



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 Javában.....	33
3.3.1.2. Eszköz- és kapcsolataazonosító.....	34
3.3.1.3. Titkosított kapcsolat és felhasználóazonosítás.....	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 kontra 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).....	45
3.3.2.1.5. A kamerakép megjelenítése, átméretezés (imgsrc).....	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. Az összes jármű a térképen.....	62
4.3. Játékmód 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-en.....	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: A DisconnectProcess és a MessageProcess osztálydiagramja.....	39
37. ábra: A Data és a PartialData osztálydiagramja.....	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 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 érthető módon írja le a diplomamunka célját, valamint az alkalmazások képességeit és azok használatát. A harmadik fejezetben olvashatók a háttérben meghúzódó, mérnöki szempontból lényeges működési elvek.

A szakdolgozatban kétfajta jelzést használtam. Az egyik a szám alapú felső index, ami azt jelzi, hogy az oldal alján az aktuális fogalom lábjegyzetben meg van magyarázva vagy ki van bővítve. A másik jelzés a betű alapú felső index; az az utolsó oldalon található végjegyzetet jelöli. A végjegyzetben a diplomamunka írása közben felhasznált forrásokat tüntettem fel. A betűkből világosabban látszik, hogy melyik forrás mit tartalmaz.

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

Kellemes olvasást kíván
Farkas Zoltán

1. Bevezetés

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

A jármű irányításához a vezérlő számítógépnek el kell érnie a járművet és minden félnek egyazon „nyelvet” kell beszálnie, hogy értsék egymást. Mivel tárvezérlésről van szó, a felhasználó közvetlen környezetéből nem kap jelzést a jármű helyzetéről és állapotáról. Ebből az következik, hogy a járműre különféle szenzorokat kell elhelyezni, hogy segítsék az irányítót a tájékozódásban. A legalapvetőbb követelmény az, hogy legyen egy kamera, amelynek jelét jól lehet látni a számítógép monitorán. Sokat segít az is, ha a felhasználó látja, hogy hol van a jármű, merre néz, mekkora a pillanatnyi sebessége, vagy ha tudja a felhasználó, mekkora távot tehet még meg a járműve.

E követelmények alapján és a rendelkezésemre álló eszközök figyelembe vételével úgy határoztam, hogy egy Android operációs rendszert futtató telefont használok a jármű oldalán. A telefonban van Wi-Fi¹ adapter, valamint HSPA² modem, így a kommunikáció lebonyolítható a helyi hálózaton és az interneten át is. A telefon ezenfelül tartalmaz még gyorsulásmérőt³, magnetométert⁴, GPS⁵ szenzort és a hátoldalán egy 5 megapixeles mikrokamerát. A jármű elejéhez rögzített telefonról élő kép sugározható. A gyorsulásmérő és a magnetométer adatait egybevetésével megtudható, hogy a telefon hány fokkal tér el az északi mágneses sarktól, megtudható tehát, hogy merre néz a telefon és abból az, hogy merre néz maga az autó. A GPS szenzor alapján méter pontossággal megtudható, hol tartózkodik a jármű, és az is megállapítható, hogy nagyjából mekkora a pillanatnyi sebessége. A térbeli elhelyezkedés és a pontos idő ismeretében meghatározható a valódi északi iránytól való eltérés is, abból pedig már térképen is megjeleníthető a jármű haladási irányá.

Elektronikai szempontból a telefont alkalmassá kellett tennem a járművet pontosan irányító jelek leadására. (Pontos irányításon azt értem, hogy a jármű sebessége és kanyarodásának iránya tetszés szerint állítható legyen.) A telefont ismervén, kétféle átviteli közeg közül választhattam: az USB⁶ kábel és a Bluetooth⁷ között. (Mivel a telefonban csak egy Wi-Fi adapter van, az nem jöhettett szóba, mivel az már hálózati kommunikációra volt lefoglalva az előző bekezdésben leírtak szerint.) Mindkét közegnek vannak előnyei és hátrányai is. Ha a telefon és a jármű összeköttetésére az USB kábelt választom, akkor stabil, gyors és biztonságos lesz a kapcsolat, s azt nem zavarja meg más jeladó; ha viszont Bluetooth alapú kapcsolatot alakítok ki, akkor mobilról is irányítani lehet a járművet. A vezeték nélküli kapcsolatot sajnos könnyen működésképtelenné lehet tenni. Egyszerűen csak ugyanazon a frekvencián kell sugározni, mint a telefon, így a jel annyira legyengíthető, hogy a telefon és a jármű között a kommunikáció megszűnik. Éppen ezért úgy döntöttem, hogy a járművet és a telefont USB kábellel kötöm össze. Mivel van laptopom és otthoni Wi-Fi hálózatom, nem járt semmi hátránnyal, hogy megszűnt a lehetőség a telefonról való vezérlésre.

A feladatot hálózati szemszögből nézve felmértem, milyen nehézségekkel kell számolnom. Ehhez olyan használati esetet képzettem el, amely felöleli az összes problémát, de még viszonylag

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

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

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

Mobilinternet-szabvány; elmeletileg 28 Mb/s adatátviteli sebesség elérésére is alkalmas.

3 A gyorsulásmérő a rá ható tehetetlenségi erőket méri 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.

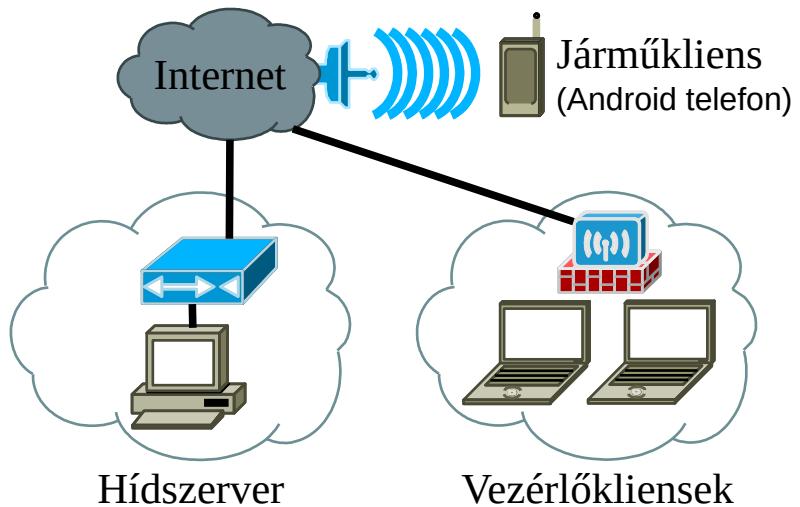
6 Universal Serial Bus, azaz univerzális soros sín. Nagyon elterjedt számítógépes csatlakozó.

Az USB 2.0 névleges adatátviteli sebessége 480 Mb/s.

7 A Bluetooth rövid hatótávolságú, adatcseréhez használt, nyílt, vezeték nélküli szabvány.

A Bluetooth 2.0-ás változata 3 Mb/s adatátviteli sebességet kínál a 2,4 GHz-es frekvenciasávban.

egyszerű hálózati struktúrát kíván. Arra jutottam, hogy három alkalmazást kell írnom ahhoz, hogy ötleteim 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, s ezáltal egy olyan elkülönített hálózat részévé válik, amelynek nincs más tagja. Ez a hálózat persze csak logikai szinten létezik: a mobiltorony hozza létre, a NAT⁸-olás segítségével. Ugyanahhoz a toronyhoz több mobileszköz is kapcsolódhat és mindegyik eszköz ugyanazt az internetes IP-címet (WAN⁹ IP) kaphatja. (A mobilszolgáltatók így érték el azt, hogy több ügyfelük lehessen egyszerre az Interneten, mint ahány IP-címet kioszthatnak. Az IPv4¹⁰ címek már mind ki vannak osztva a szolgáltatóknak, csak abból gazdálkodhatnak tehát, amire szert tettek. Az IPv6¹¹ címekre való átállás segíthet a helyzeten, de az költséges és időigényes folyamat, ezért nem egyhamar fog megtörténni széles körben.) Az én nézőpontomból ez csupán annyit jelent, hogy a telefon nem működhet szerverként, mivel egyetlen aktív kapuja¹² sincs, csakis kliensként használható. (Ez logikus is, mivel a WAN IP mögötti eszközök nem címezhetők egyértelműen, csak már akkor, ha van kiépített kapcsolat, vagyis minden irányban le van foglalva egy-egy kapu a kommunikációs csatornának.) A NAT-olás megkerülhető lenne VPN¹³ használatával, de az több problémával járna, mint ahányat megoldana: nem értenek hozzá a felhasználók és egy központi szerver is kellene hozzá; a VPN tehát nem jöhettet szóba. Ezek szerint a vezérlő programnak kellene szerverként üzemelnie?

Az ábrán a jobb oldali felhőben lévő eszközök is egy elkülönített hálózat részei. Ez a hálózat tekinthető egy étterem nyilvános hálózatának, de egy otthoni zárt hálózat is lehet éppen. Az

8 Network Address Translation, azaz hálózati címfordítás.

Lehetővé teszi a belső hálózatra kötött gépek közvetlen kommunikációját a külső gépekkel, éspedig úgy, hogy azoknak nem kell nyilvános IP-cím.

9 Wide Area Network, azaz nagy kiterjedésű hálózat.

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

12 A hálózati kapuk (portok) egy hálózati kommunikációs csatorna végpontjai.

A kapuk használata teszi lehetővé, hogy a bérkező csomagokból a számítógépen futó programok csak a nekik szóló csomagokat kapják meg. A kapu tehát egy – a konkrét gépen futó – konkrét programot címez meg.

Aktív kapun (porton) olyan hálózati kaput értek, amelyen át egy alkalmazás (a tűzfalakat figyelmen kívül hagyva) az Internet bármely pontjáról elérhetővé válik.

13 Virtual Private Network, azaz virtuális magánhálózat.

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

egyszerűség kedvéért ebben a hálózatba csak két számítógép tartozik, és minden gépen csak a vezérlő program fut. Arra a kérdésre, hogy melyik program legyen a szerver, ez az elkülönített hálózat adja meg a választ. A mobilinternet esetéből ismert akadályok itt is előjönnek: a vezérlő számítógép is NAT-olva van (például útválasztó révén), mivel nem közvetlenül kapcsolódik az Internetre, hanem egy belső hálózat része, nem az övé tehát a WAN IP-címe. Ebből az következik, hogy az ebben a hálózatban lévő gépeknek sincs aktív, a telefon mobilinternetéről elérhető kapujuk. Portátirányítással persze bármely gépnek kiosztható egy vagy több kapu, de azt csak a rendszergazda teheti meg, és a felhasználók természetesen ehhez sem értenek; a felhasználók otthoni hálózaton sem fognak ezzel bajlódni. Ha túzfal van beállítva, akkor még nehezebb a dolog. Ebből már sejteni lehet, hogy nem jó ötlet szerverként használni a vezérlő programot, de az még ennél is lényegesebb érv ellene, hogy több vezérlő számítógép is van a hálózatban, s azokról más-más felhasználók szeretnék irányítani a járművet. Sőt irányítható járműből is több van, vagyis a vezérlők és a járművek között sok-sokas a kapcsolat. Ha a vezérlő program lenne a szerver, nem lehetne több gépről nyomon követni a járművet, és a sok-sokas kapcsolatot sem lehetne kialakítani. A sok-sokas kapcsolat létrehozására a relációs adatbázisokban használt fogás itt is alkalmazható. A megoldás egy közbülső szerver beiktatása: az köti össze a vezérlőket a járművekkel. A válasz tehát arra a kérdésre, hogy a vezérlő program legyen-e a szerver: nem, az legyen egy harmadik program.

Ezt a szerveralkalmazást – a természetét figyelembe véve – Hídnak neveztem el. Ezt a szervert olyan gépen kell üzemeltetni, amelynek van aktív kapuja, a WAN IP és a kapu számának megadásával tehát bárki elérheti azt az Internetről, és vagy állandó IP-címe van, vagy doménnév alapján címzhető, tehát állandó a címe, és minden rá mutat. A többi feltétel (például az internethozzáférés, a processzor órajel) jelen nézőpontunkból nem érdekes. A hídszervert bárki üzemeltetheti, aki ezeknek a feltételeknek eleget tesz. Innentől kezdve a kliensalkalmazásokat használó felhasználók hálózati ismeretek nélkül is könnyen elérhetik a szervert, csak a pontos címet kell ismerniük. Az ábrán a bal oldali felhőben látható a fenti feltételeknek eleget tevő legegyszerűbb eset; a szerver egy közönséges kábelmodemen át közvetlenül kapcsolódik az internetre.

Összefoglalva, a három alkalmazás elnevezése és feladata:

- Járműkliens: a mobiltelefonon fut és az USB kábelen át vezéri a jármű áramkörét.
- Vezérlőkliens: asztali számítógépen fut, leadja a vezérlőjelet és megjeleníti 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 a szükséges eszközöket beszerezni, és leírást adok a használatukhoz. Az alkalmazások funkcióinak ismeretében sokkal egyszerűbb megérteni a mögöttük álló működési elvet; azt a későbbi fejezetekben fogom majd részletezni.

2.1. Rendszerkövetelmények

Az asztali számítógépekre kiadott hídszerver és vezérlőkliens Windows, GNU/Linux és Mac OS X rendszeren teszteltem; mindenkor rendszeren teljes funkcionalitással használhatók, 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

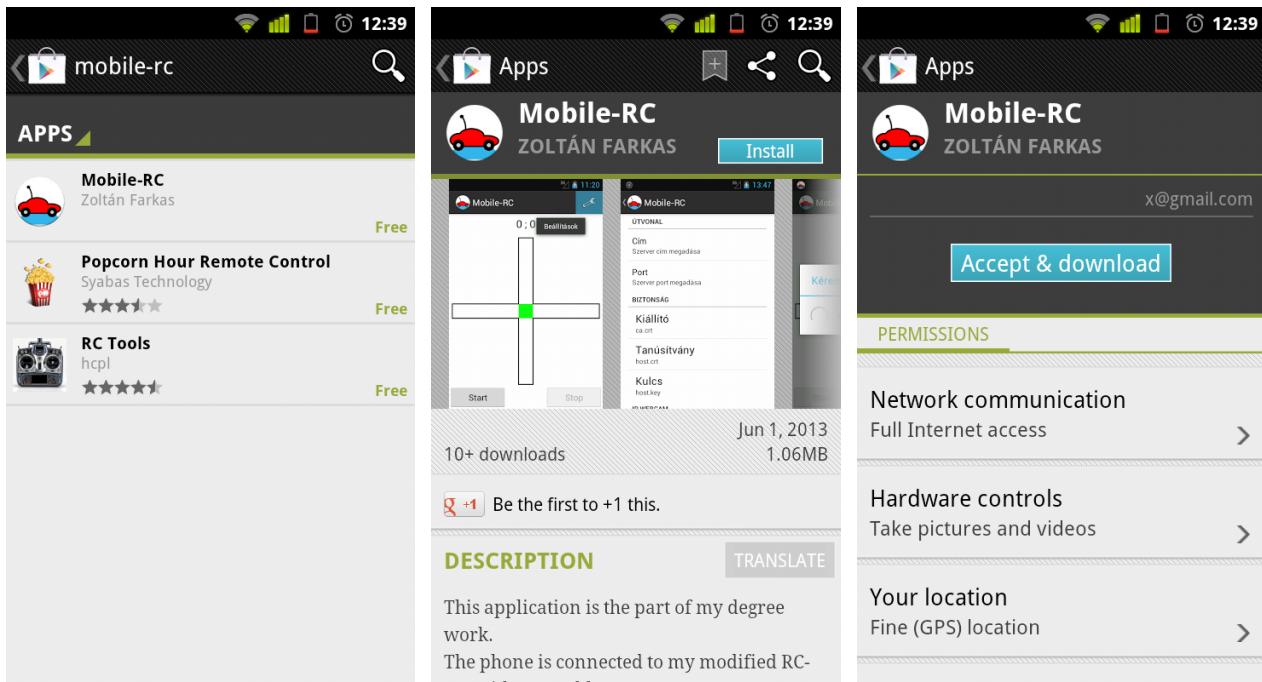
A telefonra írt járműkliens telepíthető a Google Playen át, és számítógépen is megtekinthető a következő címen: <https://play.google.com/store/apps/details?id=org.dyndns.fzoli.rccar.host>

Ha a Google Playben rákeresünk a „Mobile-RC” kifejezésre, akkor a találatok megjelenik között a program.

A hídszerver és a vezérlőkliens alkalmazásokhoz telepítő készült mindenkor operációs-rendszerre; ezek megtalálhatók a diplomamunkámhoz mellékelt lemezen; a legfrissebb verzió az Internetről is letölthető a [github.com oldalról: https://github.com/fzoli/RemoteControlCar](https://github.com/fzoli/RemoteControlCar).

2.3. Telepítés Android rendszerre

A Google Playben az alkalmazás megtalálása és kiválasztása után megjelennek az alkalmazás részletes adatai (például a leírás, képernyőképek). Ezen a lapon van mód az alkalmazás telepítésére.

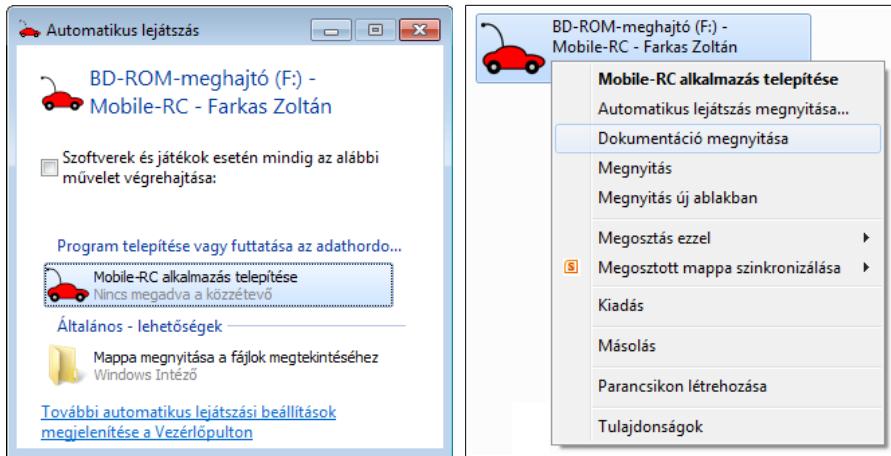


2. ábra: Google Play

Ha a telepítést választjuk, akkor megjelennek az alkalmazás által használt jogosultságok. Ezek tudomásul vétele és az az elfogadás után letöltődik és települ a járműkliens.

2.4. Telepítés Windows rendszerre

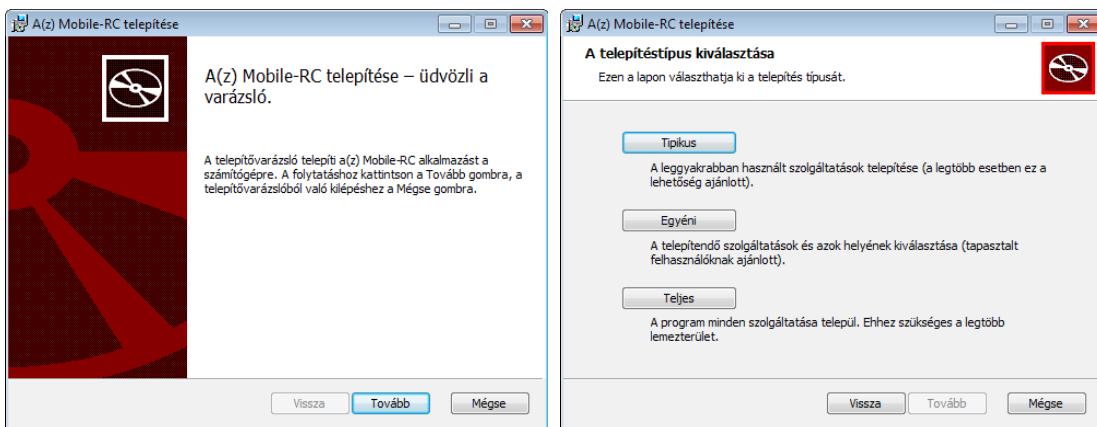
A mellékelt lemezt behelyezése után megjelenik a Windows automatikus lejátszás ablaka, s felkínálja az alkalmazás telepítését. Ha az optikai meghajtó menütől előhívjuk, akkor a diplomamunka PDF-változata is könnyen megtekinthető a Dokumentáció megnyitása opció választásával.



3. ábra: Automatikus lejátszás

Ha a telepítést választjuk, akkor az automatikus lejátszás a Setup.msi telepítőt indítja el. Windows XP SP2 vagy régebbi rendszereken nem biztos, hogy telepítve van a Windows Installer, pedig az szükséges a telepítő elindításához; ezért készítettem egy hordozható változatot is.

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

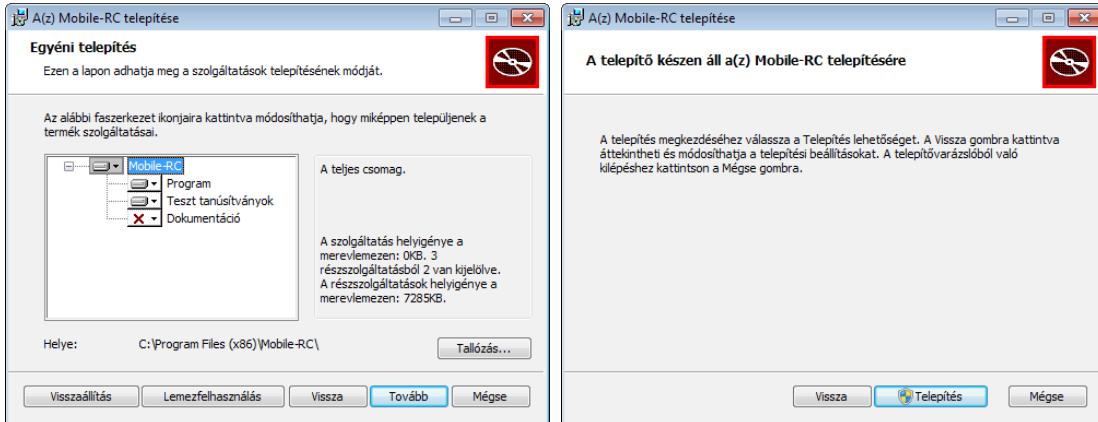


4. ábra: Windows telepítő – kezdőképernyő

A három telepítési mód a következő:

- *Tipikus* telepítés esetén az alkalmazás teszttanúsítványokkal együtt települ, így az alkalmazás könnyen kipróbálható az előre beállított konfigurációval.
- *Teljes* telepítéssel az összes komponens – az alkalmazás, a teszttanúsítványok és a diplomamunka 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 ilyenkor kézzel kell legyártani, kivéve ha a felhasználónak már van éles rendszerre saját tanúsítványa.

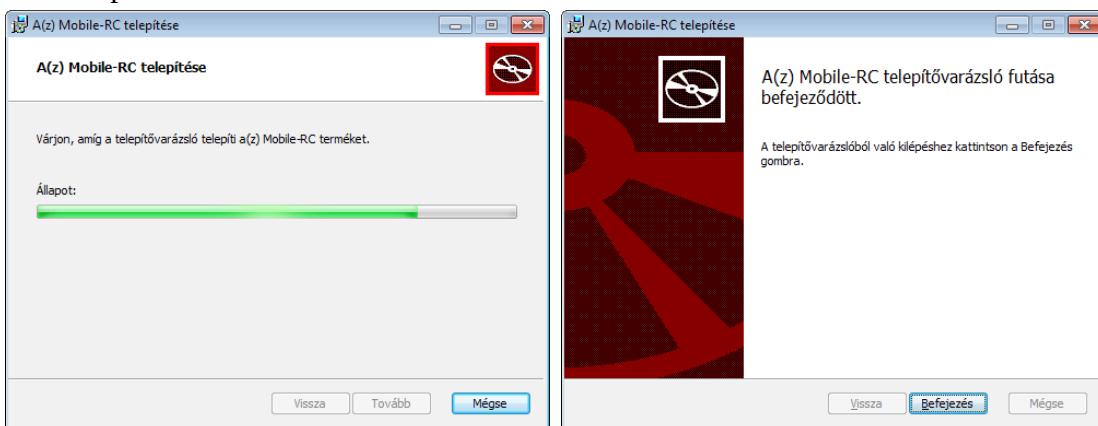
A telepítéstípus kiválasztása után a telepítő jelzi, hogy a telepítés elindítható; a Telepítés gombra való kattintással meg is kezdődik a telepítés.



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, akkor engedélyt kell adni a telepítőnek a munka megkezdésére.

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 szerveralkalmazás. A Start menüben továbbá megtalálható az alkalmazást eltávolító kód (az letörlő az esetleg feltelepített dokumentációt is). Az alkalmazás eltávolítható a Vezérlőpultról is.

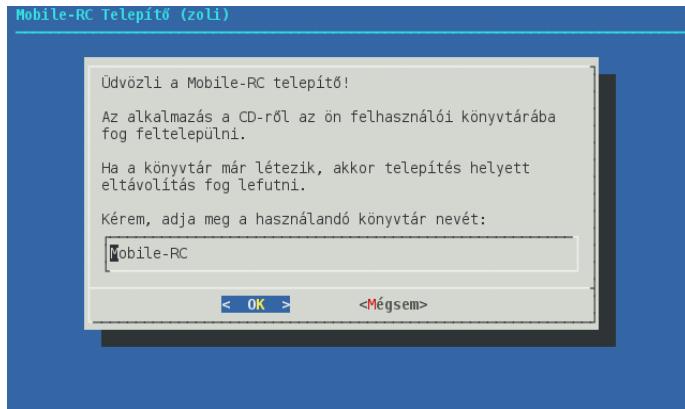
2.5. Telepítés GNU/Linux rendszerre

Linuxot főként szervereken használnak (például a Debiant), de léteznek otthoni használatra készített disztribúciók is, például az Ubuntu (annak is a Debian az alapja). Hogy a felhasználók és a rendszergazdák is kényelmesen használhatva vehessék az alkalmazásokat, Linuxra konzol alapú, de párbeszéddobozos telepítőt gyártottam. Így a felhasználók is egyszerűen telepíthetik az alkalmazást, és olyan kiszolgálógepekre is telepíthető a program, amelyeken nincs grafikus felület.

¹ User Account Control, azaz felhasználói fiókok felügyelete.

A Windows Vistával kezdve elérhető funkció; olyan kártevő programok ellen fejlesztették ki, amelyek rendszergazdai jogosultsággal próbálnak meg kárt okozni az operációs rendszerben. Az UAC jóvoltából a kártevő nem futhat le a háttérben, mivel előzetesen rendszergazdai jogot kell szereznie a felhasználótól még akkor is, ha a felhasználónak eleve rendszergazdai jogai vannak.

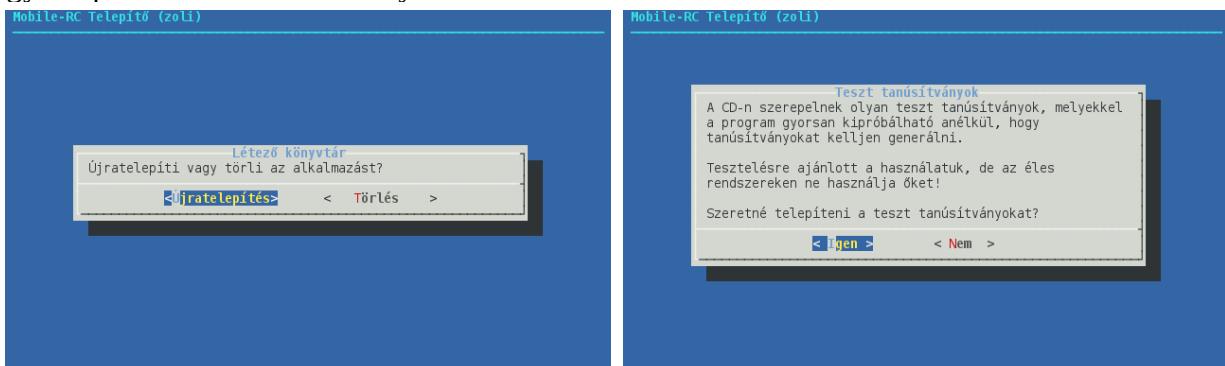
A telepítő elindítható a mellékelt lemezről a linux-installer.sh állomány futtatásával, vagy az Internetről letöltött linux-installer.bsx állomány futtatásával. Az első képernyőn megadható, hogy mely könyvtárba települjenek alkalmazások és a tesztanúsítványok. A könyvtár a felhasználó home könyvtárába települ, de relatív útvonal beírásával mélyebb könyvtár is kiválasztható.



7. ábra: Linux telepítő – kezdőképernyő

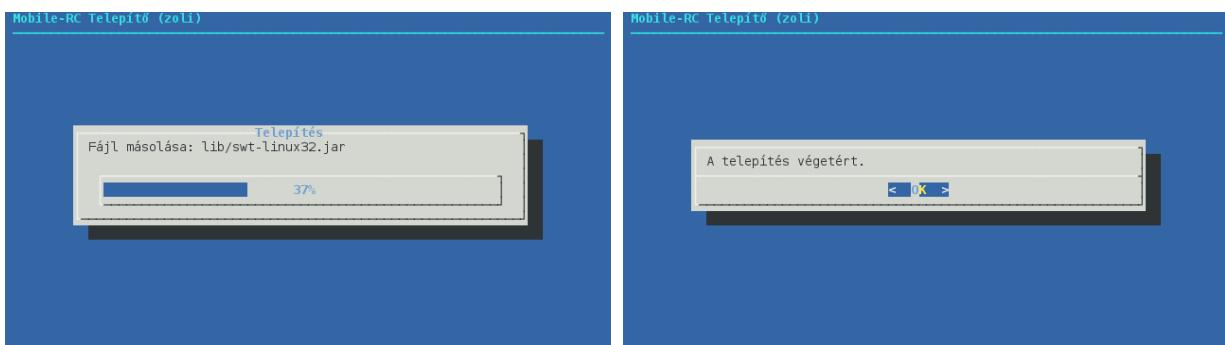
Ha a telepítőt véletlenül indítottuk el, akkor a Mégsem opcióval le lehet állítani a futását; a könyvtár megadása után pedig az OK opcióval lehet a folytatni a telepítést. Ha a folytatást választjuk, akkor már csak a varázsló működésének a varázsló befejeződése után léphetünk ki.

A telepítési útvonal megadása után a telepítő ellenőrzi, hogy létezik-e a megadott könyvtár. Ha létezik, akkor megkérdei, hogy újratelepítés vagy törlés fuzson-e le. A telepítő ez után megkérdei, hogy telepítse-e a tesztanúsítványokat.



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 tesz hivatkozásokat.



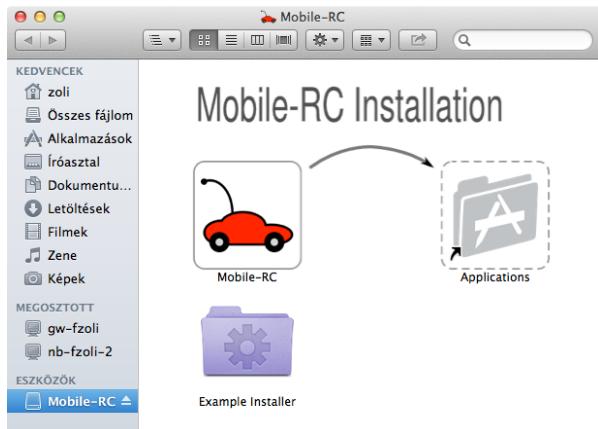
9. ábra: Linux telepítő – telepítés

A telepítés befejezése után az alkalmazások grafikus felületről a legegyszerűbben a létrehozott indítóikonokkal indíthatók el. GUI nélkül csak a Hídszerver futtatható, a telepítés megadott

könyvtáron belül a server.sh állomány elindításával.

2.6. Telepítés Mac OS X rendszerre

Az Apple a telepítések egyszerűsítésére olyan képállományt hozott létre, amely jól tömöríti a benne levő adatokat, így a DMG képállományba foglalt telepítők Internetről is gyorsabban letölthetők. OS X-en a telepítés két-három egérmozdulatból áll. A mellékelt lemezen a mac-installer.dmg állományra kell kétszer rákattintani; a rendszer azután felcsatolja (mount) a képállományt, majd 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 elindítása után a háttérben felsmásolódnak a tanúsítványok és a konfigurációs állományok. A háttérfolyamat befejeződése után a rendszer jelzi, hogy sikerült-e a másolás, vagy másolás közben hiba támadt.

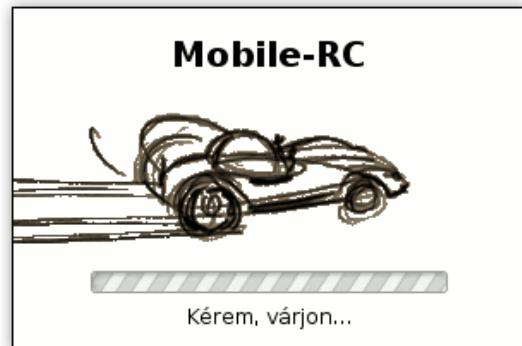
OS X-en is létezik a Windowson már megszokott szabványosított telepítővarázsló, de mivel alkalmazásom egyetlen könyvtárból áll, és nincs szükség semmilyen egyedi háttérfolyamat futtatására, azért nem hoztam létre külön telepítőt. Több OS X-re letölthető alkalmazás is ezt a módszert használja.

A telepítés után a képállomány 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 indulása után egy töltőképernyő jelenik meg, s az első látható ablak megjelenéséig látható is marad; jelzi, hogy a program már elindult, de még nem töltődött be teljesen. Erre azért van szükség, mert az indítás 10-20 másodpercbe is beletelhet. Ez az ára annak, hogy a felületi komponenseket az alkalmazás indulásakor előre legyártom; cserébe a betöltés után a felület gyorsan fog reagálni a felhasználó kéréseire.

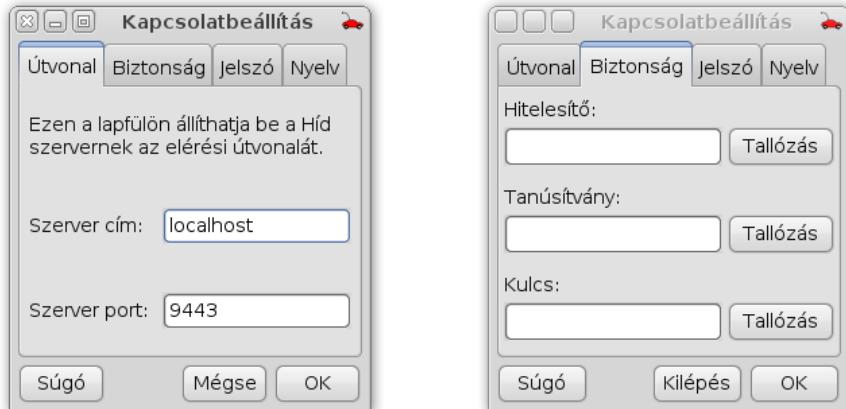


11. ábra: Töltőképernyő

A konfiguráció betöltődése után az alkalmazás megpróbál kapcsolódni a Hídszerverhez, feltéve, hogy már be van állítva a szerver elérhetősége és ismeretes a titkosított kapcsolathoz használt tanúsítványállományok helye.

2.7.2. Az alkalmazás beállítása

Ha szükség van a konfiguráció beállítására – például mert valaki áthelyezte a tanúsítványállományokat –, 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 a felhasználó vagy helyesen beállítja a kérő adatokat, vagy kilép a programból. Utólagos módosításkor a program nem áll le, csak bezárul az ablak.

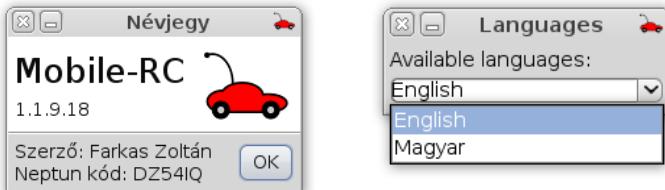
Az adatok megadása után az OK gombra való kattintással lehet elmenteni az adatokat; ha a felhasználó nem adott meg minden adatot, akkor figyelmeztető üzenet jelenik meg, és az ablak látható marad. A Mégse/Kilépés gombra kattintva nem változtatjuk meg a konfigurációt. Ha a felhasználó az ablakot bezárását az operációs rendszertől kéri, akkor módosulatlan konfiguráció esetén az ablak bezárul, egyébként meg egy Igen/Nem/Mégse dialógus jelenik meg, s megerősítést kér a konfiguráció mentésére és a bezárásra.

A kapcsolatbeállító ablakban további két beállítási lehetőség is elérhető. Ha a használt tanúsítvány jelszót igényel, és a jelszó el van mentve a konfigurációban, akkor a jelszó bármikor törölhető. Az utolsó lapfölön az alkalmazás nyelve állítható át. A nyelv az átállítás után azonnal módosul, nincs szükség újraindításra. Ha a felhasználó a Mégse gombot nyomja meg, akkor 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, egy ikont tettem az értesítési területre; annak a menüjéből elérhető egy angol nyelvű nyelvkiválasztó; a kapcsolatbeállító ablak és az alkalmazás névjegye is elérhető innen.

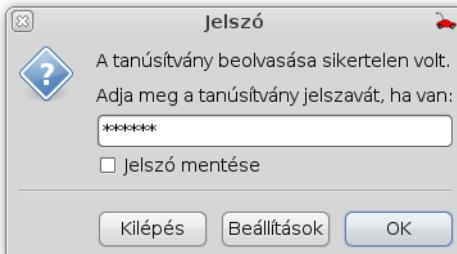


14. ábra: Névjegy és nyelvek

A konfiguráció beállítása után megkezdődik a kapcsolódás a szerverhez; ha már van kapcsolat, és a felhasználó átirta a szerver címét, akkor az alkalmazás megkérdezi, hogy kapcsolódjon-e újra. Ha a felhasználó itt véletlenül a Nem gombot nyomja le, akkor az értesítési ikon menüjéből bármikor kérhet újrakapcsolódást.

2.7.3. Kapcsolódás a Hídszerverhez, jelszóbekérés

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

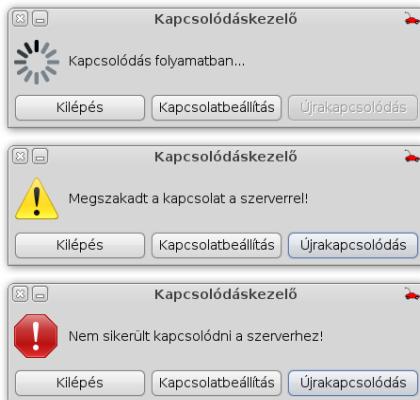


15. ábra: Tanúsítványjelszó bekérése

A jelszókérő ablak csak akkor jön elő, hogy ha jelszó nélkül nem sikerül megnyitni a titkos kulcsot (a key kiterjesztésű állományt). Hibás jelszó megadása esetén újra megjelenik a jelszókérő ablak, egészen a helyes jelszó megadásáig. Ha a felhasználó nem tudja a jelszót, kiléphet a programból, vagy átállíthatja a konfigurációt más tanúsítványra. Ha a tanúsítványhoz nem tartozik jelszó, akkor valószínűleg megsérült a crt vagy a key állomány.

2.7.4. Kapcsolódáskezelő ablak

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



16. ábra: Kapcsolódáskezelő ablak

A kapcsolódáskezelő ablak több hibaüzenetet is megjeleníthet:

- *Nem sikerült kapcsolódni a szerverhez!*

Ez a hibaüzenet arra utal, hogy a szerver nem érhető el. Ennek több oka is lehet: nem fut a szerver, vagy rossz a megadott cím. 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 doménnév van megadva IP-cím helyett, és nem sikerült a névfeloldás. Lehetséges, hogy a felhasználó elírta a címet, vagy nincs internetkapcsolat.

- *Időtúllépés a kapcsolódás közben!*

Ha a hálózat vagy az eszköz túlterhelt, akkor előfordulhat, hogy nem sikerül a tíz másodperces időkorláton belül kapcsolatot kialakítani. Ha az újrakapcsolódást választjuk, akkor jó eséllyel sikerül a kapcsolat létrehozása.

- *Az SSL² kapcsolat létrehozása nem sikerült!*

Ez a hiba akkor jelentkezik, ha sikerült kommunikálni a szerverrel, de a tanúsítványkiállító (legtöbbször ca.crt ennek az állománynak a neve) beállítása eltér a szerveren és a kliensen, vagy el van írva a cím, és teljesen más szerver van vele címzve. A megoldás a cím ellenőrzése, és ha az biztosan jó, akkor egyazon kiállító által generált kliens- és szervertanúsítványt kell használni.

Ritkább hibalehetőség az, hogy a kliens olyan tanúsítványt használ, amelyet szerverekhez állítottak ki. Ha valóban ez a hiba, a kapcsolódás pillanatában erről a szerver félreérthetetlen üzenetet küld az értesítési területen, sőt ez a szerver naplózásában utólag is látszik. Ez esetben le kell cserélni a tanúsítványt.

- *A szerver elutasította a kérést!*

Ha ez a figyelmeztetés jelenik meg, akkor sikerült kapcsolatba lépni a szerverrel, de a kliens által használt tanúsítvány tiltólistára került a szerveren, vagy az adott tanúsítvánnyal egy másik kliens már felkapcsolódott a szerverre. 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 kommunikáció tíz másodpercnél hosszabban kimarad, akkor minden oldal eldobja a kialakított kapcsolatokat, és a két fél között megszakad a kapcsolat. Kisebb kimaradások esetén a kapcsolat nem zárul be. Ha nagyobb a kimaradás, akkor valószínűleg túlterhelt a hálózat vagy az eszköz. Mobilinternet használata esetén előfordulhat, hogy gyenge a jel, vagy a forgalomkorlát meghaladása és a szolgáltató korlátozta a sebességet.

- *A tanúsítványok beállítása nem megfelelő!*

A hiba egyértelműen a kliens konfigurációjában keresendő. Ilyen hiba akkor fordul elő, ha beállítunk ugyan a három tanúsítványállományt, és azok léteznek is, sőt kiterjesztésük is megegyezik, és valóban tanúsítványok, de a crt és a key állomány nem egy tanúsítványt alkot, vagyis a nyilvános és a titkos kulcs tanúsítványa eltér egymástól.

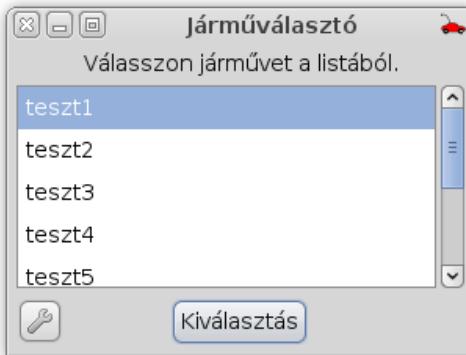
- *A szerver kliensekhez való tanúsítványt használ!*

A hiba egyértelműen a szerver oldalán keresendő. Ez a hiba akkor fordul elő, ha a szervernek kliensoldalra kiállított tanúsítványt állítanak be. Ha ez a helyzet, akkor egyetlen kliens sem kapcsolódhat a szerverhez.

2 Secure Socket Layer, azaz biztonsági alréteg. Az SSL protokollréteg a hálózati és az alkalmazási rétegek között húzódik, 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, s azon az összes, a felhasználó által választható jármű; a felhasználó közülük választhat.



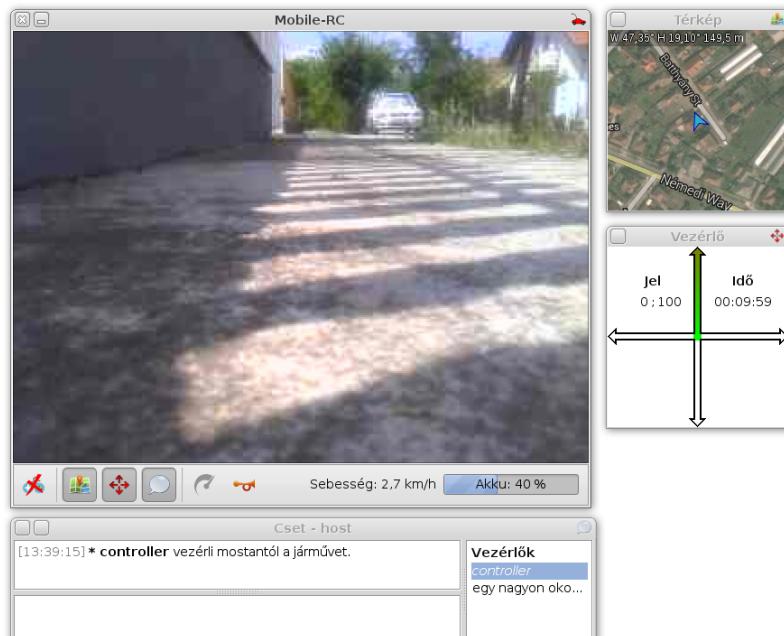
17. ábra: Járműválasztó ablak

A kapcsolatbeállító ablak innen is elérhető; nem minden rendszeren van ugyanis értesítési terület, ahonnan az alkalmazásikon menüjén át a felhasználó meghívhatná az ablakot.

A lista frissül, mihelyt egy jármű fel- vagy lekapcsolódik, de frissül akkor is, ha a felhasználó jogot kapott az egyik járműhöz, vagy éppen elvették tőle ezt a jogot. A listában nem szerepelnek offline járművek, így a kiválasztás gomb mindaddig inaktív marad, míg végre egy jármű fel nem kapcsolódik a szerverre.

2.7.6. Járművezérlés, a főablak bemutatása

A jármű kiválasztása után előjön az alkalmazás főablaka, és még melléje még három kisebb dialógusablak. A főablakon látható a 640×480 képpont felbontású élő kép a telefon kamerájából, valamint itt kérhető/adható vissza a vezérlés, és itt tüntethetők el vagy jeleníthetők meg a dialógusablakok is. Van még mellettük egy dudagomb is; azt csak az aktuális járművezérlő használhatja, és ha ő megnyomja, a telefon dudahangot játszik le. Ha a jármű precízen vezérelhető, akkor felbukkan itt egy vezérlésmódosító gomb is; ha az aktív, akkor folyamatos a gázadagolás.



18. ábra: Járművezérlő főablak és dialógusablakai

A főablak jobb alsó sarkában látható az autó akkumulátorának töltöttsége és az autó pillanatnyi sebessége. A sebesség csak akkor látszik, ha a telefonnak megfelelő GPS jele van a megbecsülésére. Az akkumulátor-szint csak akkor látszik, ha a járműnek nincs vezérlőparancs kiadva. A kijelzett százalékos érték ugyanis mindenkor a pillanatnyilag mért feszültségszint alapján számítható ki, és csak akkor szolgál megbízható adattal, 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; csak akkor jelenik meg, ha a jármű nem kapcsolódott fel a hídszerverre. A lekapcsolódás lehet szándékos, de meg is szakadhatott a kapcsolat a szerverrel.

- **Várakozás az összeköttetésre.**

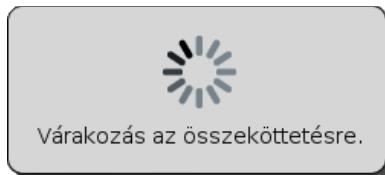
Akkor látható, ha a járműkliens kapcsolódott ugyan a hídszerverhez, de nincs összeköttetésben – az USB kábelen át – magával a járművel.

- **Várakozás a jármű kapcsolatára.**

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

- **Várakozás a Híd kapcsolatára.**

Ez is akkor jelenik meg, ha instabil a hálózati kapcsolat, de nem a jármű oldalán, hanem a vezérlőkliensén. Itt is igaz, hogy a kapcsolat 10 másodperces időtúllépés után megszakad. Ebben az esetben megjelenik a kapcsolódáskezelő ablak, s onnan újra lehet kezdeményezni a kapcsolódást a szerverhez. Sikeres kapcsolatfelvétel után – ha a szerver nem indult újra – 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; annak a bezárásával a kliensalkalmazás leáll. Ha a jármű offline, az alkalmazás megerősítést kér a jármű elhagyásakor, mivel a járműválasztóban már nem fog szerepelni a jármű, a kilépés tehát nem visszavonható művelet – legalábbis addig nem, amíg a jármű nem érhető el.

2.7.7. Járművezérlés, a jármű irányítása

A vezérléskérő gombra kattintással (gyorsgomb: Shift + Enter) lehet kérni a jármű vezérlését. Két eset lehetséges: a kliens vagy várólistára kerül, vagy azonnal megkapja a vezérlést. A művelet minden esetben visszavonható, le lehet iratkozni tehát a várólistáról, sőt a vezérlést kérő gomb (gyorsgomb: Shift + Backspace) használatával a vezérlésről is le lehet mondani. A vezérlés átvétele után a nyilakkal vagy a W, A, S, D gombbal lehet a járművet irányítani.

A vezérlésről való lemondás után a várólistán elsőként szereplő kliens kapja meg a vezérlést. A várólista főként időrend alapján áll elő, aki tehát előbb kérte a vezérlést, az előbb is fogja megkapni, de a szerveren pontosan konfigurálható, hogy melyik felhasználónak milyen prioritása van ezen vagy azon a járművön. Így előfordulhat az is, hogy az egyik vezérlő elveheti a vezérlést a másiktól, ha nagyobb a prioritása. Erről bővebben a hídszerver beállításairól szóló részben lehet olvasni.

2.7.8. Járművezérlés, a dialógusablakok bemutatása

A főablakhoz még három dialógusablak tartozik: a térkép, a vezérlő és a cset. Ha a felbontás engedi, az összes dialógusablak látható, de ha túl kicsi a képernyő felbontása, akkor esetleg 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, akkor 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 híján hibaüzenet jelenik meg; onnan elérhető az iránytű, és újra lehet próbálni a kapcsolódást a Google szerveréhez.



20. ábra: Térkép

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

A csetablak segítségével a járműhöz kapcsolódott felhasználók beszélgethetnek egymással, és itt jelennek meg rendszerüzenetek is: például 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, és ha van járműirányító, akkor a neve ki van emelve a listán.

2.8. A hídszerver használata

A hídszervert úgy írtam meg, hogy egyszerű legyen beállítani, és futtatható legyen azokon a rendszereken is, melyeken nincs grafikus felület – vagyis a szervergépeken. Ha a szervert tehát Linuxon a virtuális konzolból indítottuk el, akkor a konzolra íródik ki minden információ.

2.8.1. Az alkalmazás indulása

Az alkalmazás indulásakor ugyanúgy megjelenik a töltőképernyő, ahogyan a vezérlőkliens indításakor. Amint a szerver elindult, eltűnik a töltőképernyő. Ha a szerver beállításait tartalmazó állomány nem létezik, akkor az alkalmazás jelzi, hogy létrehozta azt az alapértelmezés szerinti beállításokkal, majd a program leáll. A következő induláskor a program felsorolja az esetleges kijavítandó hibákat. Ha a konfiguráció megfelelő, a szerver elindul; ha szükséges, előbb bekéri a tanúsítvány jelszavát.

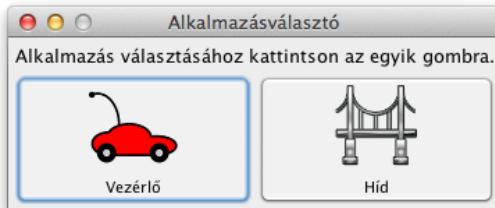


21. ábra: Szerver hibaüzenet

2.8.2. Az alkalmazás indítása Mac OS X alatt

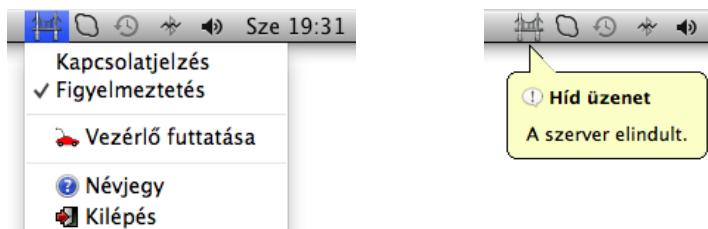
Figyelem: Mac OS X-en az alkalmazás kizártlag az Apple virtuális gépével indul el!

OS X-en az operációs rendszer által adott felületről alkalmazásonként csak egy folyamat (process) indítható el. Mivel a Mobile-RC kliens és szerver alkalmazása ugyanazon csomag része, jár hozzá egy alkalmazásválasztó is, hogy a felhasználó dönthessen, melyik alkalmazást akarja indítani.



22. ábra: Alkalmazásválasztó OS X-en

Ha szükség van a szerveralkalmazásra is, akkor az alkalmazásválasztóban a Hídszervert kell indítani. A szerver indulása után a menüből indítható el a vezérlőkliens, így OS X-en is egyszerűen indítható minden alkalmazás.



23. ábra: Értesítési terület OS X alatt

2.8.3. Az alkalmazás paraméteres indítása konzolból

Grafikus felület híján a kapcsolatjelzések és a figyelmeztetések futás közben nem állíthatók; erre az esetre a szerver három paramétert kínál grafikus menü helyett:

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

A paramétereknek nincs hatásuk, ha van grafikus felület, de ha nincs, és a felhasználó egynél több paramétert ad meg vagy a megadott paraméter nem egyezik a fenti három valamelyikével, akkor az alkalmazás megjeleníti a konzolon a súgót, majd leáll.

2.8.4. A Hídszerver konfigurációja

A Hídszerver konfigurációja az alkalmazás könyvtárában található, az állomány neve *bridge.conf*. Ebben az állományban adható meg, hogy a szerver melyik kaput figyelje, és hogy mely tanúsítványállományokat használja az SSL kapcsolat létrehozásakor. A konfigurációban opcionálisan a privát kulcs jelszava is 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
```

Az állományban minden – nem üres – sor egy paraméter. A sor a paraméter nevével kezdődik, azután egy szóköz jön, majd a paraméter értéke. Ismételt paraméter esetén az utoljára megadott lesz érvényes. Az állományba megjegyzés is írható a # jel használatával, ahogyan a példa is mutatja. Az állományok útvonalai lehet abszolút és relatív is, a példában relatív útvonal látható. Ha az opcionális paraméter hibás, akkor az alapértelmezés szerint érték lép érvénybe.

További opcionális paraméterek:

<i>timeout</i>	Percen belül megadott érték. A járművezérlők időkorlátja; ha a vezérlő ennyi időn át tétlen, akkor a hídszerver megvonja a felhasználótól a vezérlést. Értéke 1 és 120 között érvényes; alapértelmezésben 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. Alapértelmezésben minden logikai érték <i>false</i> .
<i>quiet</i>	Logikai érték. Ha aktív, akkor a program indulásakor az összes figyelmeztetés inaktív.
<i>hidden</i>	Logikai érték. Ha aktív, akkor a program a háttérben fut és az alkalmazás ikonja nem jelenik meg az értesítési területen.
<i>lang</i>	ISO 639-1-es nyelvkód (Pl. en, hu). Az alkalmazás nyelvét határozza meg. Alapértelmezés szerinti 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, amelyek nélkül az alkalmazás nem indítható el. A konfigurációs állomány programfutása alatti módosítása nincs hatással a szerverre, csak majd a következő indításkor.

A konfigurációs állományon kívül létrehozható három lista is; azok módosítása a szerver újraindítása nélkül is érvényesül, a szerver tehát zavartalanul adhat szolgáltatást. A három lista más-más célra való, de mindegyik a jogosultságkezelést szolgálja, és hasonlóan hat. Mindháromban egy-egy felhasználónév állhat soronként.

A listaállományokat két helyen keresi a szerver: először abban a könyvtárban, ahonnan a program elindult; ha ott nincs, akkor az alkalmazás könyvtárában nézi meg, és ha megvan, azt használja.

2.8.6. Tiltólista

Előfordulhat, hogy a tulajdonos egy konkrét felhasználót ki szeretne tiltani, mert az, mondjuk, megszerezte az Ő titkos kulcsát és ettől fogva már nem megbízható a tulajdonos tanúsítványa.

Ekkor jön jól a tiltólista; a *blocklist.conf*. Az állomány tartalma egy egyszerű felsorolás. Azok a felhasználók, akik ebben az állományban szerepelnek, nem kapcsolódhatnak a szerverhez és a listába való bekerüléskor éppen rajta vannak a szerveren, akkor a szerver bontja velük a kapcsolatot.

2.8.7. Fehérlista és feketelistá

A fehérlista és a feketelistá együttes használatával pontosan kezelhetők a jogosultságok. minden vezérlőfelhasználónak járművenként adható meg vagy vehető el a használati jog.

Mindkét listában járművenként lehet csoportokat kialakítani. Egy csoport a másik csoport kezdetéig tart. A csoportkezdő jel a jármű felhasználóneve, [és] karakterrel közrefogva. Ha az állomány nem csoportkezdő jellet kezdődik, akkor a felhasználók az alapértelmezés szerinti főcsoportba kerülnek. A főcsoport felsorolása az összes specializált csoport alá kerül be, de a duplázódás elkerülésére az alcsoporthoz már szereplő felhasználók nem kerülnek még egyszer, a csoport végére is.

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

Az alábbi példa érthetőbbé 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 alapértelmezés szerinti 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 jármű vezérlésének kérésekor nem mindegy, hogy a felhasználóneveknek mi a sorrendük. Tegyük fel, hogy controller3 vezérli a host nevű járművet. Ha controller2 is szeretné vezérelni, akkor a hídszerver azonnal átadja neki az irányítás jogát, mivel ezen a járműön magasabb a rangja; ha viszont controller1 kérné a vezérlést, akkor ő várólistára kerülne.

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

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

A fehérlista tehát jogot ad a járműhöz való kapcsolódáshoz és rangsort is felállít. A feketelista megvonja a járműhöz való kapcsolódás jogát – de csak akkor ha a fehérlistában nem szerepel erősebb utasítás. Ha minden listában ugyanolyan erős utasítás szerepel, akkor a feketelista jut érvényre.

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 ugyan a járműhöz, használhatja a csetet és láthatja a jármű adatait is, de nem kérhet – és nem is kap – vezérlést.

2.8.8. Naplózás

A szerveralkalmazás a naplózást is lehetővé teszi, és alapértelmezésben aktív is a naplózás. A naplóbejegyzések a *bridge.log* nevű állományba íródnak, és a naplószolgáltatás csak úgy kapcsolható ki, ha az állományrendszer erre az állományra nem ad írási jogot.

A naplóba bekerül 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* állomány 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 be a naplóba; a közönséges tájékoztatóüzenetek **INFO** jelzést kapnak. A járművezérlés átvételét a **=>** jel jelzi, a lemondását a **<=** jel. A példában tehát a controller nevű vezérlő megkapja a host járműhöz való vezérlés jogát, majd néhány másodperccel később lemond róla.

A naplózás hasznos lehet, ha a szervert megtámadják és tudni szeretnénk a támadó címét, vagy ha utólag valami miatt ki kell deríteni, hogy ekkor vagy akkor éppen 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, s azt azután meg is oszthatja az operációs rendszert használókkal. A gyári rendszer éppen ezért nem tartalmaz alkalmazást minden területre; minden felhasználó a szükségleteinek megfelelően választhat és szerezhet be alkalmazást. Ezzel a megoldással sok hely takarítható meg, mivel mindenki csak a neki hasznos alkalmazásokat telepíti fel.

A járműkliens teljes körű használatához szükség van egy olyan állománykezelőre, amely együttműködik más alkalmazásokkal; a járműklienshez emellett kell az *IP Webcam* nevű alkalmazás is. Magam az *OpenIntents File Manager* nevű állománykezelőt használom, főként azért, mert a rendszer ezt gyárilag tartalmazza. Az állománykezelő ahhoz kell, hogy ki lehessen tallázni az SD-kártyáról a tanúsítványállományokat. Az IP Webcamre a kamera hatékony megosztása miatt van szükség.

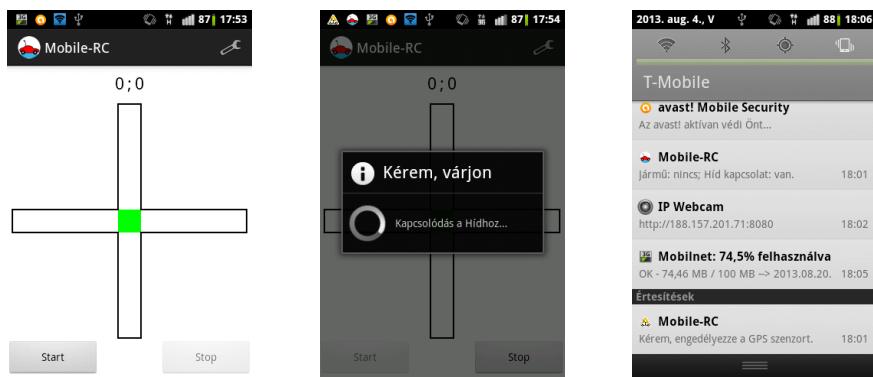
Ha az Android rendszer 3.1-nél régebbi, akkor az *USB hibakeresés* engedélyezésére is szükség van, mivel az ennél régebbi rendszerek nem működnek együtt az USB kapura köthető kiegészítő eszközökkel; ehelyett a hibakeresőn át folyik a kommunikáció a jármű áramkörével.

Az USB hibakeresés elérhetősége: Beállítások – Alkalmazások – Alkalmazásfejlesztés

2.9.2. Kapcsolódás a Hídszerverhez

A járműkliens a következőképpen tartja az összeköttetést a hídszerverrel: indítása után időről időre megkíséri a kapcsolódást mindaddig, amíg nincs kapcsolat, és ugyanezt teszi, ha a kapcsolat megszakad. A kapcsolódást az alkalmazás főablakából lehet kezdeményezni, és a kapcsolat mindaddig aktív marad, amíg a felhasználó ugyanitt le nem állítja. A kapcsolat ideje alatt a program háttérfolyamatként fut, a főablak tehát bármikor bezárható és előhozható.

Ahhoz, hogy a jármű pontos helyzete elérhető legyen, a GPS szenzor engedélyezése is szükséges. Ha a felhasználó ezt nem engedélyezi, akkor az alkalmazás figyelmeztetést küld neki a kapcsolódás kezdetekor.



24. ábra: A járműkliens

A hídhoz való kapcsolódás a *Start* gomb megnyomásával kezdődik meg. A kapcsolódás előtt az alkalmazás ellenőrzi a konfigurációt, és ha megfelelő a beállítás, akkor elindul a háttérfolyamat.

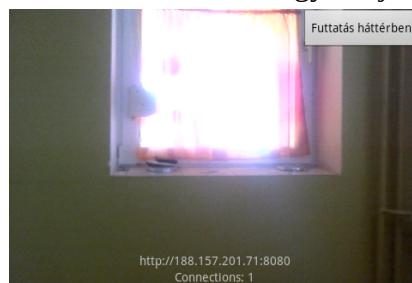
A háttérfolyamat kétféle üzenettípust jeleníthet meg:

Figyelmeztetés esetén a háttérfolyamat tovább fut, mivel nincs szükség felhasználói beavatkozásra. *Hibaüzenet* esetén a háttérfolyamat megszakad, és leáll, amint a felhasználó elolvasta az üzenetet.

Hibaüzenet több esetben is jelentkezhet. Hibaüzenet jelenik meg például, ha az IP Webcam alkalmazás nincs telepítve, a hibaüzenetre való kattintásra előjön a Google Play áruház, megjeleníti az alkalmazás részleteit, és a felhasználó ezután könnyen fel is telepítheti. Akkor is hibaüzenet jelenik meg, ha a megadott tanúsítványok nem tartoznak össze vagy hibás a tanúsítványjelszó; ha a felhasználó erre az üzenetre kattint, akkor a beállítások fognak megjelenni.

A hídhoz való kapcsolódás némelyik eszközön 1 percig is eltarthat, mivel időigényes beolvasni a memóriakártyáról a tanúsítványokat; ez a beolvasás azonban csak az első kapcsolódáskor történik meg, és amíg az Android nem törli a memóriából a tartalmukat, addig a kliens gyorsan fog kapcsolódni a szerverhez.

Sikeres kapcsolódás után megkezdődik a szenzoradatok továbbítása a hídhoz, valamint elindul az IP Webcam alkalmazás is – hogy szükség esetén gyorsan el lehessen indítani a kamerakép sugárzását is. Sajnos az Android fejlesztői nem adtak lehetőséget arra, hogy a kameraképet csak magában, valamilyen formában való megjelenítés nélkül is le lehessen kérni, ezért a program indulásakor felhözsa a főablakát; azt azután bármikor háttérbe lehet tenni, bezárni viszont nem ajánlatos, mert akkor leáll a program és a járműkliens a futása közben úgy is újra meghívja.



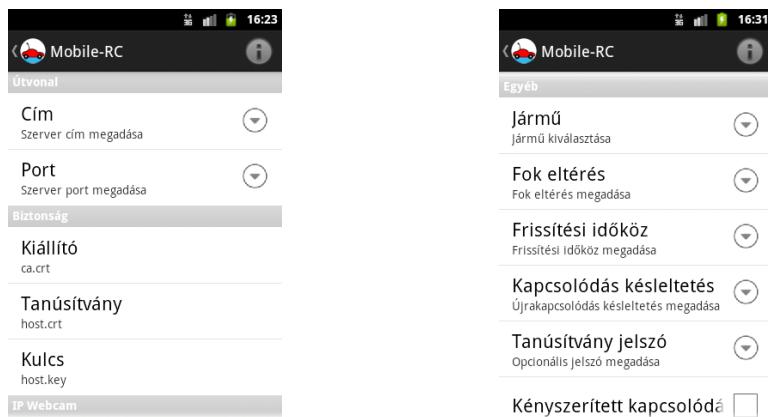
25. ábra: IP Webcam

Az adatforgalommal való takarékosság kedvéért a szenzoradatok tömörített csomagokban továbbítódnak, és ha a járműhöz egyetlen vezérlőkliens sem kapcsolódik, a kliens nem közvetíti a kameraképet a szervernek. Az adatforgalommal való takarékosságnak az a fő oka, hogy a mai mobilinternet-tarifák korlátozzák az adatforgalmat.

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 hídrozni lehessen, a felhasználónak meg kell adnia a híd pontos címét és a tanúsítványállományok helyét. Ezenkívül további hasznos beállítási lehetőségek is elérhetők a főablakból előhozható Beállítások ablakból.



26. ábra: Járműkliens-beállítások 1.

Mivel az IP Webcam szolgáltatása belső hálózaton bármelyik webböngészőből elérhető, felhasználónével és jelszóval védhető a videószervert. Ezeket az adatokat a vezérlőkliensnek is meg kell adni. Megadható emellett az is, hogy a szerver melyik kapun figyeljen. Ha a szervert a vezérlőkliens indítja el, akkor az abban megadott beállítások lesznek érvényesek akkor is, ha az IP Webcam konfigurációjában más szerepel.

Az akkumulátorszint százalékos megbecsüléséhez szükség van a feszültséghatárok megadására. A maximális feszültség az a feszültség, amely az akkumulátorok teljes töltöttségekor olvasható le a főablakról. A minimális feszültség pedig az a feszültség, amely még elég a jármű motorjának meghajtására.

A program több jármű kezelésére is fel van készítve. Az egyszerűbb járműveknek – például az általam elkészített prototípusnak – nem elég erős a motorjuk ahhoz, hogy érdemes lenne őket precízen irányítani, de vannak gyors járművek is, és azok irányításában már felmerülhet a gázadagolás lehetősége. A járműválasztóban most csak a *Prototípus* és a *PWM teszt* opciók közül lehet választani; ez utóbbi választásával megszabható, milyen legyen a jármű sebessége, ha egérrel irányítjuk.

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

Ha frissítési időközt adunk meg, azzal takarékoskodni lehet az adatforgalommal. Ez megint csak mobilinternet használatakor lehet hasznos funkció. Ha az érték nulla, akkor mielőtt változnak a szenzoradatok, nyomban ki is küldődnek. Az értéket ezredmásodpercben kell megadni, ha tehát ez az érték 1000, akkor minden másodpercben egyszer frissülnek a szenzoradatok – s velük a jármű helyzete és iránya.

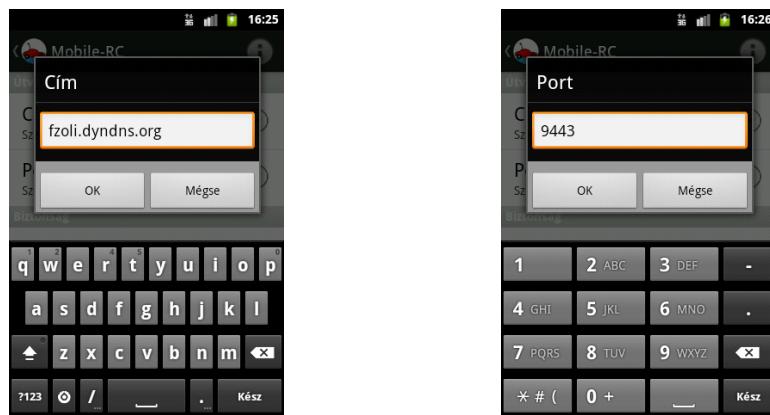
Ha a kapcsolat megszakad, akkor a kliens megpróbál újra kapcsolódni a szerverhez. Hogy a kapcsolódási kísérletek ne terheljék a hálózati forgalmat és a processzort, a kapcsolat megszakadása után nem azonnal indul meg az újrakapcsolódás. A *Kapcsolódás késleltetés* opció beállításával megszabható, hogy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

Az alkalmazás alapértelmezés szerint meg sem próbál kapcsolódni a szerverhez, ha nem észlel aktív hálózati kapcsolatot. A hálózatok működési elvében jártasabb felhasználók viszont használhatnak olyan alternatív módszereket, amelyeket a program nem detektál. A *Kényszerített kapcsolódás* opció aktiválásával kikapcsolhatják a hálózat elérhetőségének ellenőrzését. Például ha

mobilhálózatot vagy Wi-Fi hálózatot használnak, akkor ez az opció kikapcsolva maradhat, de ha, mondjuk, csak az internetmegosztást használják és laptoppal kapcsolódnak a telefonhoz, akkor be kell kapcsolniuk ezt az opciót.

A jármű irányításához a régebbi Android rendszereken szükség van az USB hibakeresés bekapcsolására; azt a fejlesztők röviden ADB-nek nevezik. Az ADB eredetileg hasznos hibakereső interfész; használatával azonban a teljes rendszer konfigurálható, így a jármű is vezérelhető a közvetítésével. Az ADB nem kis potenciális veszéllyel jár, ha állandóan be van kapcsolva, mert a hozzáértő könnyen megszerezheti a felhasználó személyes adatait, jelszavait. Az ADB-t ezért egyes-egyedül rendszeralkalmazások kapcsolhatják be. Az én alkalmazásomból tehát nem lehet bekapcsolni az USB hibakeresést, mert nem rendszeralkalmazás. Létezik azonban egy AdbToggle nevezetű program, s az rendszeralkalmazásként telepíthető rootolt rendszereken. Alkalmazásom együttműködik ezzel az alkalmazással, és ha telepítve van, induláskor automatikusan bekapcsolja az ADB-t – ha szükséges –, és a leállásakor le is állítja azt. A felhasználó az *ADB bekapcsolva marad* opcióval megadhatja, hogy bekapcsolva akarja a hagyni az USB hibakeresést az alkalmazás leállítása után, de ez az opció csak akkor érhető el, ha az AdbToggle telepítve van. Ha nincs telepítve és az ADB ki van kapcsolva, akkor az alkalmazás az indulásakor figyelmezteti a felhasználót.

Az alkalmazás *offline üzemmódban* is működhet. Offline üzemmódban a szolgáltatás indításakor a program nem kapcsolódik a hídszerverhez, csak a jármű áramköréhez. Ebben az üzemmódban a jármű könnyen tesztelhető. A főablakon található irányító panellel pontosan vezérelhető a jármű. Ha tehát a felhasználó tréfás kedvében van, akkor „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

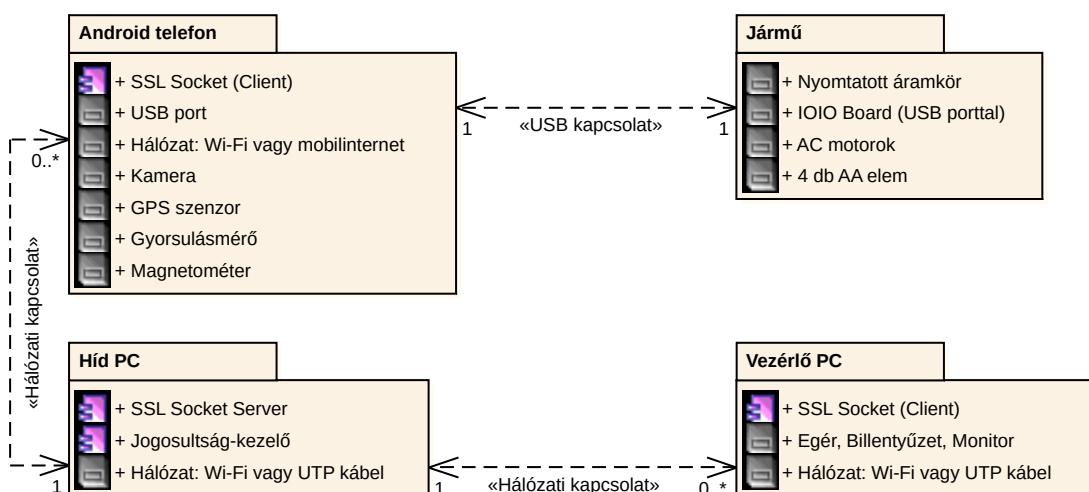
Sokat gondolkodtam azon, hogyan adhatnám át diplomamunkám működésének elvét a legérhetőbb az olvasónak. Végül arra jutottam, hogy leírom, hogyan terveztem meg lépésről lépésre az alkalmazásokat. Ebben a fejezetben felvázolom az alapötletet, majd mélyebb szinten kibontom a részeit, és azt is leírom, hogyan változtak az elkapcsolások, milyen lehetőségek közül választhattam, és végül hogyan választottam.

3.1. Eszközválasztás

Elsőre azt kellett eldöntenem, hogy milyen eszközöket fogok felhasználni a kivitelezéshez. A jármű adva volt – 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 ez volt az általam elsőnek tanult objektumorientált nyelv, nem nehéz rajta programot írnom. A diákműhely arra is kedvet adott, hogy vegyek egy okostelefont, és mivel már telefonom is volt, az Apple okostelefonjai nem is jöttek számításba.

Az elején nem volt egyértelmű, hogyan fogom a telefont összeköttetésbe hozni az áramkörrel. Választhattam az USB, illetve a Bluetooth alapú kommunikáció között. Hamar eldöntöttem, hogy inkább USB kábelt fogok használni, mert egyszerűbb, megbízhatóbb és jobban védve a támadásoktól is, de mivel ez a terület teljesen új volt nekem, utána kellett járnom, hogyan vezérelhetem az USB kapun át az áramkört, és azt is ki kellett derítenem, hogyan mozgathatom a motorokat minden irányban.

Az interneten két olyan elektronikai fejlesztőplatformot találtam, amely alkalmas USB kapcsolaton át zajló kommunikációra, és Android alapú telefonokkal is működik. Az első találatom az Arduino^a IDE volt, a második a IOIO for Android^b. Végül a IOIO Board mellett döntöttem, mivel azt egyenesen az Android telefonokhoz terveztek, és nem kell programozni a mikrovezérlőt.



28. ábra: Eszközök és kapcsolatuk

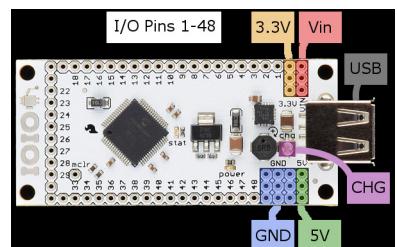
Az eszközválasztás után egy az ábrához nagyon hasonló terv állt össze a fejemben. A bevezetésben megfogalmazott észrevételeimet is ekkor írtam le. Tudtam, hogy a telefon szenzorait is fel fogom használni helymeghatározásra, és azt is, hogy három alkalmazást fogok készíteni és a két klienst összekötő szerver végzi majd fogja a teljes jogkezelést. Meg voltam róla győződve, hogy valóra is válthatom a tervemet, de mivel korábban csak szoftverfejlesztéssel foglalkoztam, azért eljött az ideje, hogy az áramkörtervezésben is mélyebbre ássak.

3.2. Az áramkör megalkotása

Miután eldöntöttem, hogy három alkalmazást fogok létrehozni, és azt is, hogy azok mely eszközökön fognak futni, kis időre félretettem a szoftverfejlesztést, és az elektronikai részt vettetem elő. Azért döntöttem így, mert kliens–szerver alapú alkalmazásokat már többször is írtam, és biztos voltam benne, hogy a programozással nem lesz baj, de abban már nem voltam ennyire biztos, hogy az áramkört is meg tudom építeni. Úgy gondoltam, hogy ha még sem sikerül, még választhatok egy másik diplomamunka-témakört, és viszonylag kevés idő fog rámenni erre a próbálkozásra, de azért reménykedtem a sikerben. Mivel a IOIO Board csak Amerikából szerezhető be, hogy még gyorsabban meglegyen, Interneten megrendeltem – hogy mire az áramkör terve elkészül, már a kezemben legyen az eszköz.

3.2.1. IOIO for Android

A IOIO-t (angol kiejtéssel yo-yo) az Android 1.5 és az afeletti rendszerekhez terveztek. Gyártója a SparkFun Electronics. Alapértelmezésben USB kapcsolaton át kommunikál, de kiegészítő eszközzel Bluetooth hálózaton át is kapcsolódhat a telefonhoz. Vezérléséhez nem kell programozni a mikrovezérlőjét. A gyártó oldaláról letölthető egy könnyen használható Java API; azt csak hozzá kell adni az Android alkalmazás projektjéhez, és attól fogva Java nyelven lehet utasításokat adni a IOIO-nak.



29. ábra: IOIO Board

Az eszközön 48 vezérelhető tű (pin) van, nagy részük digitális ki- és bemenet aszerint, hogy milyen célra használjuk fel, de a 31-től 46-ig számozott tűk csak analóg bemenetként működtethető. Az eszköznek nincs analóg kimenete, de együttműködik a PWM-mel a digitális kimeneteken. A PWM-ről – azaz az impulzusszélesség-modulációról – később még lesz szó.

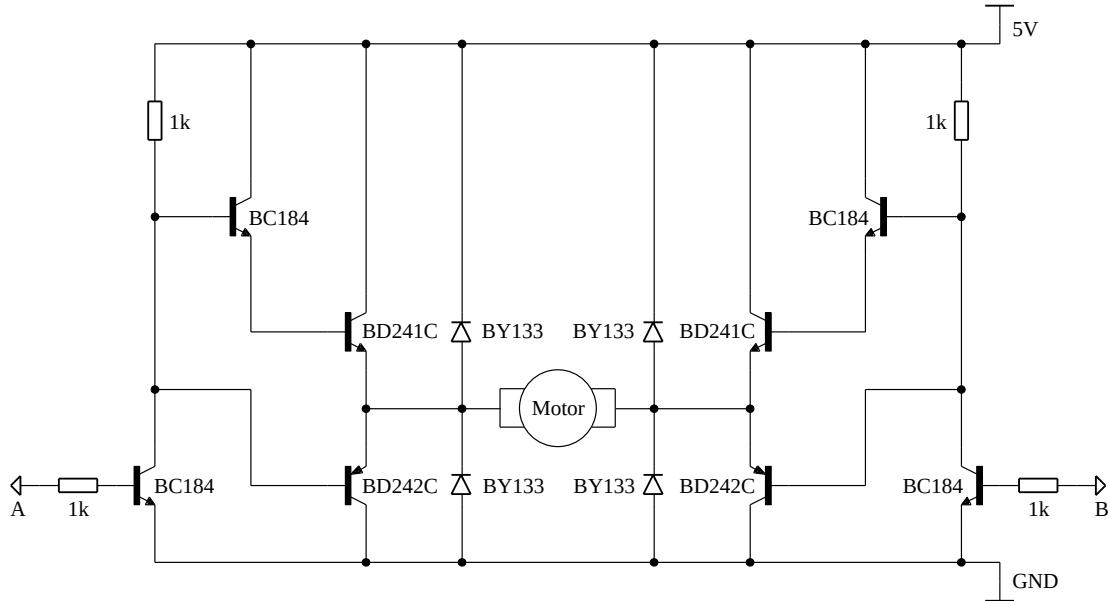
A telefonnal az eszköz az ADB-n (Android Debug Interface) át kommunikál – már ha az USB hibakeresés engedélyezve van. Azokon a rendszereken, melyek ismerik az Open Accessory protokollt (vagyis Android 2.3.4-től), nincs szükség az ADB engedélyezésére.

3.2.2. Kezdeti lépések

Korábban már említettem, hogy még soha nem terveztem áramkört önállóan, konkrét céllal. Szerencsére volt egy ismerősöm, aki otthon volt ebben a téma körben, és tőle kérhettem útmutatást. Az első és legfontosabb kérdésem az volt – ismervén a IOIO eszköz Java nyelvre írt API-ját –, hogy hogyan használhatok két digitális kimenetet úgy, hogy egy egyenáramú motort minden irányba megindíthassák. Ahogyan én keresek megoldást konkrét problémára programozás közben, ő is keresett és talált az interneten egy olyan áramköri rajzot^c, amelyet kezdők is megépíthetnek, mégis megbízhatóan működik.

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

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



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

Az ábrán látható, hogy az áramkör főként bipoláris tranzisztorokból áll. Nekem itt az volt nehéz, hogy a BC184 NPN-tranzisztor lábai pontosan C-B-E sorrendben helyezkednek el, ahogyan az ábrán is, a BD241C és BD242C tranzisztorok lábai viszont B-C-E sorrendben követik egymást, vagyis a kollektor és a bázis fel vannak cserélve. A másik, ami megzavart: az, hogy a BD242C tranzisztor nem NPN, hanem PNP típusú tranzisztor, a specifikációban mégis ugyanabban a sorrendben vannak felsorolva a lábai, mint az NPN típusú társaié, a BD241C tranzisztoránál. A diódákon szerencsére vonallal egyértelműen meg volt jelölve, 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 – hiszen teljesen mindegy, hogy az ellenállásokat milyen irányba kötjük be.

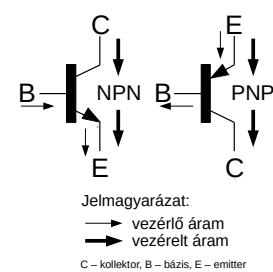
3.2.3. Az áramkör felhasználása

Ahhoz, hogy egy egyenáramú motort erre vagy arra meg lehessen hajtani, tulajdonképpen a motorra eső résznél kell szabályoznunk a feszültség polaritását. A BJT H-híd nevű áramkör ezt bipoláris tranzisztorokkal (Bipolar Junction Transistor) éri el, innen is származik a neve.

A szoftverfejlesztőknek mellékelek egy képet, hogy érhetőbbek legyenek az előző részben emlegetett tranzisztorok lábaira és típusára vonatkozó észrevételek.

Ezeket a tranzisztorokat erősítőkben, szabályzó- és kapcsolóáramkörökben használják. A BJT három kivezetésű elektronikai alkatrész. Három elektromosan szétválasztott rétege van, innen származik az NPN és a PNP megnevezés.

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



31. ábra: Bipoláris tranzisztor

A feszültség polaritását az áramkör szélein található két digitális feszültségforrás (A és B) szabályozza. Ha egyik feszültségforrás se aktív, akkor a motor áll; ha a kettő közül az egyik aktív, akkor a motor forog az egyik, illetve a másik irányba aszerint, hogy melyik feszültségforrás aktív. Az áramkör zárlat ellen is védve van, ha tehát minden kettő feszültségforrás aktív, akkor is áll a motor – de nem tanácsos a tűzzel játszani.

3.2.4. A nyomtatott áramkör megtervezése

3.2.4.1. Az első lépések

A tranzisztorok lábainak eltérő sorrendje azzal a következménnyel járt, hogy az áramkört nem építhettem meg az eredeti terv szerint, mert akkor sok helyen kellett volna áthidalni a vezetéseket. A próbapanelen az összekötőkábeleket úgy rendezhettem, ahogyan akartam, s ez ott még nem is okozott gondot, de a végleges nyomtatott áramkörön (NyÁK) az áthidalás már nem lett volna szép megoldás.

Megrajzoltam egy olyan áramkört, amely megfelelt az eredeti áramkörnek, de már a felhasznált elektronikai alkatrészek kivezetései követték rajta egymást a vezetőrétegek. Az eredeti áramkör szimmetrikus felépítését is megtartottam, és a könnyebb összeszerelhetőség kedvéért 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 hogy az IOIO eszközt is helytakarékosan kell elhelyeznem 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, s hogy azzal lehessen be- és kikapcsolni őket.

3.2.4.2. NyÁK-tervező program választása

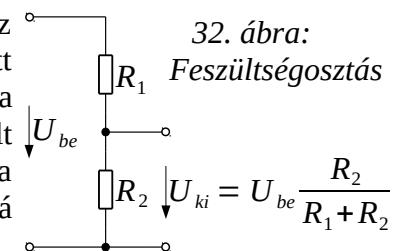
Hogy a nyomtatott áramkört egyszerű legyen megtervezni, kerestem egy Linuxon is futó NyÁK-tervező programot. Több alkalmazást is találtam, de legjobbnak az EAGLE bizonyult, a CadSoft-tól. Ebben az alkalmazásban az tetszett meg, hogy minden alkatrész méretarányos benne, és lehetővé teszi a PDF-be exportálást. PDF-állományra azért volt szükségem, mert nincs lézernyomtatóm, a NyÁK-ra pedig csak lézernyomtatával készült terv vasalható rá. A PDF megtartja a méretarányokat, és a boltban, ahol nyomtatnak lézernyomtatával, van a gépre PDF-nézegető telepítve.

Ettől a ponttól kezdve tudtam, hogy meg fogom tudni építeni az áramköri részt, és amíg várakoztam a postára – hogy meghozza az IOIO eszközt –, elkezdtem a három alkalmazás tervezését és fejlesztését. A csomag végül megérkezett; lemérhettem az IOIO és a kivezetések pontos méretét, s nekiláttam a végleges NyÁK megtervezésének.

3.2.4.3. A NyÁK kialakulása

Hogy minél kisebb legyen a helyfoglalás, a két H-híd egymás alá került, 180 fokkal elforgatva egymáshoz képest, hogy az IOIO digitális kimeneteit egyszerű legyen beilleszteni – és így a tápellátást is elvezethettem az egyik ellenállás kivezetései között.

Elkészülttem a két H-híd és az IOIO összekötésével, valamint az áramellátással, s már csak az akkumulátorszint mérését kellett belevereznem az áramkörbe. Az IOIO-nak van analóg bemenete, de a dokumentációban^d külön hangsúlyozták, hogy ha a bemenetre 3,3 volt feszültségnél több kerül, akkor az eszköz meghibásodhat. Újra találkoztam az ismerőssémmel, és megkérdeztem tőle, hogyan oldaná meg a feszültségmérést, az eszköz épségére is ügyelve.



32. ábra:
Feszültségesztás

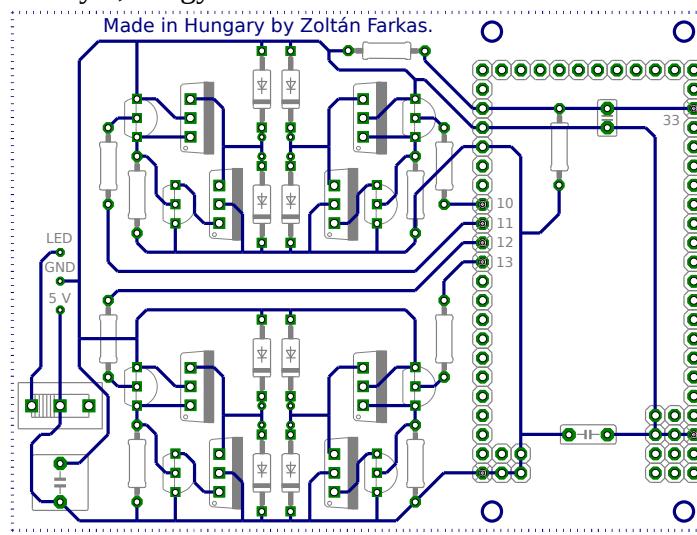
Két lehetőség merült fel. Zener-dióda és ellenállások használatával megoldható, hogy eltérő feszültségforrások használatakor se menjen a feszültség 3,3 volt fölé a bemeneten; de ha a tápellátás nem változik, akkor a feszültségesztás^e is alkalmazható. Mivel a járművet csak négy nikkel-cink (NiZn) akkumulátorral táplálom, a feszültség soha nem megy 4-szer 1,6 volt, azaz

6,4 volt fölé, tehát a két ellenállás, nincs szükség feszültségstabilizáló Zener-diódára.

Ismerősöm tanácsára az áramkörön elhelyeztem egy elektrolitkondenzátort és két kerámiakondenzátort. Az egyik kerámiakondenzátorra azért volt szükség, mert nagyon ingadozott a mért feszültség, és a kondenzátor rövid távú energiatárolása nagyban csökkentette ezt az ingadozást. A másik két kondenzátor biztonsági célból került az áramkörbe, az IOIO tápellátásának segítésére.

Az IOIO áramkörhöz való kapcsolását nem közvetlenre terveztem, hogy bármikor levehessem a NyÁK-ról és más projektekben is felhasználhassam, ezért tüskesort is felhasználtam. Ez azért is fontos, mert az IOIO alá így befért több alkatrész is, egész pontosan egy ellenállás és a két kerámiakondenzátor. Az elektrolitkondenzátort a kapcsoló mellé tettek, az áramkör másik végére, mivel csak ott volt neki elegendő hely.

A kondenzátorok elhelyezésével és bekötésével a nyomtatott áramkör megkapta végső formáját. A panelen megjelöltetem az IOIO rögzítési pontjait; maradt még egy kis szabad hely, s ott feltüntettem a gyártás helyét, ahogyan szokás.



33. ábra: A nyomtatott áramkör

A tervezés végeztével az egész áramkört kilenc példányban feltettem egy A4-es lapra, és úgy állítottam be az EAGLE-t, hogy csak a vezetéseket és a furatokat jelenítse meg. Ez után fekete-fehérre PDF-állományba exportáltam a lapot, azt lézernyomtatóval kinyomtattattam, és megkezdtettem életem első nyomtatott áramkörének elkészítését.

3.2.5. A nyomtatott áramkör kivitelezése

A NyÁK elkészítésének több útja is van. A legegyszerűbb a kézzel való megrajzolás, de ez időigényes, mert minden kézzel kell kimérni, hogy pontosan a helyén legyen minden furat. Az áramkör tervezése közben arra gondoltam, hogy ha már úgy is készítek egy méretarányos, teljes tervet, nem vihetném-e át ezt a tervet otthoni körülmények a panelre. Két elkészítési módszert találtam: a vasalásos^f és a fototechnikás^g NyÁK-készítést.

3.2.5.1. Fototechnikás NyÁK-készítés

Ebben az eljárásban nem hagyományos, hanem fényérzékeny lakkal bevont nyáklap használatos. Ebben a fényérzékeny rétegben ibolyántúli fény hatására olyan kémiai reakciók indulnak meg, amelyek révén ez a lakkréteg lúgban oldható lesz. Ahol a lakkot nem éri ibolyántúli fény, ott a lakk ellenáll a lúgnak. Az előre megtervezett nyáktervet átlátszó fóliára nyomtatjuk, majd a fényérzékeny nyáklap elé helyezzük. A lézernyomtató festéke ellenáll az ibolyántúli sugaraknak, s a mögötte levő nyáklfelület ezáltal védve van a fénytől. Így lényegében a kinyomtatott nyáktervet a rávilágítjuk nyákra. Ahol ibolyántúli fény érte a nyákot, ott le lehet majd mosni a védőlakkot; ahol nem érte

fény, ott ellenálló marad. Az oldható lakkot lúgoldatban lemossuk, így ott tiszta rézfelület alakul ki, s azt a maratoszer lemarja. A védett réteg pedig megmarad: az lesz maga a nyákrajz.

3.2.5.2. Vasalásos NyÁK-készítés

Véleményem szerint ez az eljárás biztonságosabb és egyszerűbb a fototechnikás eljárásnál. Biztonságosabb, mert nincs szükség ibolyántúli fényre, a szem tehát nincs sugárzásnak kitéve. S egyszerűbb is, mert nincs szükség lúgoldatra, se átlátszó fóliára, csak egy sima talpú vasalóra és egy fényezett fotópapírra. Mindezek miatt én a vasalásos módszert választottam.

Ezzel az eljárással a lézernyomtató tintáját égetjük rá a nyáklapra, majd azt a lapot, amelyre az áramkör nyomtatva volt, leáztatjuk a nyáklapról. A nyáktervet tükrözve kell kinyomtatni, mivel a kinyomtatott lapot nyilván a tintás oldalával lefelé vasaljuk rá a nyáklapra. A nyomtatáshoz a legtöbb 135 grammos műnyomópapírt ajánlanak, bár fotópapírt is lehet használni; a lényeg az, hogy a lap tömege 130 és 150 g között legyen, mert akkor könnyen vasalható és áztatható. Fontos, hogy a tervet fényes felületű lapra nyomtassuk, hogy a lap ne szívja magába a lézernyomtató festékét, és hogy lézernyomtatót használunk, mert a tintasugaras nyomtatónak víz alapú tintája soha nem fog a vasaló hőjétől megolvadni, majd átfolyni a nyákra. A lap fényes felét nem tanácsos kézzel megfogni, és víz alatt finom csiszolópapírral a nyáklapot is meg kell tisztítani még a vasalás előtt – ezzel a tinta nyákhoz való tapadását segítjük elő.

A tintát vasalóval égetjük rá a nyákra, de egyszerűbb laminálógéppel, ha van. A megfelelő hőmérséklet 200 °C és 220 °C közé esik, de ez változhat a papírlap vastagsága szerint. A vasalás akkor fejeződik be, ha az áramkör már a lap másik oldaláról is látható; ezután már csak le kell áztatni a papírlapot, majd középről kifelé haladva óvatosan le is kell szedni. A papírlap leszedése után láthatjuk, hogy a teljes nyákrajz felkerült a nyákra.

3.2.5.3. A NyÁK maratása

Ha az előbbi bemutatott módszerek valamelyikével elkészült a nyákrajz, akkor a rézréteg lemaratása következik. A maratáshoz vas-kloridos maratóoldatot használtam. Kerestem a háztartásban egy épp akkora műanyag edényt, amekkora a nyák – hogy ne kelljen a szükségesnél több vas-kloridot felhasználnom. Fém alapú edényt nem használhattam, mert a maratoszer azt is megtámadta volna.

Az edénybe beletettem a nyákat, és annyi vas-klorid oldatot öntöttem rá, hogy már ellepje. A maratás folyamatának gyorsítására az edényt forró vízbe nyomtam, és a két szélét fel- s lemozgattam, hogy az oldat is mozogjon. Az edény mozgatásával azt is láttam, hogy hol tart a maratás, és így még idejében kivehettem a nyákat, nehogy a vas-klorid marni kezdje a tinta alapú védőréteg széleit is.

A maratás után lakkbenzin-hígítóval eltávolítottam a nyákról a festéket, majd a nyákat vízzel lemostam. Hogy a vezetőréteget védjem a korroziótól, a nyáklapot kémiai ónozó oldatba merítettem, s az matt ónfelülettel vonta be a rézfelületet.

3.2.5.4. A NyÁK befejezése

A maratás után kialakultak a vezetőrétegek, s megvoltak a furathelyek is; következhetett tehát a furatok elkészítése. Az itthoni fúrógépek egyike sem volt alkalmas 1 milliméternél kisebb átmérőjű fúrófejek befogására, ezért szinte az összes furat 1 milliméter átmérőjű lett.

A furatok elkészülése után jöhett az elektronikai alkatrészek beforrasztása. Otthon volt egy forrasztópisztoly, s az meg is felelt, de a forrasztásban nem volt semmi gyakorlatom, ezért elég sok forrasztóhegyet tettem tönkre. A forrasztást az is nehezítette, hogy túl vékony vezetéseket terveztem, és nemely helyen feljött a rézréteg. Az áramkör vezetését multiméterrel ellenőriztem, és addig javítottam, amíg elég nagy nem lett. Végül elkészült tehát 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 hozzácsavarozhassam a járműhöz. A nyák rögzítése előtt az eredeti nyákot persze eltávolítottam, és a kábeleit felhasználtam a motorok bekötésére. A járművön levő kapcsoló kábelén át láttam el a nyákot árammal, és a LED-ek tápellátását a kapcsolóról áthelyeztem saját áramköröm kapcsolójának kivezetésére.

Mire a nyomtatott áramkörrel elkészülvtem, az Androidra írt alkalmazásom is olyan szintre jutott, hogy már tesztelhettem vele az áramkör működését. Ezt a funkciót meg is hagytam benne – és elneveztem Offline módnak. Az áramkör megfelelően műköött, így az elektronikai résszel el is készülttem; folytathattam a szoftverfejlesztést.

3.3. A szoftverek megalkotása

Általában egyszerre szoktam írni a logikai tervet és a program forráskódját, éspedig azért, mert valójában minden fejben tervezek meg. Ebben sokat segít az objektumorientált tervezés, mert nagyon jól részekre bonthatom az alkalmazást, és így minden csak az éppen fontos részre koncentrálhatok.

A legelején eldöntöm, mit is fog csinálni a program, és hogy ahhoz mi szükséges. Ez alapján kigondolok egy általános tervet. Ekkor dől el, milyen részekből fog összeállni a program, és ezután megyek mélyebben bele a részekbe.

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

3.3.1. A hálózati kommunikáció felépítése

A szoftverfejlesztést a hálózati kommunikáció megtervezésével kezdtem. Mivel mindenkor alkalmazás Java nyelven íródott, azért a hálózati kommunikáció az egyetlen közös rész a három alkalmazásban, egyszersmind a legfontosabb is, mivel az alkalmazások használhatósága nagyban függ a hálózat képességeiről.

Régebben már foglalkoztam kliens–szerver architektúrával Java alkalmazásokban, így könnyen eldönthettem, hogy nem veszek igénybe külső programkönyvtárat, például az Apache HttpClient nevű projektjét, mert jelen esetben a HTTP protokoll használata nem lenne kifizetődő.

A hálózaton át egyszerre zajlik a kamerakép és az üzenetek közvetítése. Mindez valós időben zajlik, 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 Javában

A Java SDK eleve támogatja a socketek használatát. A szerver szerepét betöltő alkalmazás használja a *ServerSocket* nevű osztályt; egy ilyen osztályú objektum létrehozásához elegendő megadni a konstruktőrben egy kaput: azt fogja lefoglalni magának az alkalmazás. A kliensalkalmazásban a kapcsolat létrehozásához a *Socket* nevű osztályt kell példányosítani, s ahhoz két paraméter szükséges: cím és kapu. Mihelyt egy kliens kapcsolódik a szerverhez, a szerveroldalon a *ServerSocket accept()* metódusa visszatér egy *Socket* objektummal, az áll összeköttetésben a kliensoldal *Socket* objektumával. A *Socket* objektumtól elérhető a kimenő és a bejövő adatfolyam referenciája. Ettől a ponttól kezdve pontosan úgy használható az *InputStream* és az *OutputStream* objektum, mint az állománykezelésben.

3.3.1.2. Eszköz- és kapcsolataazonosító

Ahhoz, hogy a hálózaton egy időben többféle információt lehessen közvetíteni, a klienseknek nem elég egy kapcsolatot létrehozniuk a szerverrel. Az MJPEG folyam és az üzenetobjektumok tehát külön csatornán közvetítendők. Ez nyomban felvet egy kérdést: honnan fogja tudni a szerver, hogy mi célból kapcsolódott hozzá egy kliens?

Az egyik megoldás lehetne a különböző kapu használata. Ez esetben két kapu lenne a szerveren lefoglalva, és két ServerSocket objektumot használnánk. A kliens célját abból lehetne megtudni, hogy melyikhez kapcsolódott. Ezt a megoldást elvettem, mert szerettem volna úgy megoldani a feladatot, hogy a szerver a takarékkosságra és a könnyebb kezelhetőségre való tekintettel csak egy kaput (portot) foglaljon le.

Azt találtam ki, hogy kapcsolatazonosítót vezetek be; azt a kliens küldi el a szervernek aszerint, hogy milyen céllal lép vele kapcsolatba. minden kapcsolatfelvétel úgy történik tehát, hogy a kliens elküldi és a szerver fogadja az első bájtot; az a kapcsolatazonosító, és annak ismeretében cselekednek. Az azonosítót küldhetné éppen a szerver is aszerint, hogy a kliens melyik kapcsolatazonosítót nem vette még használatba, de a kliensoldal jobb választásnak tűnt, mert kevesebb az adminisztráció.

A kapcsolatazonosítóból a szerver tehát már megtudhatja a kapcsolódás célját, de azt még nem, hogy vajon melyik alkalmazás kapcsolódott hozzá. A kapcsolatazonosító erre is fel lehetett volna használni (csak eltérő azonosító kellett volna), de a könnyebb átláthatóság kedvéért szerettem volna, hogy a két kliensalkalmazás ugyanazt az azonosítót használja, ha ugyanaz a céljuk. Például ha a járműkliens és a vezérlőkliens is a kameraképet akarja kezelní, akkor használják ugyanazt az azonosítót.

Ennek a problémának a megoldására vezettem be az eszközazonosítót; azt is a kliens küldi a szervernek, még a kapcsolatazonosító előtt. Így végül a második bájt lett a kapcsolatazonosító, és az első bájt az eszközazonosító.

Itt még mindig nincs vége a dolegnek. A szerver most már tudja az eszköz típusát és a kapcsolatteremtés célját, de még nem különítheti el egymástól az alkalmazásokat, vagyis nem tudja, hogy ki kapcsolódott hozzá. Emiatt is be kellett vezetni a felhasználóazonosítást. A felhasználónév alapján már elkülöníthetők egymástól az alkalmazások, mivel a rendszert úgy alakítottam ki, hogy egy időben csak egy alkalmazás használhatja ezt vagy azt a felhasználónevét.

3.3.1.3. Titkosított kapcsolat és felhasználóazonosítás

Manapság az olcsó kategóriájú okostelefonokat egészen jó áron adják, de még így is több tízezer forintba kerülnek, célszerű tehát vigyázni rájuk. Éppen ezért úgy döntöttem, hogy a kapcsolatot titkosítom és csak azokat a klienseket engedem fel a szerverre, amelyeknek van általam kiállított tanúsítványuk; ezzel kerülök el a jogtalan hozzáférést vagy az alkalmazás megkerülését.

Elsőre azt gondoltam, hogy felhasználónév-jelszó párral azonosíthatnám a felhasználókat, csak hogy titkosítatlan kapcsolaton könnyen elfogható a jelszó; ha viszont már titkosított a kapcsolat, akkor nincs értelme felhasználónevét kérni, mivel a tanúsítvány tartalmazhatja azt a CN mezőben; azt webszerverek esetén a doménnévre tartják fenn. Én felhasználónévnek használom.

Attól sem kell tartanom, hogy a tanúsítvány CN mezejét valaki megmásítja és más felhasználó nevénben kapcsolódik a szerverhez, mivel a kiállított tanúsítvány alá van írva, és az aláírás érvényét veszti, ha az állományt utólag módosítják; aláírni pedig csak a kiállító tud, feltéve ha csak neki van birtokában a titkos kulcs. Jelszóval védekkedhetők a titkos kulcsok is kiállíthatók, s ha megszerzi is valaki a tanúsítványállományokat, akkor sem használja 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 kérem 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, amelynek a kiállítója nem szerepel a megbízható tanúsítványok között. Ugyanezen okból keletkezik *SSLHandshakeException* akkor, amikor önmaga által aláírt tanúsítványt használó webszerverhez kapcsolódunk. Ezt a kivételt könnyen el lehet kerülni, ha implementáljuk a *TrustHandler* interfészét, de nálam ez a probléma most fel se merült, csak egy korábbi projektemben.

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

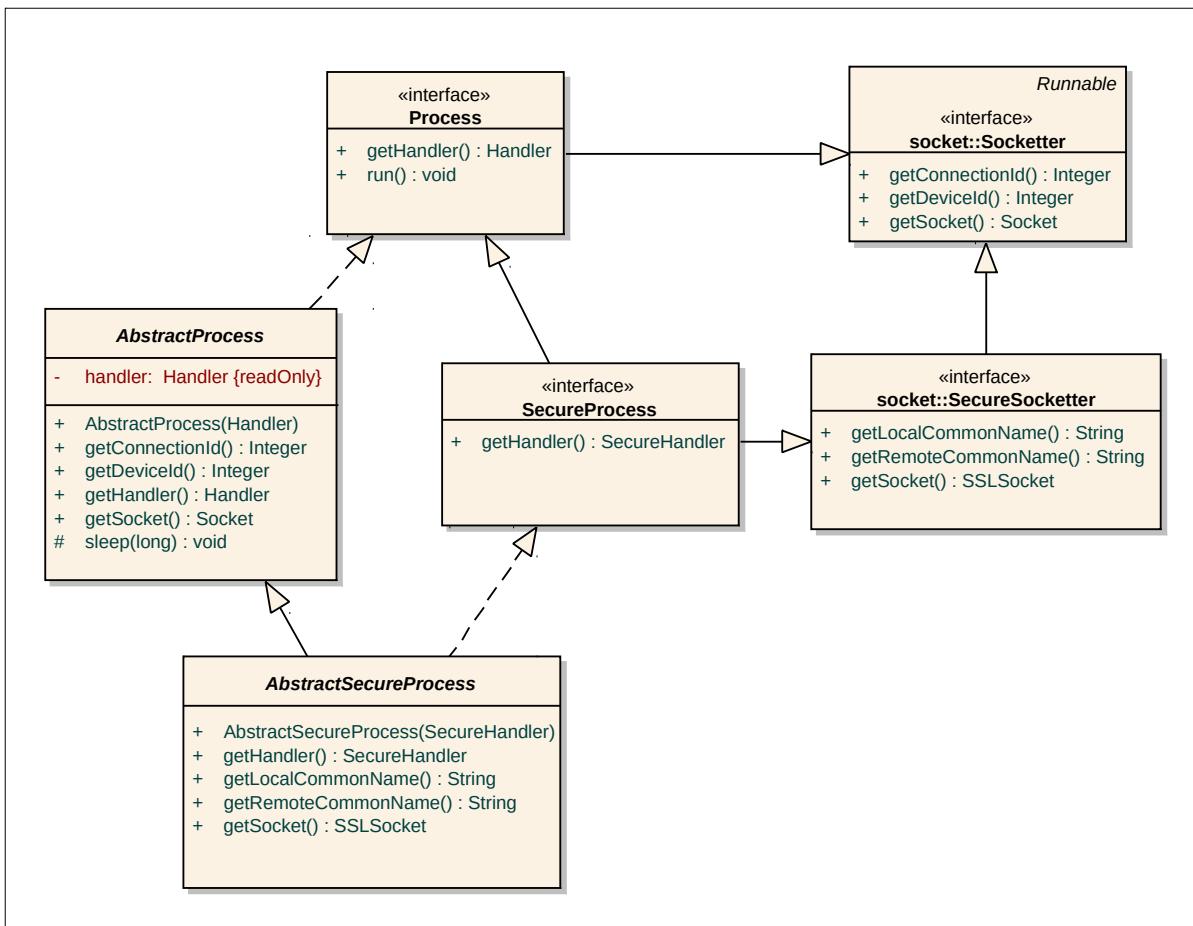
3.3.1.4. Az általános terv

Ahhoz, hogy a szerver egy időben több kapcsolatot kezelhessen, az *accept()* metódust egy előltesztele ciklusban kell folyamatosan futtatni, és mihelyt az visszatér egy *Socket* objektummal, a feldolgozást új szálban kell tovább folytatni, hogy a feldolgozás közben a további kapcsolódó klienseket is lehessen fogadni.

Mindent összevéve – eszközazonosító, kapcsolatazonosító, SSL és szálkezelés alkalmazásával – megalkottam a saját rendszeremet; az 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, ha hibás a kód, pedig tudja az ember, hogy nem kellene annak lennie; végül azután rájön, hogy nem ugyanarról az osztályról van szó. Jobb nevet viszont nem tudtam kitalálni, és egy ponton túl már a *SecureProcess* és *SecureHandler* interfészét használtam, a névegyezés 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 kiértékelése után létrehozni és futtatni a megfelelő *Process* objektumot.

A *Process* is implementálja a *Runnable* interfészét, de csak azért, mert neki is van egy *run()* metódusa; azt a *Handler* hívja meg. Az egyszerűbb felhasználás kedvéért van pár lekérdezőmetódus (getter), azokat a *Handler* értékeli ki, de elérhetők a *Process* osztályban is. A *Process* osztály voltaképpen kapcsolattfeldolgozó. Ezzel gondoskodik arról, hogy más kapcsolatazonosítóhoz más viselkedés tartozzon, és az az eszközazonosító meghatározza a szerepkört.



34. ábra: Process osztálydiagram

A Handler és a Process közös metódusai a `Socketter` interfészbe vannak belefoglalva. Ez tartalmazza az azonosítók getter metódusait és a socket is elérhető tőle, hogy a kommunikáció lehetővé váljon. Titkosított kapcsolat esetén már azt is tudni lehet, hogy ki a helyi és a távoli felhasználó, így a `SecureSocketter` – annak a `Socketter` az ósa – lehetővé teszi ezeknek az adatoknak a lekérését is; a `Socket` típusát `SSLSocket` típusúra pontosítja. Az `AbstractProcess` és az `AbstractSecureProcess` implementálja a getter metódusokat a kezelő (`handler`) objektum adataira támaszkodva, az egyetlen kifejtendő metódus itt a `run()` metódus.

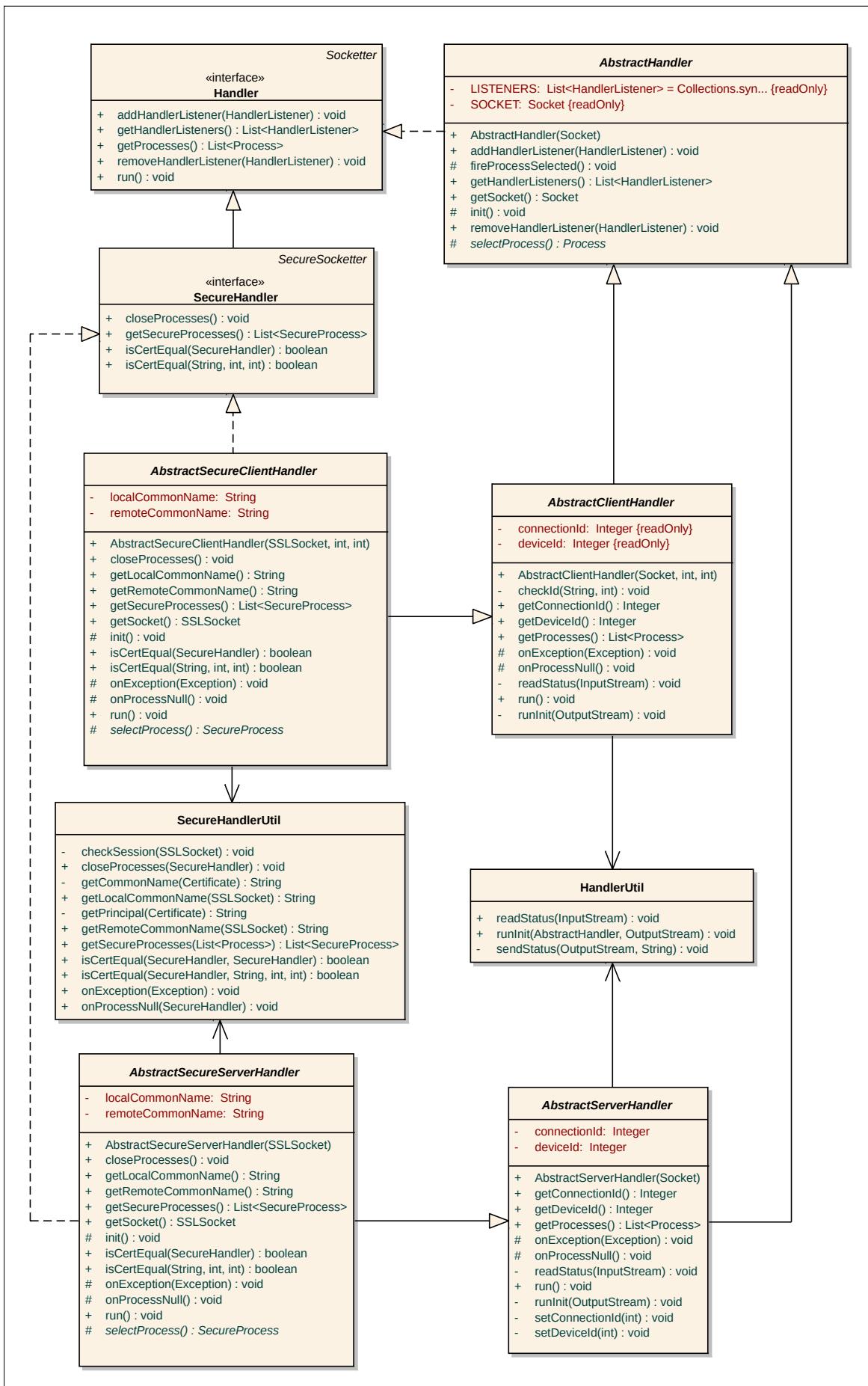
A Handler működtetése valamivel összetettebb folyamat volt. Ennek az az oka, hogy a kliens- és a szerveroldal eltérően viselkedik: a kliens küldi a kapcsolatazonosítókat és a szerver fogadja őket. Ezután már minden fél ugyanúgy működik: létrehozza a megfelelő `Process` objektumot, majd meghívja a `run()` metódust.

A Handler osztálydiagramja a következő oldalra került, mert nagyon nagy.

Az Handler interfések hierarchiája megegyezik a Process interfések hierarchiájával, de az eltérő viselkedés miatt itt két absztrakt osztályt írtam a két oldalhoz. Az `AbstractClientHandler` konstruktora paraméterben várja a két azonosító értékét, az `AbstractServerHandler` viszont nem vár ilyen értéket, mivel ő a kliensoldaltól kapja meg ezeket az adatokat.

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

A Java szándékosan nem támogatja a többszörös öröklést, mert tervezői szerint az több bajjal jár, mint ahányat 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 végett rendszeres időközönként ellenőrizni kell a szerverrel kiépített kapcsolatot. A járműkliens mindenkor az utoljára kiadott parancsot hajtja végre, egy újabb parancs beérkezésig. Ha a kapcsolat megszakad, és az az utolsó parancs az, hogy a jármű menjen egyenesen előre, akkor ellenőrizetlen kapcsolattal a jármű megállíthatatlan lenne, és biztosan beleütközne valamibe.

A kapcsolat instabilitását csak akkor lehet észlelni, ha az egyik oldal ír a kimenő folyamba, a másik oldal meg olvassa a bejövő folyamot. Azon az oldalon, ahol az olvasás történik, instabil vagy megszakadt kapcsolat esetén időtúllépés tapasztalható. Ha mindenkor oldal felváltva ír és olvas, akkor mindenkor oldalon észlelni lehet az időtúllépést.

Ennek megfelelően két *DisconnectProcess* osztályt hoztam létre, az egyiket kliensoldalra, a másikat szerveroldalra. Működésük annyiban tér el, hogy ameddig az egyik ír, addig a másik olvas, és addig csinálják ezt, amíg a kapcsolatot be nem zárul vagy meg nem szakad. Az én esetben azonban nem elég a kapcsolat megszakadását észlelni; azt is észre kell venni, ha pillanatnyi instabilitás van a hálózatban. Emiatt kétrépcsős időtúllépést vezettem be.

A `read()` metódus időtúllépését 1 másodpercre állítottam be, ez felel meg a pillanatnyi instabilitásnak. Időtúllépéskor elindul egy időzítő, s az 10 másodperctől számol vissza; ha ez idő alatt sem érkezik válasz, akkor a program úgy megszakadtnak tekinti a kapcsolatot. Ha 10 másodpercen belül válasz érkezik, akkor az időzítő leáll, a hálózat újra stabilnak tekinthető, és a kapcsolat ellenőrzése tovább folytatódik.

Az első időtúllépéskor eltárolódik a jármű utolsó parancsa, majd a kliens megállítja a járművet. A kapcsolat helyreállásával érvénybe lép az utolsó parancs, és a jármű újra elindul, ha előtte is ment. A jármű sorban megkapja az időtúllépés közben esetleg küldött parancsokat, ha tehát az időtúllépés alatt leállítás parancsot is kapott, akkor megáll.

3.3.1.5.2. Üzenetküldés és -fogadás

A Javában van egy *Serializable* nevű interfész. Az ezt implementáló osztályok objektumai szerializálhatóvá válnak, feltéve, hogy a tulajdonságaik is szerializálhatóak. Objektum állományba való szerializálásakor az objektum összes tulajdonsága tárolódik az állományban. A program leállta után, újból induláskor az állomány tartalmának deserializálásával az objektum visszatölthető a memóriába.

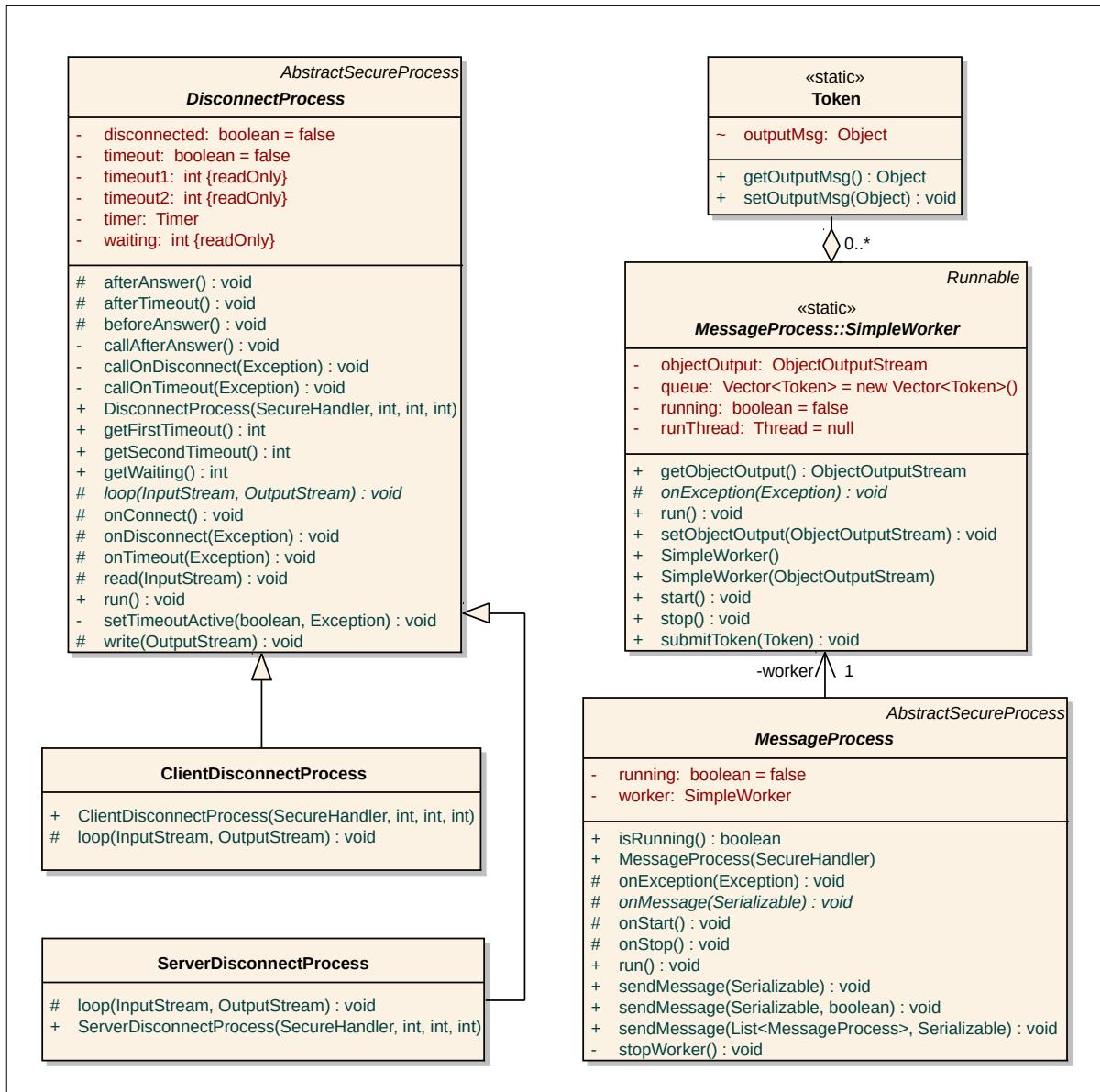
Állományba való szerializáláshoz szükség van egy olyan *FileOutputStream* objektumra, amely fel van ruházva egy konkrét állományba való írás képességével, valamint egy *ObjectOutputStream* nevű objektumra, amely a *FileOutputStream* objektumot felhasználva képes a szerializálásra. Deserializáláskor ugyanez a folyamat zajlik le, csak az írás helyébe olvasás lép, vagyis *FileInputStream* és *ObjectInputStream* objektumra kell támaszkodni.

A Java *ObjectOutputStream* és *ObjectInputStream* osztályát arra használtam fel, hogy üzenetobjektumokat küldjek a socket kapcsolaton keresztül. Az elv ugyanaz, mint az állományba való szerializálás, csak *FileOutputStream* helyett a socket *OutputStream* példányát adom át az *ObjectOutputStream* példányosításakor. A socket másik oldalán megtörténik a deserializálás, s a konkrét objektum egyik alkalmazásból átkerült a másikba.

Ennek az elgondolásnak megfelelően írtam meg a *MessageProcess* nevű osztályt; az a létrejötte után elindít egy szálat, s abban szálkezelt üzenetküldés zajlik, és a szál indítása után ciklusban játszódik le az objektumok fogadása.

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

A szálkezelt üzenetküldésre találtam egy jó megoldást¹ az interneten, és fel is használtam. Az elve viszonylag egyszerű. A küldendő üzenetet Tokenbe kell helyezni, majd megkérni a szálkezelt feldolgozót, hogy küldje el azt. A feldolgozó a tokeneket egy vektorba teszi, és egymás után elküldi őket, mihelyt sorra kerülnek.



36. ábra: A `DisconnectProcess` és a `MessageProcess` osztálydiagramja

3.3.1.5.3. Üzenetobjektumok: adatok, részadatok

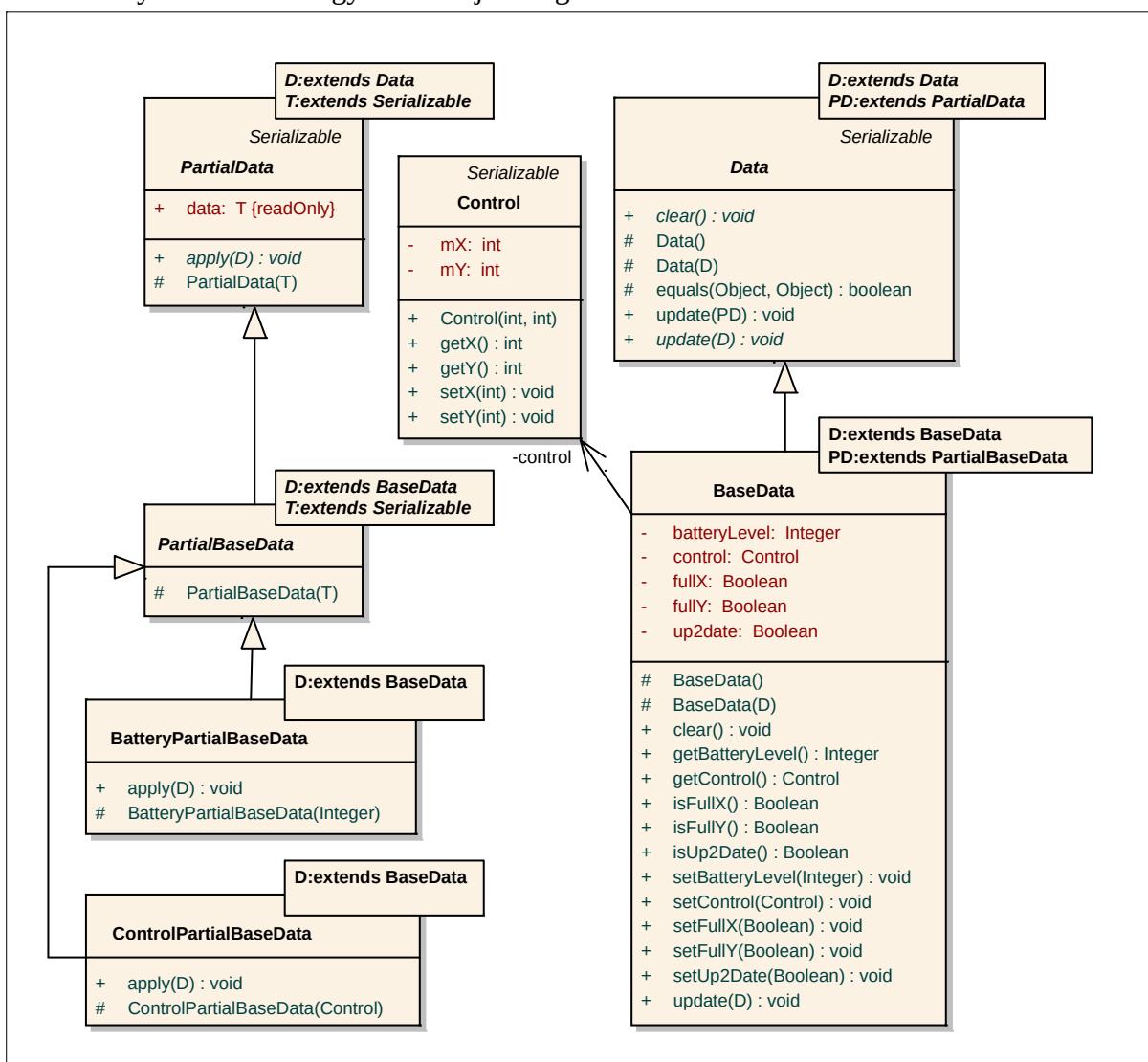
Ebben a fejezetben már szó esett arról, hogyan megy az üzenetküldés, de arról még nem beszéltem, hogy voltaképpen mit küldök át ezen a csatornán.

Az üzeneteket két részre osztottam fel: az egyik részbe a vezérlőkliens és a hídszerver közötti üzenetek tartoznak, a másik részbe a járműkliens és a hídszerver közöttiek.

A szétválasztásra elsősorban azért volt szükség, mert csak azokat az objektumokat lehet deserializálni, amelyeket ismer az alkalmazás, tehát az osztálybetöltőnek el kell tudnia érni ezeknek az osztályoknak a bájtkódját. Másrészt más az információcsere, ha járműkliensről van szó, és más, ha vezérlőkliensről. A vezérlőkliens például küldhet és kaphat csetüzenetet, járművezérlő-kérelmet, és a jármű nyers adatait helyett is csak a neki hasznos, átalakított üzenetet kap. A járműkliens viszont csak nyers szenzoradatokat közvetít, és a jármű vezérlőjelét fogadja.

Valójában az üzenetobjektumok soha sem utasítások, mindig adatmódosulást jeleznek, és az alkalmazások arra reagálnak. Például ha a vezérlőjel megváltozik, akkor a járműkliens módosítja az IOIO digitális kimeneteit, és ezzel megváltozik a jármű viselkedése. Felmerül a kérdés: honnan tudják a kliensek, hogy ez vagy az üzenet milyen adatmódosulást jelez, és hogyan kell azt a módosulást feldolgozni?

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



37. ábra: A *Data* és a *PartialData* osztálydiagramja

Amikor egy kliens kapcsolódik a hídszerverhez, megkapja/elküldi a neki megfelelő adatosztályt, így megvan minden kezdőadata, és a teljes felület frissül. Ettől a ponttól kezdve már csak részadatot kap; a többi adatot fölösleges lenne elküldeni, mert nem minden változik meg.

A Data és a PartialData osztálynak két típusparamétere van. A Data első paramétere D – az határozza meg, hogy milyen adatról van szó. Ez a típusparaméter megegyezik az osztály típusával. A célja az, hogy korlátozza az update(D) metódus használhatóságát.

Hasonló elv szerint íródott meg többek között a Boolean osztály is a Javában:

public final class Boolean implements Comparable<Boolean>

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

Ebből az adódik, hogy Boolean típusú objektum csak Boolean típusú objektummal hasonlítható össze.

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

A PartialData osztálynak is két típusparamétere van. A D paraméter azt adja meg, hogy a részadat pontosan milyen típusú Data osztályhoz tartozik, a T paraméter pedig a megváltozott adat típusát. Itt is igaz, hogy a metódus használata konkrét adattípusra van behatárolva, de itt elsősorban a generikusság előnyét használtam ki, hogy ne kelljen típuskényszerítést használnom az utódosztályokban.

A PartialData update(D) metódusa végzi el ugyanis az adatmódosítást a Data objektumban, s azt paraméterben kell neki átadni. Ezzel a fogással nem kell if–else if szerkezetet használni és minden üzenetre odafigyelni; maga az elküldött üzenetobjektum határozza meg azt is, hogyan kell kezelni az adat módosulását.

Az osztálydiagramon két részadat látható. A BatteryPartialBaseData egy Integer számot tárol – az akkumulátor töltöttség szintjének új értékét százalékban –, és az apply(BaseData) metódusa meghívja a BaseData objektum setBatteryLevel(Integer) metódusát, átadva neki az új értéket. A ControlPartialBaseData a jármű vezérlőjelét tartalmazza és a setControl(Control) metódust futtatja.

A setter metódusokat az utódban át lehet definiálni, be lehet teát tenni olyan utasításokat is, amelyeknek az adat módosulására le kell futniuk; így frissíthető például a vezérlőkliensen a vezérlőpanel vagy az akkumulátor töltöttség szintje.

Elsőre úgy tűnhet, hogy ez szembemegy az eddig megszokott eseménykezeléses eljárással, de megvan az előnye, hogy nagyon egyszerűen észlelhető mindenfajta adatmódosulás anélkül is, hogy Listener interfészket kellene létrehozni mindenféle adattípushoz; az ugyanis jócskán és fölöslegesen megnövelné a kód méretét. Helyette minden alkalmazás megörökli a *BaseData* osztályt, és úgy bővíti ki a setter metódusokat, hogy feldolgozhassa az adatok módosulását.

3.3.1.6. A kamrakép közvetítése

A kamerakép közvetítésére nem lehetett általános tervet készíteni, mivel minden alkalmazásnak más feladata van az JPEG folyammal. Ezért minden alkalmazásnak van egy-egy saját *VideoProcess* osztálya.

A telefonon futó járműkliens – ha már létrejött a híddal való kapcsolat és objektum keletkezett az az osztályból – kapcsolódni próbál a telefonon futó IP Webcam alkalmazás szerverével. Ha az nem érhető el, akkor a járműkliens elindítja a szervert, majd megvárja, hogy a szerver elérhető legyen. Ha végre kapcsolódott hozzá, akkor egyszerűen elkéri a bejövő folyamot és a hídszerverhez vezető kimenő folyamot, és ciklusban, egy kilobájtos egységekben továbbítja a hídszervernek az IP Webcam által generált JPEG folyamot.

A hídszerver járműkliens felőli feldolgozója pedig egyszerűen csak elkezdi olvasni a bejövő folyamot, dekódolja az JPEG-folyam képkockáit, majd tárolja az utolsó képkockát. A hídszerver vezérlőkliens felőli feldolgozója hozzáfér a képkockákat tároló objektumhoz és a kapcsolat

létrejöttekor az MJPEG szabványnak megfelelően előállít egy fejlécet, majd elküldi. Ha van a választott járműről tárolt képkocka, azt is elküldi. Ezután vár a képkocka megváltozására, és változás esetén kiküldi az új képkockát. minden járműhöz külön-külön tárolt képkocka tartozik.

A vezérlőkliens kapcsolatfeldolgozója ugyanúgy dekódolja az MJPEG folyamot, ahogyan a hídszerver, de már nem tárolja a képkockákat, hanem átadja a főablak JLabel komponensének, sz pedig megjeleníti az aktuális képkockát.

Ezzel a módszerrel tulajdonképpen olyan funkció alakult ki, mint a proxykban, azzal a különbséggel, hogy a hídszerver többfelé szórja az adatfolyamot.

3.3.1.7. Összegzés

Ebben a részben bemutattam, a hálózati kommunikáció működési elvét. Láthattuk, hogyan zajlik a kialakított kapcsolat ellenőrzése, hogyan küldenek és fogadnak üzenetet és hogy mik is ezek az üzenetek. Az utolsó fejezetben felvázoltam, hogyan közvetítik a kameráképet a különböző alkalmazások.

Ezzel a teljes a kép a hálózati részről: a kapcsolatot tehát a kliensek kezdeményezik, és minden alkalmazás három kapcsolatot alakít ki a szerverrel. Elsőnek létrejön a kapcsolatellenőrző, azután az üzenetküldő és fogadó, és az MJPEG folyam közvetítésére vagy fogadására épül ki a harmadik kapcsolat.

Hangot már nem közvetíték a telefonról, mert már így is a szokásosnál nagyobb terhelést kell elviselnie az MJPEG folyam megalkotásakor, de a fenti elvet követve létrejöhette egy negyedik kapcsolat is, s az a hangot közvetítené valamely kodekre támaszkodva.

3.3.2. A vezérlőkliens fontosabb elemei

A vezérlőkliens programozását a grafikus felülettel kezdttem. Ezután jött a hálózati kommunikációra írt osztályok felhasználása. Lényegét tekintve e két lépés után az alkalmazás eljutott a végső formába, már csak apróbb simítások kellettek, hogy minden alkalmazás 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; róluk 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 küllemük központilag módosítható, s ezzel az alkalmazás külleme is hozzáigazítható az operációs rendszer felületéhez. A Javában erre a célra hozták létre a Look And Feel (LAF) osztályokat. A különböző operációs rendszerekhez kiadtak Java virtuális gépek tartalmazzák az adott rendszerre írt LAF osztályt, s ha azokat még a Swing komponensek használata előtt alkalmazzuk, akkor a Swing komponensek hasonlítani fognak a natív komponensekhez.

A grafikus felületekre gyakran használatos eljárások összefogására létrehoztam egy Util osztályt. Ez az osztály tartalmazza többek között a rendszerhez igazított Look And Feel osztály beállítását úgy, hogy az Linux, Windows és OS X rendszeren is megfelelően működjön.

A LAF beállító metódus elsőre a linuxos GTK Look And Feel osztályt próbálja alkalmazni, mert a getSystemLookAndFeelClassName() metódus nem minden Linux rendszeren tér vissza a GTK LAF osztály nevével. Linuxtól különböző rendszeren a beállító metódus kihagyja a GTK LAF beállítását, s ezzel elkerüli az OS X rendszereken való alkalmazását. A Linuxtól különböző rendszereken a System Look And Feel megfelelően alkalmazható, így a rendszerhez tartozó LAF beállítása az utolsó lépés, 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 – a Java Natív Interfészre (JNI) támaszkodó – komponensgyűjtemény; az operációs rendszer adta natív felületi komponenseket használva lehet vele grafikus felhasználói felületeket létrehozni. Eredetileg az IBM fejlesztette ki, de ma már az Eclipse Foundation fejleszti és tartja karban, mivel az Eclipse IDE volt az első SWT-t használó alkalmazás.

Az SWT nem része a szabvány Java API-nak, csupán egy lehetőség az AWT és a Swing mellett. Előnye, hogy gyorsabb a Swing alkalmazásoknál és nincs szükség a Look And Feel használatára, mivel natív komponenseken alapszik. Hátránya, hogy sok helyet foglal az alkalmazások hordozható, platformfüggetlen kiadásaiban, mert minden operációs rendszerhez külön jar állományt tartozik, és minden rendszeren a neki megfelelőt 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 egy API-t a térképszolgáltatásához, s az elérhető JavaScript nyelven a webes alkalmazásokban, Androidra írt alkalmazásokban és iOS rendszeren is.

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

Java nyelvre több webböngésző-támogatás is létezik; a legfejlettebb közülük az SWT által adott 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 vele.

Az SWT használatának tehát az a fő oka, hogy megjeleníthessem a Google adta térképet.

3.3.2.1.3.1. Swing kontra SWT

Az SWT készítői felhívják a fejlesztők figyelmet arra, hogy az SWT-re támaszkodó alkalmazásokban a Swing és az AWT osztályai nem használhatók felület megjelenítésére, mert együttes használatuk az egész alkalmazás összeomlásával járhat, és rossz hatása lehet magára a grafikus felületre is. Debianon teszteltem ezt, és a GNOME valóban kifagyott az SWT leállítása után, sőt a Java alkalmazás sem volt hajlandó leállni. Ha konzolból „kilőttem” az alkalmazást, a GNOME újra megfelelően működött.

Megoldás lehetett elhagyni a Swinget és AWT-t, és teljes SWT alapú felületet írni, de inkább maradtam a Swing felületnél, mert nem kell hozzá külön jar állomány; kerestem erre a problémára egy megoldást. A tervem az volt, hogy mellékelem az SWT állományokat, és így elérhető marad a térkép szolgáltatása, de a térképszolgáltatás inaktiválásával SWT nélkül továbbra is működni fog a teljes alkalmazás.

3.3.2.1.3.2. Native Swing és bővítése

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

Christopher a következő fogást használta ebben a projektben: létrehozott egy natív interfészt, s annak az inicializálásakor egy új Java folyamat indul el a háttérben az SWT komponensek kezelésére. Ezzel a natív interfészt használó alkalmazásnak valójában nincsenek SWT komponensei, az új folyamatban indított Java alkalmazásnak pedig csak SWT komponensei vannak, mindenkit program biztonságosan üzemel tehát.

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

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

Ezenkívül létrehoztam egy adapterosztályt, mely az általam Native Swingre írt JTray osztályt használja, ha az SWT elérhető, különben meg az AWT SystemTray osztályát. Az értesítési ikonok tehát akkor is megjelennek, ha az SWT nem lenne elérhető.

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 operációs rendszerhez illő SWT helyét; a Native Swing projekt csak ezután használható.

Alapesetben erre az a bevett eljárás, hogy a classpath változóban felsoroljuk az alkalmazás által használt osztályokat, de mivel minden rendszerhez és ezen belül is architektúrához külön SWT tartozik, ezért több jar állományt kellene kiadni, s ez nehezítené a hordozhatóságot.

A megoldás az SWTJar^l használata. Azzal az SWT alapú alkalmazások könnyen becsomagolhatók

egyetlen jar állományba. A végeredményben kapott állomány tartalmazza az SWT Windowsra, Linuxra és OS X rendszerre kiadott változatát minden architektúrához – összesen tehát 6 jar állományt –, valamint tartalmazza az összes osztályt az ant feladatban megadott jar állományokból. A jar állomány minden rendszeren a megfelelő SWT könyvtárat tölti be jar-in-jar osztálybetöltővel, vagyis csak egyetlen jar állomány kell az alkalmazás indításához.

Az ant feladat (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 állomány magában foglalja a jar-in-jar osztálybetöltőt és az SWTLoader nevű osztályt; ez az osztály választja ki és tölt be a megfelelő SWT jar állományt. Az ant feladat a dist könyvtárba tesz egy ui.jar nevű állományt, magát a csomagolt alkalmazást.

Az SWTJar példájára írtam egy metódust; az a classpath változóhoz hozzáadja a megfelelő SWT jar állományt; az összecsomagolt állományból kivettem az SWT-t, mert különben túl nagy lett volna. Az SWT ehelyett a lib könyvtárban kapott helyet, így a különböző rendszerre kiadott telepítők jóval kisebb lettek, hiszen a más rendszerekhez tartozó SWT-támogatás nem foglalt további helyet.

Az SWT-t csak akkor töltetem be, ha az általános változókat deklaráló SWT osztály 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.

Ha kivétel keletkezik, akkor ki kell bővíteni a classpath tartalmát; ez az URLClassLoader osztály addURL(URL) privát metódusával érhető el, de azt 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 másodpercig is eltarthat, ezért készítettem egy töltőképernyőt, s az

nyomban megjelenik a Java indulása után. A hosszú betöltési időt főként a Native Swing projekt okozza, mert a natív interfész teljes betöltődéséig nem jelenik meg a kezdőképernyő.

A töltőképernyő^m gyors megjelenéséről maga a Java gondoskodik.

A *SplashScreen-Image*: *org/dyndns/fzoli/rccar/resource/splash.gif* attribútumot beállítottam a jar állománybeli META-INF/MANIFEST.MF állományban. 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 át. A SplashScreen.get SplashScreen() metódus nullhivatkozással tér vissza, ha a megadott gif állomány nem létezik vagy nincs megadva a SplashScreen-Image paraméter, vagyis ha nincs töltőképernyő-megjelenítés. Ha a töltőképernyő megjelent, akkor a SplashScreen.createGraphics() példánymetódussal készíthető egy grafikus objektum, s az arra való rajzolással 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 az első ablak megjelenésekor, de eltüntethető a close() metódus hívásával is. Fontos tudni, hogy ha a töltőképernyő egyszer eltűnt, akkor nincs mód az újból megjelenítésre.

3.3.2.1.5. A kamerakép megjelenítése, átméretezés (imgscrl)

A Motion JPEG folyam valójában JPEG képkockák sorozata; egy előre meghatározott jel választja el egyiket a másiktól. A folyamot olvasva így egyszerűen két határolójel közötti részt vehetünk egy-egy képkockának. A határolójelek közti részt bájtömbben tárolhatjuk, majd a tömb használatával létrehozunk egy *ByteArrayInputStream* objektumot, és az *ImageIO.read(InputStream)* metódussal létrehozhatunk egy *BufferedImage* objektumot.

A megjelenítés előtt azonban a képet 640×480 méretűre kell átméretezni, hogy illeszkedjen a felülethez. Erre azért van szükség, mert a telefon 320×240 képpontos képeket közvetít, hogy minél kisebb legyen az adatforgalom és a processzor terhelése.

A kép átméretezésére a Java Image Scaling Libraryt használom – röviden imgscrl –; a BufferedImage objektumok könnyen átméretezhetők vele. Nagy előnye, hogy gyors, testreszabható az átméretezés sebessége és minősége, valamint van benne élsimítás is.

A JLabel nem csak szöveget jeleníthet meg, hanem képet is: az átméretezett kép a JLabel.setImage(Image) metódus használata révén jelenik meg a felületen.

3.3.2.1.6. Az alkalmazás lokalizálása (ResourceBundle)

Mindhárom alkalmazás használható angol és a magyar nyelven. Most a hídszerver és a vezérlőkliens lokalizálásáról lesz szó; az Android rendszerre írt alkalmazások másképpen lokalizálhatók.

A Java nyelvben a lokalizációⁿ elterjedt és javasolt módszere a *ResourceBundle* osztály alkalmazása. minden nyelvhez külön properties kiterjesztésű állományt kell létrehozni. Ezeket az állományokat is a forráskódok közé kell betenni, és csomagon belül is szerepelhetnek. Az állományok kulcs–érték párokból állnak. A nyelvi állományokból a kulcsokkal lehet lekérni az adott nyelvhez tartozó szöveget. A *ResourceBundle.getBundle(String, Locale)* metódus első paramétere az állomány helyét és nevét adja meg, a második a kért nyelvet definiálja. A metódus a megadott Locale alapján egy *ResourceBundle* objektummal tér vissza, s az tartalmazza a helynek megfelelő nyelv kulcs–érték párait. Ha az adott nyelvhez nem tartozik properties állomány, akkor az alapértelmezés szerinti állomány jut szóhoz.

Alkalmazásomhoz három különböző nyelvi állományt hoztam létre. Mindhárom a org.dyndns.fzoli.rccar.l10n nevű csomagba került. A *chooser* properties állományok tartalmazzák az alkalmazásválasztó ablakhoz használt szövegeket; a *bridge* a hídszerver lokalizációja és a *controller* a vezérlőkliensé.

Az alkalmazásválasztó a következő utasítást használja a nyelvi állomány betöltéséhez: *ResourceBundle.getBundle("org.dyndns.fzoli.rccar.l10n.chooser", Locale.getDefault())*

A *Locale.getDefault()* az operációs rendszerben megadott helyen tér vissza, Magyarországon tehát magyar lesz az alkalmazásválasztó nyelve, másutt meg angol. A vezérlőkliens és hídszerver esetén az alapértelmezés szintén a tartózkodási hely nyelve, de a konfigurációban ez bármikor átállítható.

A lokalizált szövegek a *ResourceBundle.getString(String)* metódussal kérhetők le a kulccsal mint paraméterrel. Példa a properties állományok nevére: *bridge_en.properties*, *bridge_hu.properties*.

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

Hogy ne kelljen alapértelmezés szerinti nyelvi állományt létrehozni, az angol nyelvet állítottam a helyébe:

```
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 átdefiniálásával elértem, hogy ha a választott nyelvhez nem lenne definiálva valamelyik kulcs, akkor az angol nyelvi állomány jusszon szóhoz.

3.3.2.2. A konfiguráció tárolása

A vezérlőkliens konfigurációja nem szöveges állomány, mint a hídszerveré. Ennek az a fő oka, hogy a vezérlőkliens van grafikus felülete, s azon 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 adatreprézentálására fejlesztették ki őket.

Új konfigurációt a Config osztály példányosításával lehet létrehozni. Ez az objektum tárolja a hídszerverhez való kapcsolódáshoz szükséges adatokat. Ha a felhasználó elmenti az adatokat, akkor ennek az objektumnak a szerializáltja íródik bele egy állományba. Ha ez az állomány létezik és érvényes objektumot tartalmaz, akkor a program indításakor betöltődik a memóriába, és nem jön létre új konfiguráció.

Minden felhasználóhoz külön-külön létrejön egy ilyen, a Config objektumot tároló *controller.ser* nevű állomány, nehogy a felhasználók átírják egymás beállításait. Az alkalmazásoknak minden operációs rendszeren felhasználónként van egy-egy könyvtár; oda menthető a konfiguráció. Linux rendszeren ez a *.config* nevű könyvtár a felhasználó könyvtárában. Linuxon eszerint a valaki nevű felhasználó konfigurációs állománya a */home/valaki/.config/Mobile-RC/controller.ser* helyen található. Unix alapú rendszereken a felhasználó könyvtára a *~* jellel is helyettesíthető. OS X-en a *~/Library/Application Support/Mobile-RC* könyvtár van kijelölve a végleges és ideiglenes adatok tárolására.

Windows rendszereken nem mondható meg egyértelműen, melyik könyvtárba célszerű ezeket az adatokat menteni, mert erre verzióinként más-más hely szolgál. Van azonban a Windowsban egy SHGetFolderPath(HWND, int, HANDLE, DWORD, LPTSTR) szintaxisú C++-beli metódus, éspedig a shell32.dll nevű állományban. Ezzel a metódussal megtudható, hogy a kérdéses rendszeren melyik könyvtár van fenntartva az alkalmazásoknak. A metódus futása után az LPTSTR típusú változóban lehet megtalálni ennek a könyvtárnak a nevét.

3.3.2.2.1. Java Native Access

A Java Native Access – röviden JNA – olyan API, amellyel könnyű hozzáérni a natív megosztott könyvtárakhoz, így Windowson a dinamikus hivatkozású könyvtárakhoz is (DLL).

A JNA használatához nincs szükség C/C++ kód írására, mint a Java Native Interface-ben (JNI). Egyszerűen definiálni kell egy, a C++-ban írt fejlécnek megfelelő interfész. Ezután a JNA-val létrehozható egy, a megírt interfész implementáló objektum, s futtatja a megadott dll állomány metódusait.

A shell32.dll fájl SHGetFolderPath metódusának hívására 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 implementációja és példányosítása a JNA-val, majd 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 – arra az SWT szolgál – és arról a natív interfészről, amely révén ugyanabban az alkalmazásban egyszerre használható a Swing és az SWT. Ezenkívül bemutattam, hogyan hozható létre Javán többnyelvű alkalmazás asztali környezetben, és hogy mi a módja a leggyorsabban működő töltőképernyő létrehozásának. Végezetül bemutattam a JNA API-t;azzal egyszerűen lehet használni a megosztott könyvtárakban szereplő metódusokat.

3.3.3. A járműkliens fontosabb elemei

Ebben a fejezetben az Android fejlesztőkörnyezet főbb elemeit fogom bemutatni a járműkliens alkalmazásban játszott szerepe szerint.

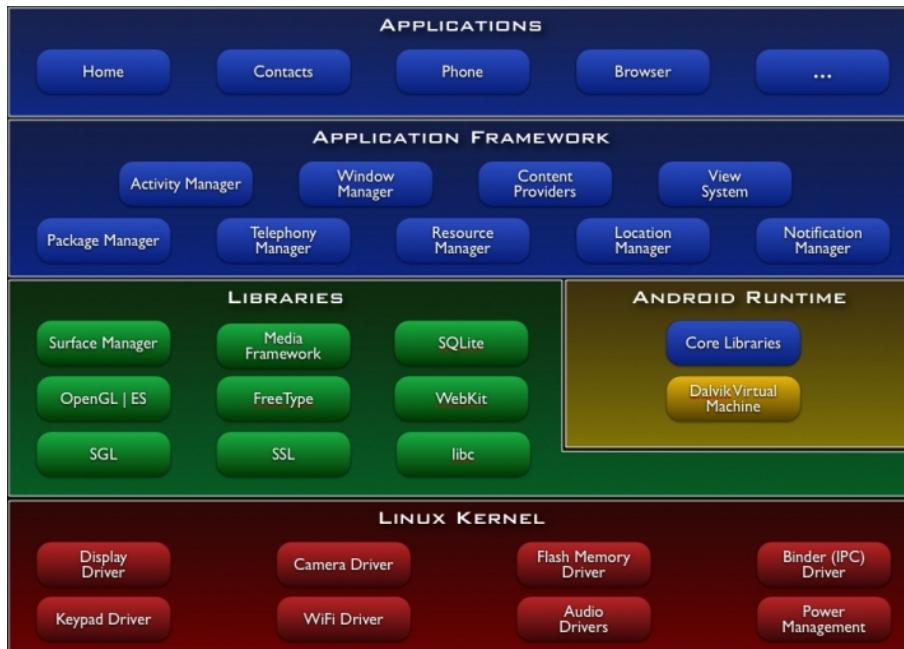
A járműkliens három fő részből áll. A felületen a főablak mutatja a vezérlőjelet és a feszültségszintet; emellett itt lehet elindítani és leállítani a háttérfolyamatokat. Az alkalmazáshoz tartozik egy tárhely is, ott tárolódnak a beállítások. A háttérfolyamatokról egy szolgáltatás gondoskodik. A továbbiakban ezeknek a részeknek a felépítéséről lesz szó.

3.3.3.1. Az Android platform

Az Android^p egy Linux rendszermagra épülő operációs rendszer. A Linux rendszermagon egy Java virtuális gép nyugszik: a Dalvik virtuális gép. A virtuális gép dolga a felhasználói felület kezelése és az alkalmazások futtatása.

A platform alapját a Linux rendszermag adja; abba vannak belefoglalva a hardver által kezelendő eszközök meghajtóprogramjai. Ezeket azok a cégek készítik el, amelyek a maguk készülékein használni akarják az Android platformot – hiszen a gyártónál jobban senki sem ismerheti a mobil eszközbe foglalt perifériákat. Ez a kis méretű rendszer kezeli a memóriát, ütemezi a folyamatokat és kezeli a teljesítményt is, hogy minél kisebb legyen az energiafogyasztás.

A rendszermag szolgáltatásait használják a Linux rendszerekben meglévő különféle programkönyvtárak, a libc, az SSL és az SQLite; ezeket C/C++ nyelven írták, és minden közvetlenül futnak a Linux rendszermagban. Részben ezekre épül a Dalvik virtuális gép; a Dalvik egyáltalán nem kompatibilis az Oracle virtuális gépével, teljesen más az utasításkészlete, és más bináris programot is futtat. A Java programok nem egy-egy .class állományba kerülnek a fordítás után, hanem egy nagyobb Dalvik Executable formátumú, .dex kiterjesztésű állományba, az általában kisebb, mint a forrásul szolgáló .class állományok, mivel a Dalvik fordító a több Java állományban meglevő konstansokat csak egyszer fordítja bele. A virtuális gép más, mint a Javában megszokott virtuális gép, vagyis a Java itt csak mint nyelv jelenik meg!



38. ábra: Az Android rendszer felépítése

A kék színnel jelölt részekben már csak a virtuális gép futtatta Java forrást találunk, s ez adja az Android lényegét: a látható és tapintható operációs rendszert, illetve a futó programokat. A virtuális gép teljesen el is rejtheti a Linux által használt állományrendszert, és csak az Android Runtime által kínált állományrendszert láthatjuk.

3.3.3.2. Az Android SDK

A fejlesztőkörnyezetet az Eclipse IDE adja. Az Androidra való szoftverek fejlesztése előtt le kell tölteni egy beépülőmodult az Eclipse-hez, és magát az Android SDK-t is. A beépülőmodul telepítése után a hozzá tartozó beállításokban meg kell adni az Android SDK helyét. A Window menüből ezután elindítható az SDK manager, s onnan letölthető a különféle Android rendszerekhez kiadott API. Ez után a Virtual Device Manager segítségével virtuális telefont is létre lehet hozni, az egyszerű alkalmazások teszteléséhez tehát nincs szükség valódi telefonra.

Az Android SDK-nak sajátos MVC modellje van. A Java nyelv mellett az XML-nek is fontos szerepet játszik az Android alkalmazások fejlesztésében. A nézeteket is XML nyelvben definiálják. A nézetet az Activity osztály jeleníti meg, és a nézet és vezérlő ebben az osztályban köthető össze az eseménykezelők használatával. A beállítások tárolhatók is XML állományban definiált struktúra

használatával, s ahhoz olyan nézet is rendelhető, amely a struktúra alapján létrehozza a felületet, és a segítségével a felhasználó módosíthatja a beállításokat. Maga a manifest állomány is XML formátumú.

3.3.3.3. AndroidManifest.xml

Az AndroidManifest.xml állomány nevezi meg a Java csomagok nevét, leírja az alkalmazás komponenseit. Meghatározza, hogy az alkalmazás komponensei melyik folyamathoz tartoznak. Meghatároz azokat az engedélyeket is, amelyek ahhoz kellene, hogy alkalmazások használhassák a komponenseit, és meghatározza az alkalmazás futtatásához szükséges minimális API-szintet.

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 alapértelmezésben mi az alkalmazás csomagja. Az állományban a ponttal kezdődő osztályok ebből a csomagból indulnak ki. A versionCode egyértelműen definiálja az alkalmazás verzióját, más kiadást nem lehet ugyanazzal a kóddal kiadni. A versionName attribútum látható a felhasználóknak is, és általában MAJOR.MINOR.PATCH^q formátumú.

A uses-sdk tagben (címkében) van definiálva az, hogy az alkalmazás melyik rendszerre íródott, s hogy milyen minimális verziószámmal futtatható.

Az alkalmazás telepítése előtt a rendszer kiírja, hogy az alkalmazás milyen jogosultságokra támaszkodik. Ezek a jogosultságok is a manifestben vannak felsorolva, egy jogosultság egy uses-permission címkének (tagnek) felel meg.

Az application címkében van felsorolva a használt Activity és Service osztály, meg a rendszerszintű események fogadására létrehozott IntentReceiver is. A receiver címkében vannak felsorolva azok a szűrők, melyekre az alkalmazás feliratkozott. Így például a BOOT_COMPLETED esemény hatására leprogramozható, hogy az alkalmazás elinduljon a rendszer felállása után.

A manifest alapján a MainActivity nevű osztály indul el az alkalmazásindítóban szereplő ikon kiválasztásakor.

3.3.3.4. Az alkalmazás főablaka, Activity

Az alkalmazás főablaka egy közönséges Activity; ennek küllemét activity_main.xml szabja meg:

```
<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>
```

A RelativeLayout szolgál elrendezésmenedzserül. Egyszerűen lehet vele komponenseket a képernyő közepére igazítani, és a komponenseket is egyszerűen lehet egymás mellé helyezni.

Minden komponensnek egyedi azonosító adható. Ennek birtokában lehet a Java kódban hozzáérni a felületi komponensekhez.

A TextView egy címke; az asztali Java alkalmazásokban a JLabel-nek feleltethető meg neki. A feszültségszint megjelenítésére szolgál; az `android:layout_centerHorizontal="true"` tulajdonság beállításával az elrendezésmenedzser a képernyő közepére helyezi a címkét, az `android:layout_below="@+id/arrow"` beállítás hatására a vezérlőjelet ábrázoló komponens alá.

A vezérlőjelet ábrázoló komponens saját komponens, nem része az Android SDK-nak, ezért ki kell írni a teljes nevét. Az ArrowView komponens a képernyő közepén helyezkedik el; csak a szükséges nagyságú helyet foglalja el, és 5 sűrűségfüggetlen pixel a margója. A sűrűségfüggetlen pixel (density-independent pixel) révén azonos távolságú margó állítódik be az egymástól eltérő felbontású kijelzőkön is.

A képernyő alján látható Start és Stop nyomógomb valójában Button komponens; feliratuk a a nyelvi állományuktól függ. Az `android:text` értéke a `"@string/text_start"` kulcsú szövegnek megfelelően értékelődik ki. A lokalizációról később még szó fog esni.

A Java forráskódban az Activity létrejötte után lehet a felületet legyártani az XML állományból, a `setContentView(R.layout.activity_main)` utasítással. Ezután a legyártott komponensek referenciaja a `findViewById(int)` metódussal szerezhető meg.

Példa a feszültségszintet megjelenítő címke referenciájának megszerzésére:

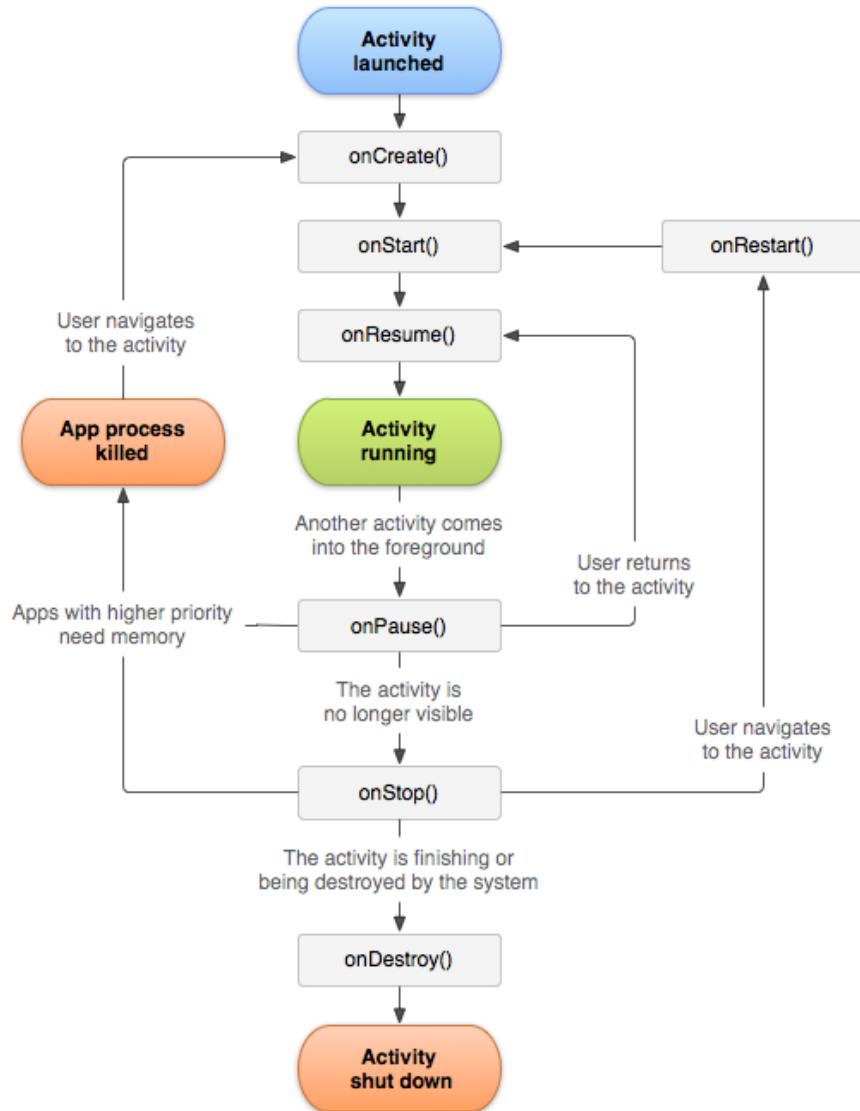
```
tvVoltage = (TextView) findViewById(R.id.tv_voltage);
```

Az R osztály neve a Resource rövidítése. Az osztályt az SDK állítja elő az XML állományok alapján. Csak azonosítókat tartalmaz; azokkal címezhetők az XML állományok és a bennük szereplő, azonosítóval ellátott komponensek is. A layout könyvtárban levő activity_main.xml tehát az R.layout.activity_main azonosítóval érhető el, az abban szereplő, tv_voltage azonosítót használó komponens az R.id.tv_voltage azonosítóval címezhető meg.

3.3.3.4.1. Az Activity életciklusa

Minden Activity indíthat további Activityt. Amikor egy új Activity elindul, akkor az előző megáll, és a rendszer közreműködésével egy verembe kerül. Ez a verem LIFO mechanizmus alapján működik, ha tehát a felhasználó visszatér az új Activityból, akkor az előző fog folytatódni. Az ilyen állapotváltozásokat különböző visszahívási (callback) metódusok hívásával jelzi a rendszer az Activitynek. Az Activity életciklusát^r és a visszahívó metódusokat a következő oldalon lévő ábra mutatja. Amikor egy Activity háttérbe kerül, a rendszer leállíthatja, hogy memóriát szabadítson fel más Activitynek. Ha ilyenkor újra meghívjuk az Activityt, akkor a rendszer újra létre fogja hozni.

A főablak felületének létrehozása, a felületi komponensek referenciáinak megszerzése és a kapcsolat kialakítása a háttérfolyamattal az `onCreate()` metódusban fut le, mivel ezeknek csak az Activity létrejöttekor kell lefutniuk. Az `onResume()` metódus lefutásakor az alkalmazás ellenőrzi, hogy történt-e végzetes hiba a háttérfolyamat futása közben, és ha történt, akkor leállítja a háttérfolyamatot. Az `onDestroy()` metódus lefutásakor lezárul a háttérfolyamattal kialakított kapcsolat, és ha a háttérfolyamat leáll, akkor leáll vele az IP Webcam alkalmazás is, majd meghívódik az ős metódus; az a felületi komponensek és az Activity törlésével felszabadítja a memóriát.



39. ábra: Az Activity életciklusa

3.3.3.4.2. SherlockActivity, ActionBarSherlock

Az Activity vékony címsorát az Android 3.0-tól kezdve az Action Bar váltotta fel. Ez azért jó, mert azokon az eszközökön, melyeken nincs vissza gomb és menügomb, nem kell emulálni őket; az Action Bar ugyanis helyettesíti ezeket a gombokat. A régi menü helyébe az action overflow lépett; ha rákattintunk, akkor lenyílik a menü; a menü bizonyos komponensei azonban action itemként is feltehetők a sávra. A főablak felületén a beállításokat megjelenítő gomb szintén action item.

Az én telefonom eredetileg 2.1-es verziójú Androiddal működött; nem hivatalos forrásból sikerült a verziószámot 2.3.3-ra növelni, de az Action Bar még ez a verziószám sem elég nagy. Mégis találtam olyan alkalmazásokat, megjelenítik ezt a sávot 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 a hivatalosan nem támogatott rendszereken eltüntetik a címsort, és helyére kirajzolják az eredeti sávval megegyező küllemű utánzatot. Ha van hivatalos támogatás, akkor az eredeti metódusok lépnek működésbe.

Az ActionBarSherlockot egyszerű használni. Csupán a SherlockActivity osztályt kell örököltetni, nem az Activityt. A SherlockActivitynak őse az Activity, így utólag sem kell a kódot módosítani.

3.3.3.5. Az alkalmazás konfigurációja, PreferenceActivity

Az alkalmazás konfigurációját az Android rendszer tárolja. Az xml könyvtárban létrehoztam egy preferences.xml nevű állományt; ez definiálja a tárolandó adatok típusát, azonosítóját, az alapértelmezés szerinti 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 állományban két kategória definíciója látható. Az első csoportban két EditTextPreference került, szerkeszthető beviteli mező mindenkorral. Az első EditTexttel normál billentyűzet jelenik meg, a másodikkal csak számok vihetők be, ezért numerikus billentyűzetet jelenít meg a rendszer.

A második kategóriában az első elem egy felsorolást jelenít meg, ezért kapta a ListPreference nevet. A felsorolás elemeit az arrays.xml állományból olvassa ki; ennek egy egyszerűsített változata a következő oldalon látható.

A CheckBoxPreference logikai értéket tárol. Ezt a típust függőségnek is fel lehet használni. Alapértelmezésben ha egy másik mező függ az adott CheckBoxPreference értékétől, akkor nem szerkeszthető az értéke, csak ha a CheckBox be van pipálva. Ha azt szeretnénk, hogy fordítva viselkedjen, akkor a `android:disableDependentsState="true"` tulajdonságot kell megadnunk; így a többi Preference akkor lesz csak szerkeszthető, ha az adott CheckBox értéke hamis.

Az egyszerűsített values-hu/arrays.xml állomány 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 PreferenceActivity megjeleníteni a preference.xml állományt. Mivel én Action Bar visszamenőleges támogatásáért az ActionBarSherlock projektet használom, ezért valójában a SherlockPreferenceActivity osztályt használom a beállítások megjelenítésére és manipulálására.

A beállításokat kezelő osztály neve SettingActivity; az Activity létrejötte után az addPreferencesFromResource(R.xml.preferences) utasítással jeleníti meg az XML állományban megadott komponenseket.

Az XML-ben beállított korlátozásokat bevitelimező-validálással bővítettem ki; ennek az eseménykezelés az alapja. Két fajtájú eseményfigyelőt használok. A TextWatcher a szöveg módosulása előtt és után is fogadhat eseményt; ezzel elértem, hogy a biztosan tiltott karakterek nem kerülnek bele az EditText komponensbe. Az OnPreferenceChangeListener még a Preference módosulása előtt megakadályozhatja a módosítást, így a félbehagyott bevittelek nem mentődnek el.

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; ha az alkalmazás logójára kattintunk, akkor az Activity bezárul.

3.3.3.5.1. SherlockPreferenceActivity, az ActionBarSherlock előnye

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

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

Az appcompat librarynak az az egyetlen hátránya, hogy csak Activity osztályokat lehet vele helyettesíteni az android.support.v7.app.ActionBarActivity osztály révén, PreferenceActivity helyettesítésére ma még nincs lehetőség.

Az ActionBarSherlock viszont a SherlockPreferenceActivity osztály támogatásával a PreferenceActivity osztályt is helyettesítheti, így alkalmazásom beállítások ablaka is megjelenítheti az ActionBar sávot.

3.3.3.6. Nyelvi fájlok

Az Android többnyelvű operációs rendszer. A felhasználó az első indításkor kiválaszthatja, milyen nyelvű legyen, s azt a Beállításokban utólag is bármikor módosítani lehet. A konvenciót betartó alkalmazások nyelve a rendszer nyelvével együtt változik. Ennek az a feltétele, hogy ne égesse be a szöveget az alkalmazásba, hanem a rendszer által támogatott lokalizációt használja.

Minden Android alkalmazásnak van egy res nevű könyvtára. Az alapértelmezés szerinti nyelvhez tartozó XML állományok 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 állományok ezekben a könyvtárakban foglalnak helyet, és strings.xml a nevük. A rendszer a beállított nyelvnek megfelelő values könyvtárat használja; így értem el, hogy a magyar nyelvű rendszeren alkalmazásom magyar nyelvű legyen, minden más nyelv esetén viszont angol, mert alkalmazásomnak az az alapértelmezés szerinti nyelve. A nyelvi állományok használat az alkalmazásom kódjával mutatom be.

A res/values/strings.xml állomány 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 állomány 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 R nevű generált osztály az XML-ben deklarált feliratoknak megfelelően tartalmazza a szövegek 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 révén maga a String objektum is létrehozható, ha esetleg több szöveg összefűzésére lenne szükség, de a felhasználónak String nélkül is lehet üzenetet megjeleníteni, a legegyszerűbben a Toast osztály használatával: Toast.makeText(this, R.string.set_config, Toast.LENGTH_SHORT).show()

Az előbbi utasítás egy két másodpercig látható üzenetet jelenít meg a használt nyelvi állománynak megfelelően.

XML állományból is lehet hivatkozni a strings.xml állomány szövegeire. Erre már láthattunk példát az AndroidManifest.xml állomány bemutatásakor; abban adható meg az alkalmazás neve. A `@string/app_name` hivatkozik az alkalmazás névére; így lehet nyelvfüggő nevet adni az alkalmazásnak.

3.3.3.7. Az alkalmazás háttérfolyamatai, Service

A Service egy felhasználói felület nélküli komponens; elsősorban hosszabb időigényű háttérfolyamatok elvégzésére használhatjuk. Más komponensek indíthatják a Service-t és a háttérben akkor is tovább futnak, ha közben a felhasználó egy másik alkalmazásra váltott. Például háttérben zajló zenelejátszás, hálózati kommunikáció stb. Alapjában kétféle Service létezik.

A Service „started” típusú, ha egy alkalmazáskomponens (például egy Activity) elindítja a startService(Intent) metódussal. Ha egyszer a Service elindult, akkor folyamatosan futhat még azután is, hogy megszűnik az őt meghívó komponens. A „started” Service-ek nem adnak vissza értéket a hívónak.

A Service „bound” típusú, ha egy alkalmazáskomponens meghívja a bindService(Intent, ServiceConnection, int) metódust. Az egy kliens–szerver interfész kínál, vagyis kéréseket küldhetünk és válaszokat fogadhatunk vele.

Az én alkalmazásomban a Service minden típusba beletartozik. Egyszerűbb szükség van a folyamatos futásra, hogy az áramkör vezérelhető legyen, és a folyamatos hálózati kommunikáció fenntartására is, másfelől a főablaknak is el kell érnie a Service objektumait, hogy kirajzolhassa a vezérlőjelet, illetve offline módban még szükség van a vezérlőjel módosításra is.

3.3.3.7.1. IOIO Service

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

Az eszköz gyártója ad egy, az internetről lehet beszerezhető library projektet. Ennek a projektnek saját Activity és Service osztálya van, s azok révén kapcsolat teremthető az IOIO-val. A `public IOIOLooper createIOIOLooper()` metódust az Activityben, a Service-ben át kell definiálni úgy, hogy egy IOIOLooper interfész implementáló objektummal térjen vissza.

Az IOIOLooper használja a mikrogéppel kialakított kapcsolatot. Mihelyt létrejön a kapcsolat, a `setup(IOIO)` metódus meghívódik, egy IOIO objektummal mint paraméterrel, mely a kapcsolatot képviseli és a használható utasításokat definiálja. Ebben a metódusban kell lefoglalni az IOIO-nak az alkalmazás által használandó PIN-jeit. A lefogláskor definiálni kell a PIN használatának módja is. Például az `ioio.openDigitalOutput(10, false)` utasítás a 10-es PIN-t lefoglalja digitális kimenetnek, és logikai hamisra állítja a kezdőértékét. A metódus egy DigitalOutput típusú objektummal tér vissza, azon át állítható a kimenet értéke.

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

Az IOIO leválasztásakor meghívódik a `disconnected()` metódus. Ebben a metódusban tilos és értelmetlen is használni az IOIO objektumot, mivel a kapcsolat már lezárult. Ez a metódus csupán tájékoztatásul szolgál.

3.3.3.7.2. Impulzusszélesség-moduláció (PWM)

Az IOIO kimenetei csak digitális kimenetek. Ez azt jelenti, hogy egy logikai értéknek megfelelően vagy van jel a kivezetésen, vagy nincs jel; a kimenet feszültsége vagy 0 volt, vagy 3,3 volt. De ha, mondjuk, 1,65 voltra lenne szükség, akkor azt csak valamilyen kerülővel érhetjük el. Megtehetjük, hogy a digitális kimenetet meghatározott időközönként ki-be kapcsoljuk úgy, hogy egyforma ideig legyen ki- és bekapcsolva, és akkor a hosszú távú átlagfeszültség a 3,3 volt fele, vagyis 1,65 volt lesz. Ha ennél nagyobb átlagfeszültséget szeretnénk, akkor valamivel tovább kell bekapcsolva hagyni a kimenetet, ha meg alacsonyabbat, akkor nyilván rövidebben.

Éppen erre találták ki a PWM-et (Pulse Width Modulation). A PWM^s jel előállítása alapszükséglet, ezért minden modern mikrovezérlő hardverében ott van, nem kell szoftveresen előállítani, s ez tehermentesíti a processzort. A hardveres PWM-mel csak közölnünk kell, hogy milyen periódusidejű és kitöltési tényezőjű PWM jelre van szükségünk, és ezzel a CPU dolga véget is ér, dolgozhat tovább más feladaton; a PWM hardver majd előállítja a kívánt jelet a hozzá tartozó kimeneti lábon.

Az IOIO is hardveresen állítja elő a PWM jeleket. 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 egy PwmOutput objektummal tér vissza; azt a loop() metódusban lehet használni. A kitöltési idő a `setDutyCycle(float)` utasítással módosítható. A kitöltési idő paramétere 0 és 1 közötti érték lehet: a 0 felel meg a 0%-os kitöltésnek, az 1 felel a 100%-osnak.

3.3.3.7.3. A motor vezérlése

A motor vezérléséhez írtam egy, az IOIOLopper interfészről leszármazó Vehicle interfést. A Vehicle interfész implementálásával vezérelhető a jármű. Két osztályt is írtam erre a célra: az egyik DigitalOutput objektumokkal működik, a másik PwmOutput objektumokkal. A kettőben azonos módon zajlik a feszültségszint kiolvasása és a vezérlőjel megszerzése, ezért a közös metódusoknak írtam egy AbstractVehicle nevű osztályt.

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 a Service referenciáját várja paraméterül – hogy olvashassa a vezérlőjelet –, és amellett még a konfigurációban megadott indexet.

A Service az IOIOLopper létrehozásakor használja ezt a metódust:

```
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 bejegyezhessen egy, a feszültségszint változását jelző eseményfigyelőt.

3.3.3.7.4. A telefon szenzorainak kezelése

Az alkalmazás három szenzort használ: GPS-t, gyorsulásmérőt (gravitációs tér) és a mágneses teret érzékelő szenzort.

Az Androidban a LocationManager osztály használható helymeghatározásra. A rendszer a telefon pozíóját több forrásból is kiderítheti, azok pontosságban térnek el egymástól. Város szintű helymeghatározáshoz az alkalmazásnak nem kell engedélyt kérnie, a pontos helymeghatározáshoz már igen.

A rendszerben a helyadatok megváltozása a LocationListener eseményfigyelővel észlelhető. Ha a pozíció módosul, akkor 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 sugárral.

Az eseményfigyelő bejegyeztetéséhez előbb meg kell tudni a LocationManager referenciáját: (LocationManager) getSystemService(Context.LOCATION_SERVICE);

Ezután az addGpsStatusListener(LocationListener) metódus használatával lehet eseményfigyelőt bejegyeztetni, kizárolag GPS jel figyelésére.

Az eseményfigyelő bejegyeztetése azonban egymagában még nem elég, mert a rendszert meg is kell kérni, hogy kezdje meg a helymeghatározást. Enélkül a kérés nélkül nem lennének események, az eseményfigyelő tehát nem kapna semmit.

A GPS jel gyorsabb megtalálásához a rendszer használhatja a mobilhálózatot: lekérheti a műholdak pozíóját. Ezért a szolgáltatásért esetleg fizetni kell, ezért a helymeghatározást ingyenes forrásokra korlátoztam:

```
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 helymeghatározást az utolsó utasítás indítja el, és az események másodpercenként vagy 10 méterenként keletkeznek majd.

A telefon szenzoraiban keletkezett adatok (például a mágneses térerősség) figyelése is eseménykezelésen alapszik. Erre a célra a SensorEventListener eseményfigyelő használható. Ha valamelyik szenzortól új adat érkezik, akkor az onSensorChanged(SensorEvent) metódus fut le. A SensorEvent megadja, hogy melyik szenzortól érkezett az adat; a float típusú values tömb tartalmazza a kapott értékeket.

A gravitációs és a mágneses mező erőssége háromdimenziós adat, a values tömb tehát három értéket tartalmaz. Az egyszerűség kedvéért létrehoztam egy tárolóosztályt a szenzoradatok tárolására; azok részadatba csomagolva elküldhetők a hídszervernek. Ennek megfelelően így fest az esemény feldolgozása:

```
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 a gravitációs vagy a mágneses mező erősségéről van-e szó, s aszerint módosítom a jármű adatában a Point3D objektumot. Ezután meghívom a fireSensorChanged() metódust, s az elküldi a részadatot a hídszervernek.

A két térerősség adataiból megállapítható, hogy a telefon (és maga a jármű) merre néz. A mágneses mező erőssége azonban nem elég ahoz, hogy ezt ki lehessen számolni; a telefon függőlegeshez viszonyított állására is szükség van. Ezt a gravitációs mező erősségéből lehet kideríteni.

3.3.3.7.4.1. Északi mágneses pólus, földrajzi észak

A mágneses északi pólustól való eltérés irányszögét (azimut) a hídszerver számítja ki a szenzoradatokból, és a fokban megadott értéket továbbküldi a vezérlőklienseknek. Mivel az Androidban már megvan, hogyan kell ezt kiszámítani, a szerveroldalon felhasználtam a forráskódot.

Az irány meghatározása tehát így megy:

```

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 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 ahoz is ad segítséget, hogyan lehet kiszámítani a szenzoradatokból a valós északi eltérést. 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, és a hídszerver a kiszámolt értékhez hozzáadja az így kapott elhajlást. Az elhajlás pozitív, ha a valódi északi pólus kelet felé esik, 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 magának az Android rendszernek a felépítését, fejlesztőkörnyezetét. Láthattuk, hogyan épül fel egy alkalmazás projektje; s hogy abban a Java forráskódon kívül az XML állományok is fontos szerepet kapnak. Bemutattam a felületet megjelenítő Activity osztályok általam használt típusait, és a hozzájuk tartozó XML állományokat – azok definiálják, milyen lesz majd a felület. Végül szó esett a háttérfolyamatokat üzemeltető Service osztályról: az vezérli a szenzorokat, a hálózati kommunikációt és az IOIO-t. Ezzel fel is vázoltam a járműkliens fő elemeit.

4. Befejezés: továbbfejlesztés

Az eddigi fejezetekben bemutattam a program funkcióit és útmutatást adtam hozzájuk, majd felvázoltam az alkalmazások hátterében álló gondolatokat. Befejezésként megemlítek néhány továbbfejlesztési lehetőséget; azok elvezhetőbbé tennék a jármű vezérlését és eladhatóbbá a projektet.

4.1. Hang továbbítása

Hangtovábbítást nem implementáltam az alkalmazásokba, mert még tömörítéssel is nagyon megnövelné az adatforgalmat, és a tömörítés a processzort is terhelné. Az én telefonom nem birkózna meg ekkora terheléssel, és a kép és a hang csak is egyre jobban késne.

Ám ha a telefonnak többmagos processzora lenne, akkor már továbbítani lehetne a hangot is – WiFi hálózaton díjmentesen. Viszonylag nagy területen már hasznos funkció lenne a hang továbbítása.

A járműről a hídszerverre való hangtovábbítást tetszés szerint ki- és be lehetne kapcsolni, de a hang fogadását minden vezérlőkliensen egyedileg lehetne ki-be kapcsolni, ha az éppen engedélyezve lenne. A járművet irányító felhasználó is küldhetne lejátszandó hangot a járműnek. Ez elég mulatságos helyzeteket teremthetne.

4.2. Az összes jármű a térképen

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

4.3. Játékmód útvonaltervezéssel

Az eladhatóságot tovább növelné a virtuális játékvilág kiterjesztése a valóságra. A játékmód úgy festene, hogy a hídszerveren előre definiálva lenne a versenypályát megszabó útvonal. A járművek a versenypályán versenyezhetnének egymással – éppúgy, ahogyan az autós játékokban. A rendszer nyomon követné a járművek útvonalát, és az nyerne, aki az útvonalról le nem térve elsőként érne célba. Az eredmények a chat ablakon jelennének meg.

4.4. Egyedi karosszéria, nagyobb teljesítmény

A projekt szerintem akkor válna piacképessé, ha lenne hozzá olyan egyedi karosszéria, amelybe biztonságosan betehető egy okostelefon, és a jármű nagy sebességen is könnyen irányítható maradna. A telefont integrált áramkörrel lehetne helyettesíteni, de gondolom, jobb lenne megmaradni a telefon mellett, mivel így sokkal olcsóbban lehetne árulni a járművet. Az Android okostelefons felhasználók olcsón juthatnának vele gyors, mobilinterneten át vezérelhető elektromos kisautóhoz. A projekt ezzel elérné a célját.

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