifest.xml

Az AndroidManifest.xml fájl nevezi meg a Java csomagok neveit, leírja az alkalmazás komponenseit. Meghatározza, melyik folyamathoz tartoznak az alkalmazás komponensei. Meghatározza azokat az engedélyeket is, amivel más alkalmazásoknak rendelkezni kell, hogy használni tudják a komponenseit, és meghatározza a minimum API szintet, ami szükséges az alkalmazás futtatásához.

A vezérlőkliens egyszerűsített manifest állománya a következő:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dyndns.fzoli.rccar.host"
    android:versionCode="29"
    android:versionName="1.2.0.29" >
    <uses-sdk
        android:minSdkVersion="7"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:allowBackup="true" >
        <uses-library
            android:name="com.android.future.usb.accessory"
            android:required="false" />
            <receiver android:name=".ConnectionIntentReceiver">
                <intent-filter>
                    <action android:name="android.intent.action.BOOT_COMPLETED" />
                </intent-filter>
                <intent-filter>
                    <action android:name="android.location.PROVIDERS_CHANGED" />
                    <category android:name="android.intent.category.DEFAULT" />
                </intent-filter>
                <intent-filter>
                    <action android:name="android.intent.action.PACKAGE_ADDED" />
                    <data android:scheme="package" />
                </intent-filter>
            </receiver>
            <activity
                android:name=".MainActivity"
                android:label="@string/app_name"
                android:screenOrientation="portrait"
                android:launchMode="singleTask">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
            <activity
                android:name=".SettingActivity"
                android:theme="@style/PrefTheme" />
        </application>
    </manifest>
```

A package attribútum határozza meg, hogy mi az alkalmazás alapértelmezett csomagja. A fájlban a ponttal kezdődő osztályok ebből a csomagból indulnak ki. A versionCode egyértelműen definiálja az alkalmazás verzióját, nem lehet eltérő kiadást ugyan azzal a kóddal kiadni. A versionName attribútum látható a felhasználók számára, mely általában MAJOR.MINOR.PATCH^q formátumú.

A uses-sdk tegben van definiálva az, hogy melyik rendszerre íródott az alkalmazás, valamint mi a minimum verziószám, amin még futtatható.

Az alkalmazás telepítése előtt a rendszer kijelzi, hogy milyen jogosultságokat vesz igénybe az alkalmazás. Ezen jogosultságok is a manifestben vannak felsorolva, egy jogosultság egy uses-permission tegnek felel meg.

Az application tegben vannak felsorolva a használt Activity és Service osztályok és a rendszerszintű események fogadására létrehozott IntentReceiver is. A receiver tegben vannak felsorolva azok a szűrők, melyekre az alkalmazás fel van iratkozva. Így például a BOOT_COMPLETED esemény hatására leprogramozható az alkalmazás elindítása a rendszer felállása után.

A manifest alapján a MainActivity nevű osztály kerül elindításra az alkalmazásindítóban szereplő ikon kiválasztásakor.

3.3.3.4. Az alkalmazás főablaka, Activity

Az alkalmazás főablaka egy közönséges Activity, mely kinézetéért az activity_main.xml felelős:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <org.dyndns.fzoli.rccar.host.ArrowView
        android:id="@+id/arrow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:layout_margin="5dp" />
    <TextView
        android:id="@+id/tv_voltage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/arrow"
        android:layout_centerHorizontal="true"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <Button
        android:id="@+id/bt_start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:minWidth="120dp"
        android:text="@string/text_start" />
    <Button
        android:id="@+id/bt_stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:minWidth="120dp"
        android:text="@string/text_stop" />
</RelativeLayout>
```

Az elrendezés menedzsernek a RelativeLayout van használva. Segítségével egyszerűen lehet komponenseket a képernyő közepére igazítani, és a komponenseket is egyszerűen lehet egymás mellé helyezni.

Minden komponensnek adható egy egyedi azonosító. Ezen azonosító segítségével lehet a Java kódban hozzáérni a felületi komponensekhez.

A TextView egy címke, ami a JLabel-nek feleltethető meg az asztali Java alkalmazásokban. A feszültség megjelenítésére van használva, és az `android:layout_centerHorizontal="true"` tulajdonság beállításával az elrendezés menedzser a képernyő közepére helyezi a címkét a vezérlőjelet ábrázoló komponens alá az `android:layout_below="@+id/arrow"` beállítás hatására.

A vezérlőjelet ábrázoló komponens saját komponens, nem része az Android SDK-nak, ezért a teljes nevét ki kell írni. Az ArrowView komponens a képernyő közepén helyezkedik el, csak annyi helyet foglal el, amennyi szükséges számára és margója 5 sűrűség független pixel. A sűrűség független pixel (density-independent pixel) segítségével az eltérő felbontású kijelzőkön is azonos távolságú margó állítódik be.

A képernyő alján látható Start és Stop nyomógombok valójában Button komponensek, melyek felirata a nyelvi fájloktól függ. Az `android:text` értéke a `"@string/text_start"` kulcsú szövegtől függően értékelődik ki. A lokalizációról később még szó fog esni.

A Java forráskódban az Activity létrejötte után lehet a felületet legenerálni az XML fájlból a `setContentView(R.layout.activity_main)` utasítás használatával. Ez után a legyártott komponensek referenciaja a `findViewById(int)` metódussal szerezhető meg.

Példa a feszültségszintet megjelenítő címke referenciájának a megszerzésére:

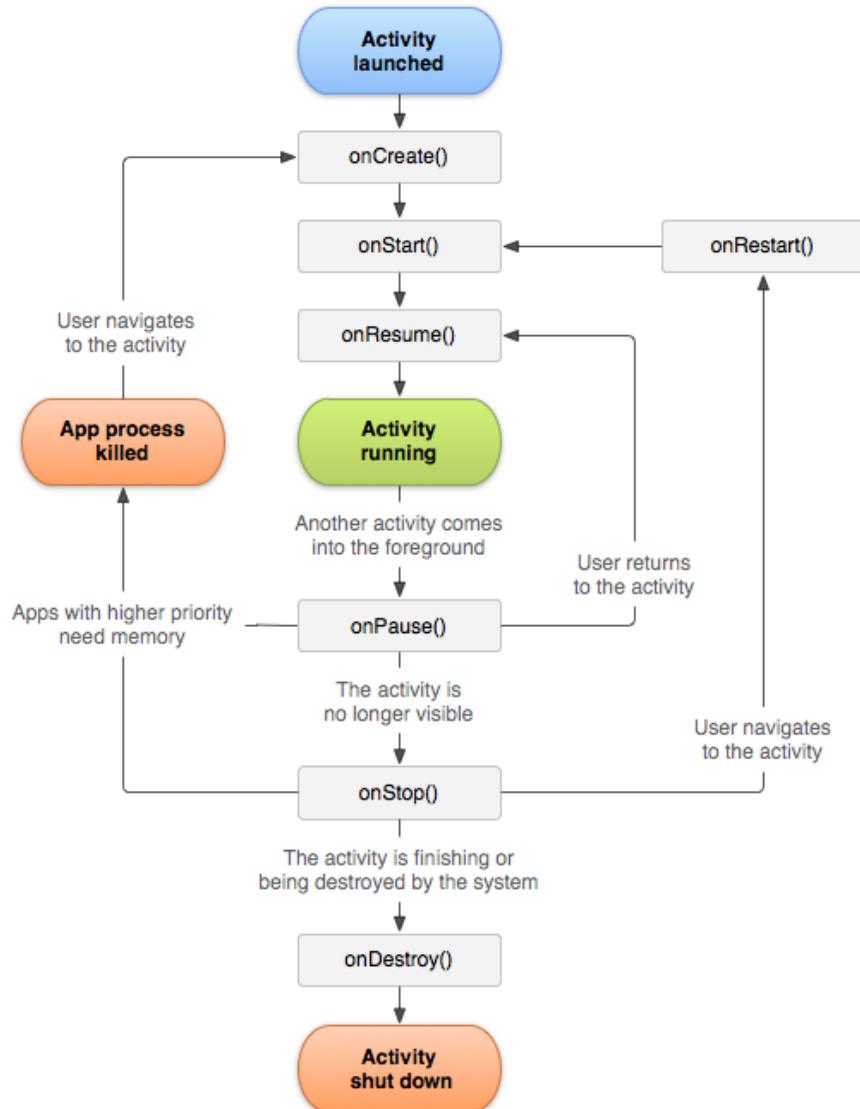
```
tvVoltage = (TextView) findViewById(R.id.tv_voltage);
```

Az R nevű osztály a Resource rövidítése. Az osztályt az SDK generálja az XML fájlok alapján. Kizárolag azonosítókat tartalmaz, melyek segítségével címezhetők az XML fájlok és az azokban szereplő azonosítóval ellátott komponensek is. A layout könyvtárban szereplő activity_main.xml tehát az R.layout.activity_main azonosítóval érhető el, az abban szereplő tv_voltage azonosítót használó komponens viszont az R.id.tv_voltage azonosítóval címezhető.

3.3.3.4.1. Az Activity életciklusa

Minden Activity indíthat másik Activityt. Amikor egy új Activity indul, az előző megáll és ezt a rendszer egy verembe teszi. Ez a verem LIFO mechanizmus alapján működik, így ha a felhasználó visszatér az új Activityból, akkor az előző fog folytatódni. Az ilyen állapotváltozásokat különböző callback metódusok hívásával jelzi a rendszer az Activitynek. Az Activity életciklusát^r és a callback metódusokat a következő oldalon lévő ábra mutatja. Amikor egy Activity háttérbe kerül, a rendszer leállíthatja, hogy memóriát szabadítson fel más Activityk számára. Ekkor, ha újra meghívjuk az Activityt, akkor a rendszer újra létrehozza azt.

A főablak felületének generálása, a felületi komponensek referenciáinak megszerzése és a kapcsolat kialakítása a háttérfolyamattal az `onCreate()` metódusban fut le, mivel ezeknek csak az Activity létrejöttekor kell lefutni. Az `onResume()` metódus lefutásakor az alkalmazás ellenőrzi, hogy történt-e fatális hiba a háttérfolyamat futása közben és ha történt, leállítja azt. Az `onDestroy()` metódus lefutásakor a háttérfolyamattal kialakított kapcsolat zárul be, és ha a háttérfolyamat le lett állítva, akkor az IP Webcam alkalmazás is leállításra kerül, majd meghívódik az ős metódus, ami felszabadítja a memóriát a felületi komponensek és az Activity törlésével.



39. ábra: Az Activity életciklusa

3.3.3.4.2. SherlockActivity, ActionBarSherlock

Android 3.0 óta az Activity vékony címsorát felváltotta az Action Bar. Előnye, hogy azokon az eszközökön, melyeken nincs vissza gomb és menü gomb, nem kell emulálni őket, mivel az Action Bar helyettesíti ezeket a gombokat. A régi menüt felváltotta az action overflow, melyre kattintva lenyílik a menü, de a menü egyes komponensei action itemként is feltehetők a sávra. A beállításokat megjelenítő gomb a főablak felületén szintén egy action item.

Az én telefonom eredetileg 2.1-es Android verzióval lett kiadva, nem hivatalos forrásból sikerült a verziószámot megnövelni 2.3.3-ra, de így is alacsony a verzió ahhoz, hogy az Action Bar támogatva legyen. Ennek ellenére találtam olyan alkalmazásokat, melyek képesek ezt a sávot megjeleníteni a telefonomon. Ilyen többek között a Play áruház és a Superuser nevű alkalmazás is.

A megoldás az ActionBarSherlock library projekt használata. Segítségével egészen a 2.1-es verzióig megjeleníthető az Action Bar. Ezt úgy érték el, hogy azokon a rendszereken, melyeken nincs hivatalos támogatás, ott eltüntetik a címsort és helyére kirajzolják az eredeti sávval megegyező kinézetű utánzatot. Ha van hivatalos támogatás, az eredeti metódusok vannak használva.

Használata egyszerű. Csupán az Activity osztály helyett a SherlockActivity osztályt kell örököltetni. A SherlockActivity őse az Activity, így a kód módosítására utólag sincs szükség.

3.3.3.5. Az alkalmazás konfigurációja, PreferenceActivity

Az alkalmazás konfigurációjának a tárolását az Android rendszer végzi. Az xml könyvtárban létrehoztam egy preferences.xml nevű fájlt, ami definiálja a tárolandó adatok típusát, azonosítóját, az alapértelmezett értékeket és a megjelenítendő szöveget.

A preferences.xml egyszerűsített változata:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="@string/category_path">
        <EditTextPreference
            android:key="address"
            android:title="@string/title_address"
            android:summary="@string/summary_address"
            android:inputType="textUri"
            android:defaultValue="10.0.2.2"
            android:dependency="offline" />
        <EditTextPreference
            android:key="port"
            android:title="@string/title_port"
            android:summary="@string/summary_port"
            android:maxLength="5"
            android:inputType="number"
            android:defaultValue="8443"
            android:dependency="offline" />
    </PreferenceCategory>
    <PreferenceCategory android:title="@string/category_other">
        <ListPreference
            android:key="vehicle"
            android:title="@string/title_vehicle"
            android:summary="@string/summary_vehicle"
            android:defaultValue="0"
            android:entries="@array/vehicleNames"
            android:entryValues="@array/vehicleIndexes" />
        <CheckBoxPreference
            android:key="offline"
            android:title="@string/title_offline"
            android:summary="@string/summary_offline"
            android:defaultValue="false"
            android:disableDependentsState="true" />
    </PreferenceCategory>
</PreferenceScreen>
```

Az egyszerűsített xml fájlban látható két kategória definiálása. Az első csoportban két EditTextPreference van, melyek egy szerkeszthető beviteli mezőt jelenítenek meg. Az első EditText esetén normál billentyűzet jelenik meg, viszont a második szerkesztéskor kizárálag számok bevitele lehetséges, ezért numerikus billentyűzetet jelenít meg a rendszer.

A második kategóriában az első elem egy felsorolást jelenít meg, innen kapta a ListPreference nevet. A felsorolás elemeit az arrays.xml fájlból olvassa ki, mely egyszerűsített változata a következő oldalon található.

A CheckBoxPreference egy logikai értéket tárol. Ezen típust függőségnek is fel lehet használni. Alapértelmezés szerint, ha egy másik mező függ az adott CheckBoxPreference értékétől, csak akkor szerkeszthető az értéke, ha a CheckBox be van pipálva. Hogy ha fordított viselkedést szeretnénk, akkor a `android:disableDependentsState="true"` tulajdonság megadásával érhetjük azt el, így a többi Preference akkor lesz csak szerkeszthető, ha az adott CheckBox értéke hamis.

Az egyszerűsített values-hu/arrays.xml fájl tartalma:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="vehicleNames">
        <item name="prototype">Prototípus</item>
        <item name="pwm">PWM teszt</item>
    </string-array>
    <string-array name="vehicleIndexes">
        <item name="prototype">0</item>
        <item name="pwm">1</item>
    </string-array>
</resources>
```

Látható, hogy a listában megjelenő neveket és a hozzá tartozó értékeket két külön tömb definiálja.

A preference.xml fájlt a PreferenceActivity képes megjeleníteni. Mivel én az ActionBarSherlock projektet használom, hogy legyen visszamenőleg is Action Bar támogatás, valójában a SherlockPreferenceActivity osztályt használom a beállítások megjelenítésére és manipulálására.

A beállításokat kezelő osztályom neve SettingActivity. Az Activity létrejötte után az addPreferencesFromResource(R.xml.preferences) utasítással jelenítem meg az XML fájlban megadott komponenseket.

Az XML-ben beállított korlátozásokat input mező validálással bővítettem ki, melynek alapja az eseménykezelés. Két típusú eseményfigyelőt használom. A TextWatcher a szöveg módosulása előtt és után is képes eseményt fogadni, így ezzel elértem, hogy a biztosan tiltott karakterek nem kerülnek bele az EditText komponensbe. Az OnPreferenceChangeListener még a Preference módosulása előtt képes megakadályozni a módosítást, így a félbehagyott beviterek nem kerülnek mentésre.

Az Action Bar egyik elemének kiválasztásakor az alábbi metódus fut le:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_author: // információ a szerzőről
            new AlertDialog.Builder(this)
                .setTitle(R.string.author)
                .setMessage(getString(R.string.author) + ": Farkas Zoltán\n")
                .setIcon(android.R.drawable.ic_dialog_info)
                .setNeutralButton(android.R.string.ok,
                    new DialogInterface.OnClickListener() {

                        @Override
                        public void onClick(DialogInterface dialog, int which){
                            dialog.cancel();
                        }

                    }).create().show(); // dialógus megjelenítése
            break;
        case android.R.id.home: // vissza nyíl
            finish(); // az Activity bezárása
    }
    return super.onOptionsItemSelected(item);
}
```

A kódban példa látható a kiválasztott menüelem meghatározására és feldolgozására. A szerző adatait egy információs dialógus jeleníti meg; az alkalmazás logójára kattintva az Activity bezárul.

3.3.3.5.1. SherlockPreferenceActivity, az ActionBarSherlock előnye

Az ActionBarSherlock projektről már esett szó, de nem említettem még meg az alternatívákat.

A Google is gondolt azokra a felhasználókra, akik nem a legkorszerűbb rendszert futtatják a telefonjukon, ezért több úgynevezett support libraryt is kiadtak. Az appcompat v7 nevű library projekt tartalmazza az ActionBar támogatást egészen a 7-es API szintig, tehát a 2.1-es Android rendszerig.

Az egyetlen hátránya az appcompat librarynak, hogy kizárolag Activity osztályokat lehet vele helyettesíteni az android.support.v7.app.ActionBarActivity osztály segítségével, PreferenceActivity helyettesítésére jelenleg nincs lehetőség.

Ezzel szemben az ActionBarSherlock a PreferenceActivity osztály helyettesítésére is képes a SherlockPreferenceActivity osztály támogatásával, így az alkalmazásom beállítások ablaka is támogatja az ActionBar sávot.

3.3.3.6. Nyelvi fájlok

Az Android egy többnyelvű operációs-rendszer. A felhasználó az első indításkor kiválaszthatja, milyen nyelvű legyen a rendszer, melyet a Beállításokban utólag is bármikor lehet módosítani. A konvenciót betartó alkalmazások nyelve a rendszer nyelvével együtt változik. Ennek feltétele, hogy az alkalmazásba beégetett szövegek helyett a rendszer által támogatott lokalizációt használja a programozó.

Minden Android-alkalmazás tartalmaz egy res nevű könyvtárat. Az alapértelmezett nyelvhez tartozó XML fájlok a values könyvtárba kerülnek, de minden nyelvhez külön létre lehet hozni ezt a könyvtárat. A magyar nyelvhez például a values-hu könyvtár tartozik.

A nyelvi fájlok ezekben a könyvtárakban foglalnak helyet strings.xml néven. A rendszer a beállított nyelvnek megfelelő values könyvtárat használja, így értem el, hogy a magyar nyelvű rendszeren az alkalmazásom magyar nyelvű, de minden más nyelv esetén az alkalmazásom angol, mivel az az alapértelmezett nyelve az alkalmazásomnak. A nyelvi fájlok használatának bemutatását az alkalmazásom kódján keresztül mutatom be.

A res/values/strings.xml fájl egyszerűsített változata:

```
<resources>
    <string name="app_name">Mobile-RC</string>
    <string name="menu_settings">Settings</string>
    <string name="set_config">Please, set a correct configuration.</string>
</resources>
```

A res/values-hu/strings.xml fájl egyszerűsített változata:

```
<resources>
    <string name="app_name">Mobile-RC</string>
    <string name="menu_settings">Beállítások</string>
    <string name="set_config">Kérém, állítsa be jól a konfigurációt.</string>
</resources>
```

Az XML-ben deklarált feliratoknak megfelelően a generált R nevű osztály tartalmazza a szövegnek megfelelő kulcsokat. A beállítások szövegre ennek megfelelően az R.string.menu_settings Integer típusú változó hivatkozik.

Activity osztályból a getString(R.string.menu_settings) metódus használatával maga a String objektum is létrehozható, ha esetleg több szöveg összefűzésére lenne szükség, de String nélkül is lehet üzenetet megjeleníteni a felhasználó számára. Legegyszerűbb módon a Toast osztály használatával: Toast.makeText(this, R.string.set_config, Toast.LENGTH_SHORT).show()

Az előbbi utasítás egy két másodpercen keresztül látható üzenetet jelenít meg a használt nyelvi fájlnak megfelelően.

XML fájlból is lehet hivatkozni a strings.xml fájl szövegeire. Erre már láthattunk példát az AndroidManifest.xml fájl bemutatásakor, melyben az alkalmazás neve meg lett adva. A `@string/app_name` hivatkozik az alkalmazás nevére, így lehet nyelvfüggő nevet adni az alkalmazásnak.

3.3.3.7. Az alkalmazás háttérfolyamatai, Service

A Service egy felhasználói felülettel nem rendelkező komponens, melyet elsősorban hosszabb időigényű háttérfolyamatok elvégzésére használhatunk. Más komponensek indíthatják a Service-t és a háttérben akkor is tovább futnak, ha közben a felhasználó egy másik alkalmazásra váltott. Például háttérben történő zenelejátszás, hálózati kommunikáció, stb. Alapvetően 2-féle Service létezik.

Egy Service "started", ha egy alkalmazás komponens (például egy Activity) elindítja a startService(Intent) metódussal. Ha egyszer elindult a Service, akkor folyamatosan futhat még azután is, hogy az őt meghívó komponens megszűnt. A "started" Service-k nem adnak vissza értéket a hívónak.

Egy service "bound", ha egy alkalmazás komponens meghívja a bindService(Intent, ServiceConnection, int) metódust. Ez egy kliens-szerver interfész kínál, tehát kéréseket küldhetünk és válaszokat fogadhatunk vele.

Az én alkalmazásomban a Service minden típusba tartozik. Egyszerű szükség van a folyamatos futásra, hogy az áramkör vezérelhető legyen, valamint a folyamatos hálózati kommunikáció fenntartása érdekében, de a főablaknak is el kell tudnia érni a Service objektumait, hogy kirajzolhassa a vezérlőjelet, illetve offline mód esetében a vezérlőjel módosításra is szükség van.

3.3.3.7.1. IOIO Service

A jármű elektronikai részében már szó esett az IOIO eszközről és arról, hogy hogyan vezérli az áramkört. Ebben a fejezetben az IOIO programozásáról lesz szó.

Az eszköz gyártója biztosít egy library projektet, melyet az internetről lehet beszerezni. A projekt saját Activity és Service osztályt tartalmaz, melyek segítségével kapcsolat teremthető az IOIO-val. A `public IOIOLooper createIOIOLooper()` metódust mind az Activity, mind a Service esetén felül kell definiálni úgy, hogy egy IOIOLooper interfész megvalósító objektummal térjen vissza.

Az IOIOLooper használja a mikrogéppel kialakított kapcsolatot. Amint létrejön a kapcsolat, a `setup(IOIO)` metódus meghívódik paraméterül kapva egy IOIO objektumot, mely a kapcsolatot reprezentálja és a használható utasításokat definiálja. Ebben a metódusban kell lefoglalni az IOIO PIN-jeit, melyeket az alkalmazás használni fog. A lefoglaláskor definíálásra kerül a PIN használatának módja is. Így például az `ioio.openDigitalOutput(10, false)` utasítás digitális kimenetnek lefoglalja a 10-es PIN-t és kezdőértékét logikai hamisra állítja. A metódus visszatér egy DigitalOutput típusú objektummal, melyen keresztül a kimenet értéke állítható.

Az IOIO csatlakoztatása és beállítása után az Android alkalmazás és a mikrogép között állandó kommunikáció kezdődik meg. Mindaddig, míg az IOIO összeköttetésben van a telefonnal, a `loop()` metódus ismételten hívódik meg. Ebben a metódusban lehet a jelenlegi állapotnak megfelelően a kimeneteket módosítani és a bemeneteket olvasni.

Az IOIO leválasztásakor hívódik meg a `disconnected()` metódus. Ebben a metódusban tilos és értelmetlen az IOIO objektum használata, mivel a kapcsolat már lezárult. Ez a metódus csupán tájékoztatásul szolgál.

3.3.3.7.2. Impulzusszélesség moduláció (PWM)

Az IOIO kimenetei kizárolag digitális kimenetek. Ez azt jelenti, hogy egy logikai érték alapján vagy van jel, vagy nincs jel a kivezetésen. Ennek eredménye, hogy a kimenet feszültsége vagy 0 V vagy 3.3 V. Ez esetben viszont, ha mondjuk 1.65 V-ra lenne szükség, kizárolag trükkel valósíthatjuk ezt meg. Megtehetjük, hogy a digitális kimenetet meghatározott időközönként ki-be kapcsoljuk úgy, hogy egyforma ideig legyen ki és bekapcsolva a kimenet. Ez esetben a hosszú távú átlagfeszültség a 3.3 V fele lesz, tehát 1.65 V. Ha ennél nagyobb átlagfeszültséget szeretnénk, valamivel tovább kell bekapcsolva hagyni a kimenetet, és ha alacsonyabbat, akkor értelelem szerűen kevesebb ideig.

A PWM (Pulse Width Modulation) pontosan erre lett kitalálva. A PWM^s jel előállítása olyan alapvető szükséglet, hogy minden modern mikrovezárló hardveresen tartalmazza, nem kell szoftveresen előállítani, ezáltal tehermentesíti a processzort. A hardveres PWM esetén csak közöljük a hardverrel, hogy milyen periódusidejű és kitöltési tényezőjű PWM jelre van szükségünk, és ezzel a CPU dolga véget is ért, dolgozhat tovább más feladaton, a PWM hardver pedig előállítja a kívánt jelet a hozzá tartozó kimeneti lábon.

Az IOIO is hardveresen támogatja a PWM jelek generálását. A kimenetek lefoglalásakor az `ioio.openPwmOutput(10, 1000)` utasítás a 10-es PIN-t foglalja le és 1000 Hz kitöltési időt állít be. A metódus visszatér egy PwmOutput objektummal, melyet a loop() metódusban lehet használni. A kitöltési időt a `setDutyCycle(float)` utasítással lehet módosítani. A kitöltési idő paramétere 0 és 1 között lehet. Értelemszerűen a 0 felel meg a 0%-os és az 1 felel meg a 100%-os kitöltésnek.

3.3.3.7.3. A motor vezérlése

A motor vezérléséhez írtam egy Vehicle nevű interfész, mely öröklí az IOIOLopper interfészét. A Vehicle interfész implementálásával valósítható meg a jármű vezérlése. Két osztályt is írtam a jármű vezérlésére: az egyik DigitalOutput objektumokkal működik, a másik PwmOutput objektumokkal. Mindkettőben azonos módon zajlik a feszültségszint kiolvasása és a vezérlőjel megszerzése, ezért létrehoztam egy AbstractVehicle nevű osztályt, melyben a közös metódusokat írtam meg.

A Vehicles nevű osztály példányosítja a megfelelő Vehicle osztályt:

```
public static Vehicle createVehicle(ConnectionService service, int index) {
    switch (index) {
        case 1: // PWM-teszt
            return new PWMVehicle(service);
        default: // alapértelmezett jármű: Prototípus
            return new DefaultVehicle(service);
    }
}
```

A példányosító metódus paraméterül várja a Service referenciáját, hogy képes legyen a vezérlőjel olvasására, valamint az indexet, mely a konfigurációban van megadva.

A metódust a Service az IOIOLopper létrehozásakor használja:

```
public IOIOLopper createIOIOLopper() {
    return vehicle = Vehicles.createVehicle(this, Integer.parseInt(
        getSharedPreferences(this).getString("vehicle", "0")));
}
```

A Service tárolja a Vehicle objektum referenciáját, hogy később a szolgáltatáshoz kapcsolódó főablak képes legyen regisztrálni egy eseményfigyelőt, mely jelzi a feszültségszint változását.

3.3.3.7.4. A telefon szenzorainak kezelése

Az alkalmazás három szenzort használ: GPS, accelerometer (gravitációs tér) és mágneses tér.

Az Androidban a helymeghatározásra a LocationManager osztály használható. A rendszer a telefon pozíóját több forrásból is ki tudja deríteni, az eltérés a pontosságban mutatkozik meg. Város szintű helymeghatározáshoz az alkalmazásnak nem kell engedélyt kérnie, viszont a pontos helymeghatározáshoz igen.

A rendszerben a helyadatok megváltozása a LocationListener eseményfigyelővel detektálható. Ha a pozíció módosul, a rendszer eseménykezelője meghívja az onLocationChanged(Location) metódust. A Location tartalmazza a földrajzi szélességet (latitude) és hosszúságot (longitude), a tengerszint feletti magasságot (altitude), a pillanatnyi sebességet m/s-ban, valamint az adatok pontosságát méterben megadott sugárban értelmezve.

Ahhoz, hogy az eseményfigyelőt be lehessen regisztrálni, előbb szükség van a LocationManager referenciajára: (`LocationManager`) `getSystemService(Context.LOCATION_SERVICE)`;

Ez után az `addGpsStatusListener(LocationListener)` metódust használva lehet eseményfigyelőt regisztrálni kizárolag GPS jel figyelésére.

Az eseményfigyelő regisztrálása viszont önmagában nem elég, mert a rendszert meg is kell kérni, hogy kezdje meg a helymeghatározást. Enélkül a kérés nélkül nem lennének események, melyeket megkaphatna az eseményfigyelő.

A GPS jel megtalálásának gyorsításához a rendszer igénybe veheti a mobilhálózatot és lekérheti a műholdak pozíóját. Ez a szolgáltatás fizetős lehet, ezért a helymeghatározást korlátoztam kizárolag ingyenes forrásokra:

```
final Criteria criteria = new Criteria();
criteria.setCostAllowed(false);
criteria.setAccuracy(Criteria.ACCURACY_FINE);
provider = locationManager.getBestProvider(criteria, true);
if (provider != null)
    locationManager.requestLocationUpdates(provider, 1000, 10f, locationListener);
```

A helymeghatározás elindítását az utolsó utasítás indítja el és az események másodpercenként vagy 10 méterenként fognak keletkezni.

A telefon szenzorai által létrejött adatok (pl. mágneses térrősség) figyelése is eseménykezelésen alapszik. Erre a SensorEventListener eseményfigyelő használható. Ha egy szenzortól új adat érkezik, az onSensorChanged(SensorEvent) metódus fut le. A SensorEvent megadja, hogy melyik szenzortól érkezett az adat és a values float tömb tartalmazza a kapott értékeket.

A gravitációs térrősség és a mágneses térrősség 3-dimenziós adatok, így a values tömb három értéket tartalmaz. Az egyszerűség kedvéért létrehoztam egy tároló osztályt, mely tárolja a szenzoradatot és részadatba csomagolva elküldhető a hídszervernek. Ez alapján az esemény feldolgozása így néz ki:

```
public void onSensorChanged(SensorEvent event) {
    Point3D p = new Point3D(event.values[0], event.values[1], event.values[2]);
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        getHostData().setGravitationalField(p);
    }
    else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
        getHostData().setMagneticField(p);
    }
    fireSensorChanged();
}
```

A getType() metódus segítségével eldöntöm, hogy gravitációs vagy mágneses térerősség adatot kaptam-e és ennek megfelelően a jármű adatában módosítom a Point3D objektumot. Ez után meghívom a fireSensorChanged() metódust, ami elküldi a részadatot a hídszervernek.

A két térerősség adataiból megállapítható, hogy a telefon (és ezzel a jármű) merre néz. A mágneses térerősség viszont nem elég ahhoz, hogy ezt ki lehessen számolni, szükség van arra is, hogy a telefon hogyan áll. A gravitációs térerősség erre adja meg a választ.

3.3.3.7.4.1. Északi mágneses pólus, földrajzi észak

A mágneses északi pólustól való eltérés irányszögét (azimuth) a hídszerver számolja ki a szenzoradatokból, és küldi tovább a fokban megadott értéket a vezérlőklienseknek. Az Androidban mivel már volt támogatás ennek kiszámítására, a szerver oldalon felhasználtam a forráskódját.

Ez alapján az irány meghatározása így néz ki:

```
private static Integer getAzimuth(HostData d) {
    if (d != null) {
        Point3D acc = d.getGravitationalField();
        Point3D mag = d.getMagneticField();
        if (acc != null && mag != null) {
            float[] values = new float[3];
            float[] R = new float[9];
            float[] outR = new float[9];
            Location.getRotationMatrix(R, null, acc.toArray(), mag.toArray());
            Location.remapCoordinateSystem(R, Location.AXIS_X, Location.AXIS_Z, outR);
            Location.getOrientation(outR, values);
            int degree = Double.valueOf(Math.toDegrees(values[0])).intValue();
            if (d.getAdditionalDegree() != null) degree += d.getAdditionalDegree();
            return degree;
        }
    }
    return null;
}
```

Az így kapott adat viszont térképen megjelenítve becsapós lehet, mivel a mágneses északi pólus nem egyezik meg a földrajzi északi pólussal. Ráadásul a Föld mágneses tere időben is változik.

Szerencsémre az Android arra is ad támogatást, hogy könnyen kiszámolható legyen a szenzoradatokból a valós északi eltérés. Ehhez ismerni kell a helyet és az időt:

```
Float magneticDeclination = null;
Location l = locationManager.getLastKnownLocation(
    LocationManager.NETWORK_PROVIDER);
if (l != null) {
    GeomagneticField geomagneticField = new GeomagneticField(
        (float) l.getLatitude(), (float) l.getLongitude(),
        (float) l.getAltitude(), l.getTime());
    magneticDeclination = geomagneticField.getDeclination();
    Log.i(LOG_TAG, "magnetic declination: " + magneticDeclination);
}
```

Ez az adat a jármű adatainak elküldése előtt beállítódik, így a hídszerver a kiszámolt értékhez hozzáadja az így kapott elhajlást, melynek értéke pozitív, ha kelet felé esik a valódi északi pólus, és negatív, ha nyugat felé. Az írás pillanatában Dunaharasztin ez az érték $4,118^\circ$ volt.

3.3.3.8. Összefoglalás

Ebben a fejezetben – a teljesség igénye nélkül – bemutattam a járműkliens fontosabb részeit és magát az Android rendszer felépítését, fejlesztőkörnyezetét. Láthattuk, hogyan épül fel egy alkalmazás projekt, melyben a Java forráskódon kívül az XML fájlok is fontos szerepet kapnak. Bemutattam a felületet megjelenítő Activity osztályok általam használt típusait, és a hozzá tartozó XML fájlokat, melyek definiálják a felület kinézetét. Végül szó esett a háttérfolyamatokat üzemeltető Service osztályról, melyben a szenzorok, a hálózati kommunikáció és az IOIO vezérlése zajlik. Ezzel a járműkliens fő elemeit fel is vázoltam.

4. Befejezés: továbbfejlesztés

Az eddigi fejezetekben bemutattam a program funkcióit és útmutatást adtam hozzájuk, majd felvázoltam az alkalmazások hátterében álló gondolatokat, melyeket megvalósítottam. Befejezésként megemlítek pár továbbfejlesztési lehetőséget, melyek elvezethetőbbé tennék a jármű vezérlését és eladhatóbb lenne a projekt.

4.1. Hang továbbítása

A hang továbbítását nem implementáltam az alkalmazásokba, mert még tömörítést alkalmazva is nagyon megnövelné az adatforgalmat. A tömörítés másrészről a processzort is terhelné. Az én telefonom nem lenne képes ekkora terhelést feldolgozni és a kép és a hang is egyre jobban késne.

Viszont, ha rendelkezésre állna egy többmagos processzorral rendelkező telefon, azzal már továbbítható lenne a hang is, melyet Wi-Fi hálózaton díjmentesen lehetne használni. Egy viszonylag nagy területen már hasznos funkció lenne a hang továbbítása.

A hang továbbítása a járműről a hídszerverre tetszés szerint ki és bekapcsolható lenne, de minden vezérlőkliensen egyedileg lehetne ki-be kapcsolni a hang fogadását, ha az éppen engedélyezve lenne. A járművet irányító felhasználó is küldhetne hangot a járműnek, melyet az lejátszana. Ez igen vicces szituációkat eredményezhetne.

4.2. Összes jármű a térképen

A térképen egyelőre csak az aktuálisan figyelt jármű látható. Ki lehetne bővíteni a térképet úgy, hogy jelölje a hídszerverhez kapcsolódott többi autót is. Ehhez utána kéne járni annak, hogyan lehet egy adott GPS koordinátán egyedi kinézetű felületi komponenst megjeleníteni a Google térképen. Az ötletet a MÁV Start oldalán bevezetett térkép adta, ahol a vonatokat figyelemmel lehet kísérni.

4.3. Játék mód útvonaltervezéssel

Az eladhatóságot tovább növelné a virtuális játékvilág kiterjesztése a valóságba. A játékmód úgy nézne ki, hogy a hídszerveren előre definiálva lenne egy útvonal, ami a versenypályát definiálná. A járművek a versenypályán versenyezhetnek egymással, ahogyan az autós játékokban is. A rendszer nyomon követné a járművek útvonalát és az nyerne, aki az útvonalról nem letérve elsőként érne célba. Az eredmények a chat ablakon jelennének meg.

4.4. Egyedi karosszéria, nagyobb teljesítmény

A projekt szerintem akkor válna piacképessé, ha egy olyan egyedi karosszéria lenne kialakítva, melybe biztonságosan behelyezhető lenne egy okostelefon és nagy sebesség mellett is könnyen irányítható maradna. A telefont helyettesíteni lehetne egy integrált áramkörrel, de jobb ötletnek tartom a telefon meghagyását, mivel így sokkal olcsóbban lehetne árulni a járművet. Így Android okostelefonnal rendelkező felhasználók olcsón juthatnának gyors, mobilinterneten keresztül vezérelhető elektromos kisautóhoz. A projekt ezzel a célját sikeresen elérné.

Felhasznált források

- a <http://hu.wikipedia.org/wiki/Arduino>
- b <https://www.sparkfun.com/products/retired/10748>
- c <http://www.8051projects.net/dc-motor-interfacing/bjt-based-h-bridge.php>
- d <https://github.com/ytai/ioio/wiki/Analog-Input>
- e http://ilias.gdf.hu/_html/Elektrotech/343_feszltsgoszts.html
- f <http://pa-elektronika.hu/hu/cikkek/76-vasalasos-nyak-keszites.html>
- g <http://kapcsolasok.hu/nyak-keszites/30-fototechnikas-nyakkeszites>
- h <http://juliusdavies.ca/commons-ssl/>
- i <http://www.coderanch.com/t/473296/threads/java/ConcurrentModificationException-ObjectOutputStream-writeObject>
- j http://hu.wikipedia.org/wiki/Standard_Widget_Toolkit
- k <http://djproject.sourceforge.net/ns/>
- l <http://mchr3k.github.io/swtjar/>
- m <http://docs.oracle.com/javase/tutorial/uiswing/misc/splashscreen.html>
- n <http://docs.oracle.com/javase/6/docs/api/java/util/ResourceBundle.html>
- o <http://hu.wikipedia.org/wiki/JavaBeans>
- p [http://hu.wikipedia.org/wiki/Android_\(operációs_rendszer\)](http://hu.wikipedia.org/wiki/Android_(operációs_rendszer))
- q <http://semver.org/>
- r http://nyelvek.inf.elte.hu/leirasok/Android/index.php?chapter=2#section_2
- s http://www.hobbielektronika.hu/cikkek/will-i_epitese_avagy_nullarol_a_robotokig_-avr_mikrovezerlok.html?pg=8