



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Weisz Patrik

DROIDLAB MÉRŐRENDSZER KOMPONENSEINEK FEJLESZTÉSE

KONZULENS

Lajtha András Balázs

BUDAPEST, 2013

Tartalomjegyzék

Tartalomjegyzék	2
Összefoglaló	5
Abstract.....	6
1 Bevezetés.....	7
2 Crowd-sourcing és crowd-sensing megoldások.....	8
2.1 Mobil Crowd-sensing alkalmazások a piacon	10
3 DroidLab rendszer bemutatása	12
3.1 Rendszerrel szemben támasztott követelmények.....	12
3.2 Android	13
3.3 Google App Engine	15
3.4 Google Cloud Messaging.....	17
3.5 Rendszer-architektúra	17
3.6 Kommunikációs interfészek.....	19
4 Rendszer használói	24
4.1 Kutatók.....	24
4.2 Felhasználók	24
4.3 Core fejlesztők	24
5 Megvalósítás	25
5.1 Teszt-plugin	25
5.2 Tesztmodul.....	27
5.3 Keretrendszer és Plugin közötti interfész	29
5.4 Modul ütemezése	31
5.5 Module logger.....	31
5.6 PHP web-szerver prototípus	32
5.7 Felhő alapú web-szerver	32
6 Teszt-eredmények	39
6.1 Keretrendszer funkcióinak tesztelése.....	39
6.2 Plugin funkcióinak tesztelése.....	40
6.3 Modul funkcióinak tesztelése	41
6.4 Webszerver funkcióinak tesztelése	41

7 Összefoglalás.....	43
8 Köszönetnyilvánítás	44
Rövidítésjegyzék.....	45
Irodalomjegyzék.....	47

HALLGATÓI NYILATKOZAT

Alulírott **Weisz Patrik**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2013. 12. 09.

.....
Weisz Patrik

Összefoglaló

Az Android platform olcsó eszközeivel, széles közönségével és nyílt szemléletével egy ideje felkeltette a kutatói társadalom érdeklődését. Azonban az Androidra való fejlesztés igen komplex, illetve egy új kutatási célokat szolgáló alkalmazás és mérési eredményeket gyűjtő infrastruktúra elkészítése a platform programozásában jártas fejlesztőnek is több heti munka. Az alkalmazás terjesztése pedig körülményes mozgósítható felhasználói bázis nélkül. Felismerve a problémát terveztük meg a DroidLab-ot. A DroidLab elfedi a kutatók elől ezeket az általános feladatokat és használata nem igényel platform-specifikus programozási ismereteket, így megkönnyíti a kísérletek előkészítését és elvégzését.

A DroidLab platform négy fő komponensből áll. Az eredményeket gyűjtő és közzétevő web-szerver, a modulokat futtató keretrendszer, a mérési elrendezést tartalmazó modul és a méréseket végző plugin-alkalmazás. Az alkalmazást telepítő felhasználók szabályozhatják a kísérletek számára biztosított jogosultságokat és erőforrásokat. A résztvevők védelme mérési eredmények anonimizálásával válik teljessé. A kutatók a rendszerben regisztrálva specifikálhatják a kísérleti elrendezést, és feltölthetik a futtatni kívánt scripteket. A DroidLab ütemező keresi meg az erre alkalmas eszközöket, telepíti a keretrendszerbe a modult, és begyűjti az eredményeket, amelyeket webes felületen tesz elérhetővé a kutató számára.

Abstract

The cheap Android devices with a wide user audience and opened approach have attracted the interest of the research community. However, the Android application developing is complex, and it also takes several weeks for a developer to design and implement a distributed measurement-framework for smartphones. The propagation of the application could be also circumstantial without user base. Recognizing the problem we designed the DroidLab, which hides all the general tasks and the platform-specific programming knowledge to facilitate the preparation and completion of the measuring tasks.

The Droidlab platform has four main components. The web-server which collects and display the measurement data from devices, the framework that runs modules, the module that contains the runnable scripts and plugin application that does the measurements. Those who install the applications on their smartphones can control which permissions and resources will be available for the measurements. The results of the measuring are anonymous to protect the users. After registration, the researcher is able to determine the measuring parameters and upload the written script. The DroidLab scheduler gets all the available devices, deploys the module to the smartphone, and collects the results that are accessible on a web-based user interface.

1 Bevezetés

Napjainkban rohamos fejlődésnek indult a mobil iparág az okostelefonok és tabletek megjelenésével. Az eszközökben megjelenő nagyobb számítási kapacitás, nagyobb érintőkijelző, nagyobb sávszélességű adatkapcsolat egy új utat nyitott az informatika területén. Az új funkcióknak és a mobilitásnak köszönhetően az okostelefonok és tabletek széles körben kezdik kiváltani az asztali gépeket. Ezek az eszközök szerves részeivé váltak a hétköznapiainknak. Bárhol böngészhetjük a webet, posztolhatunk a közösségi oldalunkra, nézhetünk videót, tölthetünk fel fényképet, élőben közvetíthetünk, navigálhatunk, vagy éppen vásárolhatunk. A felhő alapú technológia és a folyamatos háttér-szinkronizáció segítségével mindig naprakészek lehetünk.

Ennek köszönhetően azonban megnövekedett a mobil-készülékek által generált hálózati forgalom, ami újabb kihívások elé állították a hálózatok tervezőit és fejlesztőit. A vezeték nélküli hálózatokban a mobilitásból fakadóan sokkal több szabad paraméter jelentkezik, mint egy vezetékes hálózatban. Emiatt több mérésre, fejlesztésre, és optimalizálásra van szükség. A jelenlegi, mesterséges teszhálózatok nem adnak pontos képet arról, hogy a valódi hálózatokban milyen sajátos folyamatok játszódnak le. Szükség van tehát egy olyan teszhálózatra, amely valódi környezetben, valódi eszközök segítségével nyújt információkat a hálózatról. A DroidLab teszhálózata erre vállalkozik, ám saját mobil eszközök helyett önkéntes felhasználók eszközeiből építi a teszhálózatot.

Ebben a dolgozatban elsőként a már meglévő crowd-sensig alapú alkalmazásokat, azok előnyeit, hátrányait, típusait, és sajátosságait tekintjük át. A további részekben a DroidLab alkalmazás architektúrája kerül bemutatásra, továbbá megismerkedhetünk a rendszer által használt fontosabb platformokkal. Ezek után az általam elvégzett féléves munka bemutatása következik, végül pedig egy rövid áttekintést és összefoglalást olvashatunk az eddig elkészült munkáról.

2 Crowd-sourcing és crowd-sensing megoldások

A crowd-sensing [1] (magyarul: közösségi-érzékelés) egy új paradigma, amely a közösségben rejlő lehetőségeket igyekszik kiaknázni. Fő motivációja, hogy a közösség tagjai megosszák egymással az általuk érzékelt jelenségeket és így szélesebb rálátást, tudást szerezhessenek a környezetükről. Ez a fajta megközelítés egy speciális esete a crowd-sourcing [2] paradigmának, melynek célja, hogy jól meghatározott feladatokat szervezzen ki (outsourcing) egy bizonyos célközönség (crowd) számára. A következőkben két példát láthatunk, amelyek a crowd-sourcing alapú megközelítést mutatják be.

A uTest [3] egy olyan crowd-sourcing alapú szoftver-tesztelést végző vállalat, amely több lehetőséget is biztosít webes, asztali és mobil alkalmazások teszteléséhez. Ilyen lehetőségek a funkcionális, biztonsági, teljesítmény, lokalitás-függő, és használhatósági tesztelések. A különböző tesztelési formák több oldalról is megvizsgálják az alkalmazást, így világos képet kaphatunk arról, hogy miképp fog működni az alkalmazásunk éles bevetésben. A felhasználók ebben az esetben egy válogatott fejlesztőcsapat, akik a hiba-keresést, optimalizálást és fejlesztést végzik. Minden felfedezett hibáért és annak javításáért jutalmazást kapnak a cégtől. Jól látható tehát, hogy ebben az esetben egy jól meghatározott feladat kiszervezése történt egy meghatározott közönség számára, illetve a közösség tagjai külső motiváció hatására végezték a méréseket.

A Wikipedia [4] sokak által ismert közösségi oldal, amely szintén crowd-sourcing elven működik. Ennek segítségével egy olyan nyílt tudásbázist sikerült létrehozniuk az alapítóknak, amely mind a mai napig páratlan a világon. Előnye, hogy bárki szerkesztheti, földrajzi helyzettől, kultúrától, nyelvtől, vallástól, képzettségtől függetlenül. Így különösen színes képet kaphatunk arról, hogy egy adott témában hogyan vélekednek az emberek. Ugyanakkor hátránya, hogy az adatok és tapasztalatok helyességét olykor kevésbé lehet ellenőrzés alatt tartani. Ennek javítására szolgál a Wikipedia saját review rendszere, amely garantálja, hogy olyan szerkesztők vizsgálják felül a tartalmat, akik tisztában vannak a Wikipedia alapelveivel, és kellő tapasztalattal rendelkeznek a szerkesztésekhez. Ezen hátrány miatt azonban tudományos célra kevésbé használható, de általánosabb felhasználásra megfelelő lehet. Jelen esetben tehát

a kiszervezett feladat az, hogy tudást osszunk meg egymással, vagy a már meglévő tudás-bázist bővítsük, rendszerezzük. A feladatok elvégzéséért többnyire nem jár külön jutalom, azonban a megosztott tudás egységesen érhető el mindenki számára. Így tehát valamilyen szinten kárpótolva vannak a felhasználók.

A crowd-sensing alkalmazások egyik fő előnye, hogy olyan nagy-kiterjedésű érzékelő teret hoz létre, amely más módszerekkel nagyon drága, vagy időigényes lenne. Gondoljunk például egy celluláris lefedettségi térkép elkészítésére. Ennek az elkészítése hagyományos módon igen drága: egy műszerekkel felszerelt mérőautónak kell járnia a mérési területet. Ezeknek az autóknak a fenntartása költséges, ezért csak hálózathálózatos fejlesztés után, vagy problémás területek vizsgálatára használják. Előnye, hogy ahol méréseket végez, ott nagyon pontos adatokat szolgáltat a rádiós viszonyokról, hátránya a magas költség, a kis lefedettség és a nagy késleltetés. Köztes megoldásként ezt a feladatot taxisoknak szervezik ki, ekkor a jármű fenntartásának a költségeit a taxis állja és csak az eszköz üzemeltetése terheli a mérést végző egyént. Ez a megközelítés már közelebb áll a crowd-sensinghez, de még mindig rosszul skálázódik, mivel az eszközök karbantartásának költségei mellett a résztvevőket is meg kell fizetni. Crowd-sensing alapon viszont egy megfelelő, mobil eszközre telepített alkalmazással igen gyorsan, és esetleg ingyen lehet információt gyűjteni a felhasználók által biztosított mérési eredményekből. Egyrészt tehát időt és költségeket takaríthatunk meg, másrészt pedig sokkal szélesebb körben vagyunk képesek érzékelni a környezetünket, hisz akár az egész hálózatra kiterjedő méréseket is végezhetünk.

Hátránya lehet, hogy a közösség tagjai valamilyen oknál fogva hibás mérést végeznek. Történhet ez az eszköz érzékelőjének meghibásodásából, vagy az egyén nem-hozzáértéséből fakadóan is. Ezeknek a mérési hibáknak a kiszűrésére figyelni kell, hiszen képesek teljesen elrontani a környezetről kialakítandó pontos képet. Másik hátránya lehet még a közösségi tagok „kárpótlása” vagy motiválása a mérések elvégzéséért cserébe. A közösségi adatszolgáltatás sokszor ingyen is működik, de olykor felteszi magának a kérdést a felhasználó, hogy miért éri meg neki ingyen adatokat szolgáltatni. Ezeket a felhasználói igényeket meghatározott szolgáltatásokkal lehet ellensúlyozni, például úgy, hogy a közösség által összegyűjtött információk egy részét, vagy egészét strukturáltan nyilvánosságra hozzuk a számukra. További hátránya még a crowd-sensing érzékelésnek, hogy a mérések helye és ideje nem irányított, így nem garantálható, hogy teljes lefedettséget kapunk egy adott problémáról.

2.1 Mobil Crowd-sensing alkalmazások a piacon

A mobil crowd-sensing alkalmazások három szempont szerint csoportosíthatóak. Az egyik, hogy milyen típusú értékeket mér az alkalmazás, második, hogy szükséges-e a méréshez a felhasználó aktív közreműködése, illetve a harmadik szempont, hogy a mérések eredményeiből a felhasználó profitál-e közvetlenül az alkalmazáson keresztül. Az alkalmazás által mért adatok típusa lehet környezeti, infrastrukturális, és szociális. Ezek alapján tekintsünk át most néhány példát.

Az első típus a környezeti érzékelést végző alkalmazások, amelyek a környezet jelenségeit vizsgálják. Ezek az eszközök rendelkeznek különféle szenzorokkal, amelyek a környezet olyan tulajdonságait mérik, mint a levegő szennyezettség, egy folyó vízszintjének alakulása, a szeizmikus hatások, vagy különböző meteorológiai paraméterek, mint például a szélerősség, hőmérséklet, páratartalom, légnyomás vagy UV sugárzás erőssége. A CreekWatch egy olyan környezeti érzékelést végző alkalmazás [5], amely a folyók vízszintjét, és tisztaságát méri. Ezek a mérések olyan egyénektől származnak, akik képet készítettek a folyó valamely szakaszáról, vagy szöveges véleményt írtak a folyónál található szemétmennyiségről, amelyet utána fel is töltöttek egy központi adatbázisba. Ezek az információk hasznosak lehetnek a környezeti igazgatóság számára a környezet megóvása érdekében. További jól ismert alkalmazás, a sokak által ismert, magyar fejlesztésű Időkép [6]. Az általuk szolgáltatott mérőeszközök lehetővé teszik, hogy bárki saját meteorológiai mérést végezhesen. Ezek az eszközök mérik a csapadékmennyiséget, a szél sebességét és irányát, hőmérsékletet, páratartalmat, és légnyomást. Az egyéni méréseknek köszönhetően sokkal több információt gyűjthetünk országunk pontos időjárási körülményeiről, illetve jobb és pontosabb előrejelzéseket tehetünk a nagyobb információ birtokában. Ugyanakkor fontos megjegyezni, hogy a mérési eredmények hitelessége ebben az esetben is különösen fontos.

A második típusú alkalmazások az infrastrukturális mérő-alkalmazások, amelyek például a torlódási, útminőségi, parkolási és egyéb infrastrukturális információkat gyűjtik össze. Korai fejlesztések például a CarTel [7] és a Nericell [8]. Az MIT kutatócsoportja által fejlesztett CarTel egy olyan speciális eszköz, amelyet egy autóba telepítenek és méri annak helyzetét, sebességét. A mérési eredményeket nyilvános WiFi hozzáférési ponton keresztül küldi el egy központi szervernek, ahol feldolgozzák, és az összegyűjtött információk alapján fel lehet térképezni a forgalmi

helyzetet egy adott környéken. A Microsoft által fejlesztett Nericell pedig egy olyan mobil alkalmazás, amely nem csak az átlagsebességet rögzíti és a forgalmi helyzetet térképezi fel, hanem például a dudálási szintet (különösen olyan országokban, mint India, ahol sokat dudálnak) vagy az úton fekvő nagyobb kátyúkat. Ennek a projektnek a keretein belül igyekeztek úgy fejleszteni az alkalmazásokat, hogy a szenzorok által használt erőforrásokat a lehető legoptimálisabb szinten használják ki, ezzel is törekedve az energiatakarékosságra. Ennek egy megvalósítása például, hogy a gyorsulás-mérő szenzort csak egy megadott zaj-szint (például dudálás) felett kapcsolják be. Ezeknek a fejlesztéseknek köszönhetően sok hasznos információt gyűjthetünk arról, hogy hogyan érdemes ezeket a szenzor-hálózatokat optimálisan és energiatakarékosan üzemeltetni. Következő hasonló alkalmazás a Rutgers egyetemen működő WinLab csoport által fejlesztett ParkNet [9], amely a város különböző parkolási pontjairól szolgáltat információkat. A méréseket végző készülékek szintén okos-telefonok, melyek segítségével megjelölhető, hogy egy adott helyen van-e parkolási lehetőség egy megadott időpontban. Az informatikához közelebb álló megoldás például az OpenSignal Mobile App [10], amely hálózati méréseket végez. Feltérképezi az elérhető WiFi-hálózatokat, rögzíti a celluláris hálózat jelszintjét és így egy lefedettségi térképet készít egy adott területről, amelyet weboldalon keresztül tesz böngészhetővé. A méréseket egyszerű felhasználók végzik, akik a telefonjukat használva és mozogva folyamatosan mérik az elérhető jelszinteket. A mérési eredményeket egy központi adatbázisba gyűjtik, majd különböző grafikonokon, térképeken jelenítik meg az összevont mérési adatokat.

A harmadik típus a szociális crowd-sensing alkalmazások. Ezekben az alkalmazásokban a mérés tárgya az eszközt használó személy, ezért fontos a mért adatok kellő védelemmel való ellátása, hogy illetéktelenül ne lehessen hozzáférni. Ezek az alkalmazások például a felhasználók mozgási, vásárlási, utazási szokásait rögzíthetik, vagy esetleg hogy mennyi időt töltenek bizonyos feladatokkal a nap folyamán. Ehhez hasonló fejlesztés a DietSense [11] alkalmazás, amely lehetővé teszi, hogy a felhasználók lefényképezzék az ételt, amit esznek, majd ezt megosztva láthatják, hogy mások milyen ételeket, milyen rendszerességgel fogyasztanak. Ennek gyakorlati haszna például a cukorbetegség között tapasztalható.

3 DroidLab rendszer bemutatása

Ebben a fejezetben a DroidLab rendszer-architektúra követelményeit, a használt platformokat, majd a komponenseket és azok működését, illetve a főbb kommunikációs csatornákat mutatom be.

Az architektúra tervezése a témát vezető konzulenssel közösen történt, figyelve a használt platformok jellemzőire és sajátosságaira. A következőkben bemutatott rendszer a végleges, elkészített alkalmazást tükrözi.

3.1 Rendszerrel szemben támasztott követelmények

A rendszer tervezésekor több tervezési szempontot is figyelembe kellett venni. Egyrészt a platformok által nyújtott bővíthetőségi és biztonsági megfontolásokat, illetve hogy mennyire naprakész az adott technológia. Ezen tervezési követelmények vizsgálatát alább fejtettem ki.

3.1.1 Bővíthetőség

A projekt egyik fontos tulajdonsága, hogy a kutatók saját modul írásával képesek legyen saját kódot futtatni egy adott eszközön. Ezek a modulok a készülékek képességei és rendelkezésre álló erőforrásai alapján kerülnek szétosztásra a szerver és a felhasználói készülékek között, majd a keretrendszerbe töltődve futtathatóvá válnak. Másik fontos tulajdonság, hogy egyes fejlesztők új mérő-plugineket fejleszthetnek, ezzel is bővítve a mérési lehetőségeket. Tehát a keretrendszertől függetlenül képesnek kell lennünk új modulokat és új plugin-alkalmazásokat készíteni anélkül, hogy az eszközre települt keretrendszer változna.

3.1.2 Biztonsági megfontolások

A bővíthetőség részben javítja, részben viszont rontja az alkalmazás biztonságát. Mivel az egyes mérő-eszközök egyben a felhasználók saját készülékei, ezért fontos, hogy csak azokhoz az adatokhoz férjen hozzá az alkalmazás, amely feltétlen szükséges a működéshez, és amelyeket a felhasználó a telepítéskor engedélyezett. A keretrendszer-alkalmazásnak minimális engedélyre van szüksége a futáshoz, mivel nem végez mérést, így nem kell hozzáférnie a különböző személyes adatokhoz és hardware-

komponensekhez. A külön telepíthető plugin-alkalmazások rendelkeznek a mérés-specifikus engedélyekkel, így azok a szerint telepíthetők, hogy mennyit szeretne megosztani a felhasználó a saját készülékéből. A másik biztonsági pontja az alkalmazásnak a lefordított modul betöltése az alkalmazásba. Ezt az eljárást sandboxingnak [12] nevezzük. A modul kódja ártalmas kódot is tartalmazhat, így fontos, hogy megfelelő védelem legyen a modul és a keretrendszer között. További biztonsági pont lehet az alkalmazás és a web-szerver közötti kommunikáció is. Bizalmas adatok esetén fontos, hogy a kommunikációs csatorna kellő védelemmel legyen ellátva.

3.1.3 Naprakész

A rendszer tervezése során olyan platformokat és technológiákat választottunk, amelyek megfelelnek a kor követelményeinek és elvárásainak. Fontosnak tartottuk, hogy olyan platformokon fejlesszünk, amelyek naprakészek, kellően elterjedtek, ismertek és rendelkeznek megfelelő dokumentációval. A továbbiakban ezeknek a bemutatása következik.

3.2 Android

A mai mobil iparág legelterjedtebb okostelefon operációs rendszere az Android [13]. A 2009-es év második felében a készülékek 2.8%-át szállították Android operációs rendszerrel. 2010 végére ez 33%-ra nőtt, mellyel kivívta az első helyet a többi operációs rendszerrel szemben. A Gartner által végzett felmérések alapján a 2011-es év harmadik felére már az 52,5%, majd ezzel a lendülettel tovább haladva 2012 végére a 72%-ot is elérte az android piaci részesedése. Az alábbi ábrán egy táblázatot láthatunk a jelenleg elérhető okos-telefonokon futó operációs rendszerek piaci helyzetéről.

Operációs rendszer	2013 Q3 Darab	2013 Q3 Piaci rész	2012 Q3 Darab	2012 Q3 Piaci rész
Android	205,022.7	81.9%	124,552.3	72.6%
iOS	30,330.0	12.1%	24,620.3	14.3%
Microsoft	8,912.3	3.6%	3,993.6	2.3%
BlackBerry	4,400.7	1.8%	8,946.8	5.3%
Bada	633.3	0.3%	4,454.7	2.6%
Symbian	457.5	0.2%	4,401.3	2.6%
Egyéb	475.2	0.2%	683.7	0.4%
Összesen	250,231.7	100.0%	171,652.7	100.0%

1. ábra – Okos-telefonok eladási statisztikája operációs rendszer szerint.

Forrás: Gartner (November 2013)

A mobil eszközön rendszerint egy módosított Linux kernel található, amely elfedi a hardware-specifikus elemeket. Felette találunk különböző segéd-könyvtárakat, illetve a Dalvik virtuális gépet, amely az alkalmazás futtatását végzi. A felsőbb rétegekben pedig a készülék egyes funkcióihoz való hozzáférést biztosító könyvtárakat találjuk. Az alábbi ábra az android platform egyszerűsített szoftver-architektúráját mutatja be.



2. ábra - Az android platform szoftver-architektúrája

Az alkalmazások fejlesztése Java környezetben történik. Az Android SDK [14] biztosít megfelelő könyvtárakat a fejlesztéshez, amely ingyenesen elérhető és kellő dokumentációval is rendelkezik. A fejlesztői környezetet a szintén ingyenesen elérhető Android Developer Tools [15] biztosítja, amely sok hasznos android-specifikus funkcióval bővített, Eclipse alapú fejlesztői felület. Az Android operációs rendszert futtató eszközök kellő számú és megbízható szenzorral és interfésszel rendelkeznek ahhoz, hogy általuk környezeti méréseket lehessen végezni. Ilyenek például a celluláris, WiFi, Bluetooth, GPS interfészek vizsgálata, vagy a felhasználói interakciók megfigyelése. Az android fejlesztésekor fontos szerepet kapott a biztonság is. Minden telepített alkalmazás jogosultságokat kíván a felhasználtól, ami azért fontos, hogy az alkalmazás futásakor biztosak lehessünk abban, hogy csak a megadott jogosultságoknak megfelelő adatokhoz férhet hozzá. A DroidLab architektúra tervezésekor ezt a biztonsági mechanizmust is figyelembe kellett venni, és ennek megfelelően lettek külön alkalmazásban megvalósítva az egyes méréseket végző plugin-ek. További fontos kérdés az egyes komponensek közötti kommunikációs csatornák biztonsága. Az Android lehetőséget nyújt az alkalmazások megbízható kommunikációjára a broadcast üzenetek segítségével. Ugyanakkor fontos meghatározni, hogy melyik osztályok kaphatják meg az adott broadcast üzenetet, nehogy egy kártékony alkalmazás beregisztrálva az üzenetre, hozzáférhessen bizalmas adatokhoz. Ennek megoldására szintén lehetőséget biztosít a platform. Az Android tehát megfelelően bizonyult ahhoz, hogy először erre implementáljuk a DroidLab alkalmazást, mivel az okos-telefonokon

futó operációs rendszerek között a legelterjedtebb, és az alkalmazás által elvárt biztonsági követelményeket is teljesíti. Fejlesztése egyszerű, naprakész technológia, illetve sok dokumentáció és fejlesztési tapasztalat áll rendelkezésre. Az android operációs rendszer terjedésének köszönhetően később már nem csak mobil eszközökön, hanem akár más készüléken is elérhetővé válhat a mérő-alkalmazás.

3.3 Google App Engine

A mai kor elvárásainak megfelelően szükségesnek láttuk egy olyan technológia használatát, mint amilyen a Google App Engine [16] (továbbiakban: GAE). Ennek segítségével felhő alapú, jól skálázható, robosztus web-alkalmazásokat fejleszthetünk, és futtathatunk a Google saját infrastruktúráján. Ezeket az alkalmazásokat könnyű fejleszteni, karbantartani, bővíteni, ahogyan a forgalom vagy az adatmennyiség növekszik.

A GAE többféle programozási nyelvet is támogat. A széles körben elterjedt és használt Java platform, Python, PHP környezet CloudSQL támogatással, és a kevésbé ismert GO platform. Csatatunk a Java platform mellett döntött, mivel elterjedt és kellő támogatással is rendelkezik. Az erőforrások használata egy bizonyos korlátig ingyenes, így számunkra a fejlesztés és tesztelés szakaszában megfelelőnek bizonyult ez a technológia.

Adattárolásra kétféle lehetőség is adott: a DataStore [17] és a BlobStore [18]. A DataStore segítségével különböző típusú entitásokat tárolhatunk anélkül, hogy előtte bármilyen adatstruktúrát rögzítettünk volna. A szerver entitásokat küld az adattár felé, amely az azonos típusú entitásokat azonos táblába teszi. Az egyes attribútumokra semmilyen megkötés nincs. A BlobStore fájlok tárolására ad lehetőséget rendkívül egyszerű módon.

A Google App Engine előnye, hogy nincs szükség saját eszközökre a web-alkalmazás futtatásához, így nem kell a szerverek üzemeltetésére és karbantartására időt és pénzt költeni. Mindezek mellett a GAE rendelkezik olyan beépített funkciókkal, amelyek segítik a mobil eszközök felé irányuló kommunikációt. Kész technológiák állnak a rendelkezésünkre olyan web-alkalmazások fejlesztésére, amelyek megállnak napjaink elvárásaival szemben. Ugyanakkor meg kell említeni, hogy a szolgáltatás használata csak bizonyos korlátig ingyenes. A meghatározott korlát után az erőforrások használatával arányosan kell fizetni. A legdrágább erőforrások a Front-end-instance-

hour, és a DataStore-read. Mivel az alkalmazásnak nem kell sokat számolnia szerveroldalon, illetve nem fog sok adatot aggregálni, ezért ezekből az erőforrásokból minimális használatra lesz szükség az alkalmazás futásához.

3.3.1 DataStore

A Google App Engine által biztosított DataStore szolgáltatás az adatok struktúrában való tárolását teszi lehetővé. Az SQL-ben megszokott táblák helyett entitás-típusok vannak. Egy entitás-típus tetszőleges számú és típusú mezőt tartalmazhat, és később az adatbázis változtatása nélkül lehet új mezőket felvenni az entitáshoz. Ezért ez a megoldás meglehetősen rugalmas és robusztus. A DataStore által biztosított szolgáltatásokat a DatastoreServiceFactory osztály `getDatastoreService()` függvényének segítségével vehetjük igénybe. Az adatbázisban történő lekérdezésekhez a Query osztály áll rendelkezésre, amelyet a DataStoreService `prepare` metódusának kell átadni.

3.3.2 BlobStore

A BlobStore fájlok tárolását teszi lehetővé. Új fájl feltöltése történhet JSP oldalon vagy szervlet-en keresztül is. A fájlok elérését csak a fájlhoz tartozó blob-azonosító teszi lehetővé, amelyet a sikeres feltöltés után kapunk meg. A fájl további elérése érdekében fontos ezt az azonosítót elmenteni egy adatbázisba, amellyel egyértelműen azonosítani tudjuk, hogy melyik azonosító melyik fájlhoz tartozik. A BlobStore eléréshez a BlobstoreServiceFactory osztály `getBlobstoreService()` függvénye által biztosított szolgáltatás ad megfelelő metódusokat.

3.3.3 Felhasználó-kezelés

A Google App Engine lehetőséget biztosít a felhasználók kezelésére is. A felhasználók azonosítására három lehetőség is adott. A Google azonosítási mechanizmus [19], illetve az alkalmazásunk vagy az OpenID által biztosított azonosítási rendszer használata.

A felhasználók bejelentkezés után elérhetik a weboldalakat, amelyeken keresztül meghatározott interakcióra képesek a rendszerrel. A Google App Engine lehetővé teszi, hogy bizonyos URL alá tartozó oldalakat egy meghatározott védelmi szinttel lássa el.

Ezeket a védelmi szinteket szerepeknek nevezzük. A védelmi szerep megadását a web.xml fájlban tehetjük meg egy új security-constraint hozzáadásával. Ebben megadhatjuk, hogy melyik oldalakat milyen szerepű felhasználók tekinthetik meg. Az azonosítást a Google saját azonosító-rendszere végzi a számunkra. A bejelentkezett felhasználó e-mail címe és beceneve érhető el a számunkra.

3.3.4 Szervlet

A Java szervletek [20] java nyelvű osztályok megvalósításai, amelyek a HttpServlet osztályból származnak le. Ezek az osztályok használatosak HTTP kérések feldolgozására. Felüldefiniálható függvényei például a gyakran használt doPost és doGet metódusok, amelyek a beérkező GET és POST kéréseket kezelik le. Ezekben a szervletekben többnyire adatbázis műveleteket, és a beérkező paraméterek feldolgozását lehet végezni.

3.3.5 Java Server Pages

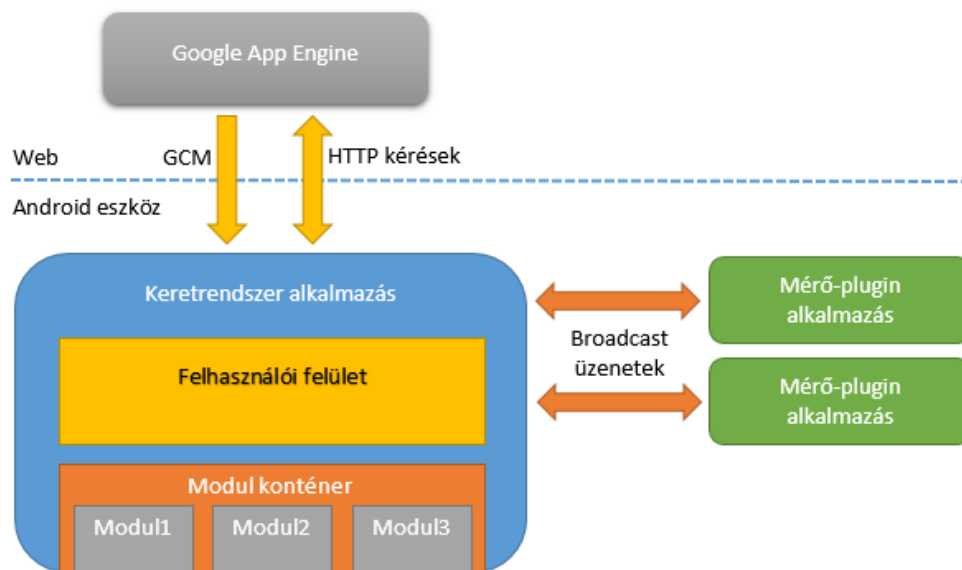
A Java Server Pages [21] (továbbiakban: JSP) egy olyan technológia, melynek segítségével dinamikusan generálhatunk HTML, XML vagy egyéb dokumentumokat HTTP kérésekre reagálva. A JSP tekinthető a szervlet réteg feletti absztrakciós szintnek, amelyből Java servlet forráskód generálódik.

3.4 Google Cloud Messaging

A Google Cloud Messaging [22] (továbbiakban: GCM) egy olyan szolgáltatás, melynek segítségével üzenetet lehet küldeni a szerverről a felhasználók eszközeire. Ez azért kényelmes megoldás, mert nem kell az alkalmazásnak folyamatosan futnia és a szervert lekérdezni egy új telepítendő modul esetén. Az üzenet fogadásáért az Android operációs rendszer felel, és ha az alkalmazás nem fut, a rendszer felébreszti az alkalmazást és kézbesíti az üzenetet feltéve, ha az alkalmazás megfelelően regisztrált a broadcast üzenetre és rendelkezik a megfelelő jogosultságokkal.

3.5 Rendszer-architektúra

A rendszernek négy fő komponense van. A keretrendszer-alkalmazás, a futtatandó modulok, a méréseket végző plugin-alkalmazások, és a webszerver. Az alább látható képen egy egyszerűsített ábra mutatja be a rendszer-komponenseket és azok kapcsolatait.



3. ábra - Rendszer-architektúra

3.5.1 Keretrendszer

A keretrendszer alkalmazás az architektúra központi eleme. Fő feladata, hogy felhasználói felületet biztosítson az alkalmazás kezeléséhez, kommunikáljon a webszerverrel és a telepített pluginek-kel, továbbá betöltse a modul-t, amelyet majd futtat a mérés elvégzéséhez. Az alkalmazás településekor csak minimális engedélyekre van szüksége, mivel a mérés-specifikus engedélyeket a plugin-alkalmazások tartalmazzák.

3.5.2 Modul

A modulok a futtatandó mérés kódját tartalmazzák, amelyek a webszerverről kerülnek letöltésre a mérő-eszközre. Egy modul rendszerint egy jar és egy desc kiterjesztésű fájlból áll. A jar fájlban a kutató által írt programkód, míg a desc fájlban az adott modul neve, osztályneve és a java osztályt tartalmazó csomag neve található. A letöltést követően a keretrendszer betölti a modulokat, amelyek futtathatóvá válnak. A mérés lefutása után a mérési eredményeket egy szöveges fájlba írják, amelyeket a keretrendszer bizonyos időközönként feltölt a webszerverre.

3.5.3 Plugin

A plugin alkalmazások tetszés szerint telepíthetőek, amelyek különféle méréseket képesek elvégezni az adott eszközön. A modulok a telepített plugin-alkalmazás által szolgáltatott metódusokat képesek meghívni. Ezek az alkalmazások

rendelkezik a megfelelő mérés-specifikus engedéllyel, így a felhasználónak csak azon plugin-alkalmazásokat kell telepítenie az eszközére, amely interfészekről információkat szeretne megosztani. Más információkhoz a keretrendszer-alkalmazás és a plugin-alkalmazások nem férhetnek hozzá. Annak érdekében, hogy a plugin metódushívásakor ne futhasson kártékony kód, az új plugin-alkalmazásokat csak az erre a feladatra kijelölt, úgynevezett plugin-fejlesztők készíthetik és az ehhez szükséges könyvtárak forráskódja sincs közzétéve.

3.5.4 Web-szerver

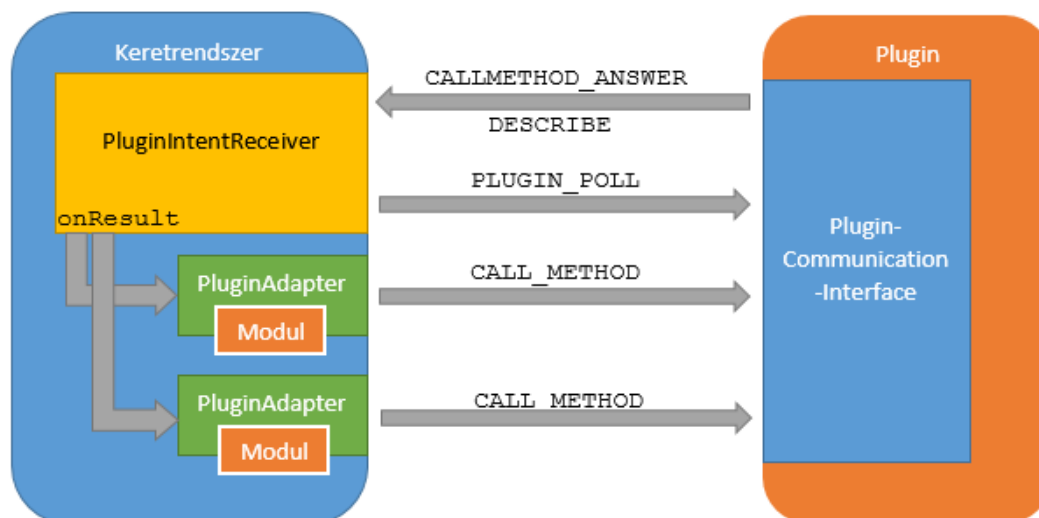
A web-szerver fő feladata, hogy tárolja a felhasználói adatokat, modulokat, plugineket, mérési eredményeket és a regisztrált mérő-eszközöket. A kutatók képesek új modulok feltöltésére, amelyekkel új méréseket végezhetnek. Az eszközök regisztrálják magukat a web-szerveren, így elérhetővé válnak az adott mérő-eszközök specifikációi, illetve értesíthetőek, ha új modult szeretnének rajtuk futtatni. A mérési eredmények is a szerverre kerülnek feltöltésre, amelyek kilistázhatóak és letölthetőek. Továbbá lehetőség van plugin-ek regisztrálására is, így az alkalmazás a szerverről is lekérheti az éppen aktuális plugin-alkalmazások listáját.

3.6 Kommunikációs interfészek

Ebben a fejezetben az egyes komponensek közötti kommunikációt mutatom be. A keretrendszer és a plugin-ek között broadcast üzenetek, a keretrendszer és a web-szerver között pedig GCM illetve HTTP kérések mennek.

3.6.1 Keretrendszer és plugin közötti interfész

A keretrendszer alkalmazásban találhatóak a betöltött modulok, amelyek mindegyikéhez tartozik legalább egy PluginAdapter. A PluginIntentReceiver a mérési eredmények továbbításáért és a plugin-információk lekéréséért felel. Ebből az osztályból csak egy található a keretrendszerben. A pluginek kommunikációjáért felel a PluginCommunication interfész, amely az összes pluginhoz érkező és onnan induló üzenetet kezeli. A következő ábrán a komponensekben található osztályok és azok közötti kommunikációs csatornák láthatóak. A továbbiakban ezeknek az osztályoknak a bemutatása olvasható.



4. ábra - Kommunikációs interfész a keretrendszer és a plugin között

3.6.1.1 PluginIntentReceiver

Ez az osztály a keretrendszerben valósítja meg a kommunikációs interfészt. Mivel az alkalmazások között broadcast üzenetek mennek, ezért a BroadcastReceiver osztályból származik le. Minden pluginról érkező üzenetet megkap. A beérkező üzeneteknek több fajtája van, attól függően, hogy milyen adatokat hordoznak.

Az első ilyen üzenet a metódushívásra érkező válasz, amely az `INTENT_ACTION_CALL_METHOD_ANSWER`. Ebben az üzenetben paraméterül megy a `CALL_ID`, amely a hívás azonosítóját tartalmazza, `PLUGIN_ID`, amely a plugin egyedi azonosítója, `VERSION`, amely a plugin verzióját mondja meg, `METHOD_NAME`, amely a meghívott metódus nevét mutatja meg és a `VALUE_RESULT`, amely az eredményt tartalmazza szöveges formátumban. Az eredményt továbbítja a `PluginResultListener`-ként beregisztrált `PluginAdapter` objektumoknak.

A következő broadcast üzenet neve az `INTENT_ACTION_DESCRIBE`. Ennek az üzenetnek két alfajtája van, amelyek arra szolgálnak, hogy a pluginról információkat küldhessünk a keretrendszer felé. Az egyik típus a plugin meta-adatait hordozza a másik pedig hibajelzésre szolgál. Az üzenet típusát a `DESCRIBE_TYPE` paraméterrel adhatjuk meg.

A plugin meta-adatait tartalmazó broadcast üzenet neve az `INTENT_EXTRA_VALUE_REPORT`. Ebben a fajta üzenetben a plugin-információkkal kapcsolatos paraméterek szerepelnek. A `PLUGIN_ID`, amely az adott plugin egyedi azonosítóját tartalmazza, `PLUGIN_AUTHOR`, amely a plugin készítőjét azonosítja,

DESCRIPTION, ami rövid leírást ad az adott pluginról, VERSION, amely a verziószámot írja le. A PLUGIN_METHODS, illetve a PLUGIN_EVENTS paraméterek pedig a plugin által szolgáltatott mérési függvények és események listáját tartalmazzák.

A plugin-ban történő hibák jelzésére szolgáló broadcast üzenet az INTENT_EXTRA_VALUE_ERROR. Ezek például a méréshez szükséges erőforrások vagy interfészek meghibásodásakor keletkezhetnek.

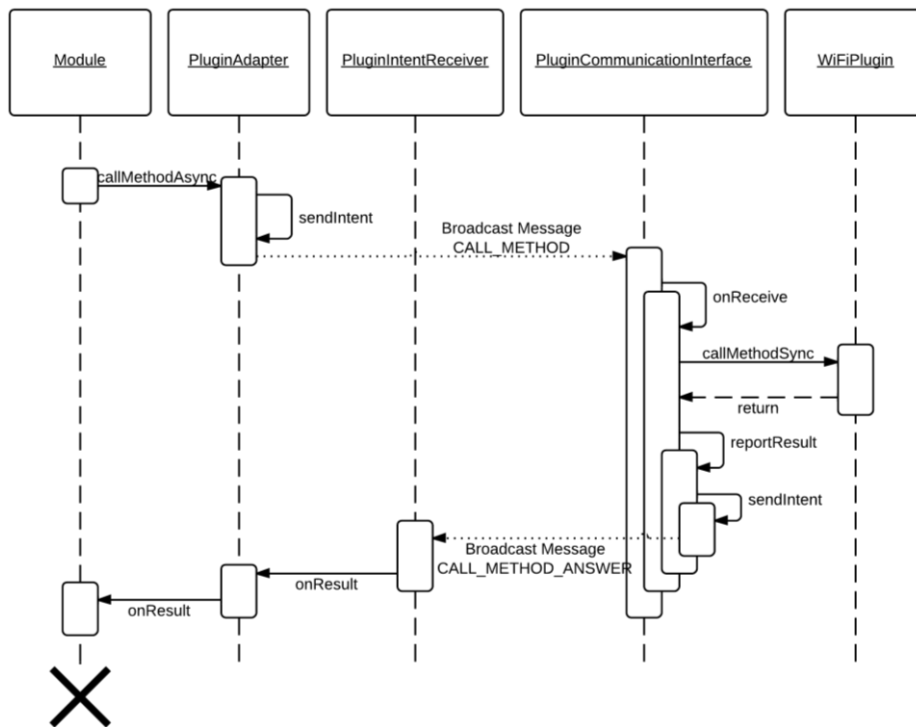
3.6.1.2 PluginCommunicationInterface

Ez az osztály a keretrendszer felől érkező broadcast üzeneteket kapja meg a plugin-alkalmazásban. A BroadcastReceiver osztályból származik le és megvalósítja a Plugin interfészt. A beérkező üzenet lehet metódushívás, illetve plugin-információk lekésére.

A plugin-információk lekérésére szolgáló broadcast üzenet az INTENT_ACTION_PLUGIN_POLL. Ez az üzenet az elérhető plugin-alkalmazások feltérképezésére szolgál. Paramétere nincs, a válasz az INTENT_ACTION_DESCRIBE üzenetben kerül elküldésre.

A metódushívások az INTENT_ACTION_CALL_METHOD broadcast üzenetben érkeznek. Ebben az üzenetben a metódushíváshoz szükséges információk találhatóak. Paraméterek között szerepel a CALL_ID, amely a hívás azonosítóját tartalmazza, METHOD_NAME, amely a hívott metódus nevét mondja meg, és végül a METHOD_PARAMETERS, amely a meghívott metódus paramétereit adja át.

A következő ábrán a modul metódushívásának teljes szekvencia-diagramját láthatjuk. A modul meghívja a hozzá tartozó PluginAdapter callMethodAsync metódusát, amely összeállítja a hívás paramétereiből a broadcast üzenetet, továbbá hozzárendel egy sorszámot a híváshoz. Majd ezt tovább küldi a plugin-ek felé, amelyek feldolgozzák a kérést. A PluginCommunicationInterface-en keresztül meghívódik a plugin-alkalmazásban implementált, mérést végző függvény, amely visszatér a mérés eredményével és a hívás azonosítójával. Az eredményt egy broadcast üzenetbe csomagolja és visszaküldi a keretrendszer felé. A PluginIntentReceiver osztály megkapja, majd továbbítja a megfelelő PluginAdapter felé, majd az továbbítja a saját modulja számára is.



5. ábra - Metódushívás szekvencia-diagramja

3.6.2 Keretrendszer és modul közötti interfész

A PluginAdapter a modul számára látható osztály, amin keresztül metódushívást lehet kezdeményezni a plugin felé. Ez az osztály megvalósítja a Plugin, PluginResultListener és a PluginEventListener interfészeket. Minden plugin alkalmazást egy ilyen PluginAdapter példány reprezentál a keretrendszerben a modul felé. A modul az AndroidPluginCollection osztályon keresztül kérheti le a jelenleg elérhető pluginok listáját. A PluginAdapter callMethodAsync metódusa egy `INTENT_ACTION_CALL_METHOD` broadcast üzenetet küld a pluginok felé, amely tartalmazza a metódushívás paramétereit. Minden metódushíváshoz tartozik egy azonosító, annak érdekében, hogy a visszaérkező eredményt hozzá lehessen rendelni. A PluginIntentReceiver osztálytól kapott mérési eredményeket továbbítja a nála beregisztrált modul felé.

3.6.3 Keretrendszer és web-szerver közötti interfész

Modulok letöltése

A GCM technológia segítségével értesítjük az alkalmazást egy új, letöltésre váró modulról. A GCM használatához az alkalmazásnak először regisztrálnia kell magát a szolgáltatásra. A regisztrációkor kap egy GCM azonosítót, melynek segítségével értesíthetjük a megadott eszközt. A web-szerver által küldött üzenetben a letöltendő modul elérési útja szerepel, amit az alkalmazás lehetőség szerint letölt a belső tárhelyre egy GET hívással.

Mérő-eszköz regisztrálása a szerveren

Az alkalmazás településekor az eszköz regisztrálja magát a szerveren. Egy POST üzenetben küldi el a web-szervernek a mérő-eszköz fontosabb tulajdonságait, mint például az IMEI szám, GCM azonosító, SDK verzió és hogy rendelkezik-e celluláris, WiFi, Bluetooth és GPS technológiával.

Mérési eredmények feltöltése

Az alkalmazás elérhető internetkapcsolat esetén feltölti a szerverre a mérési eredményeket zip formátumban, amelyet a kutatók letölthetnek. Az alkalmazás először elkéri a feltöltéshez szükséges URL-t egy GET hívással, majd utána egy POST hívással tölti fel a fájlokat a szerverre. A fájlok azonosítóit a web-szerver adatbázisában is tárolni kell.

Plugin adatok lekérése

Ahhoz, hogy a felhasználók értesülhessenek arról, hogy egy új vagy frissebb plugin-alkalmazás vált elérhetővé, a keretrendszer időnként letölti az elérhető plugin-alkalmazások meta-adatait és azok aktuális verzióját a web-szerverről. Ezek az adatok egy JSON struktúrában érkeznek meg a keretrendszerhez.

4 Rendszer használói

A rendszerben résztvevő egyéneket három nagy csoportba sorolhatjuk. A rendszer tényleges hasznélvezői a kutatók, akik méréseket végeznek a hálózatban. Felhasználóknak azokat nevezzük, akik megfelelő Android okostelefonnal rendelkeznek, és ezeket elérhetővé teszik a mérések számára. A fejlesztők pedig új mérő-alkalmazásokat, úgynevezett mérő-plugineket képesek fejleszteni. Az alábbiakban részletesebben foglalkozunk a rendszer használóival.

4.1 Kutatók

A kutatók modulok írásával tudnak új méréseket végezni. Az elkészült modult feltöltik a weboldalon keresztül, majd kiválasztják, hogy melyik modul melyik eszközre települjön. A szerver egy GCM üzenettel értesíti az eszközt az új futtatandó modulról, majd az eszköz lehetőség szerint letölti azt. A keretrendszer betölti az új modult, és futtatja, majd a mérési eredményeket feltölti a szerverre. A kutató számára listázva vannak a mérési eredmények, amelyeket letölthet a szerverről.

4.2 Felhasználók

A felhasználók eszközt biztosítanak a mérések számára. A keretrendszer önmagában nem képes méréseket végezni, így szükség van legalább egy plugin telepítésére a mérések végzéséhez. Minden plugin más és más interfészt vizsgál az eszközön, így a felhasználónak elég csak azt a plugin alkalmazást telepítenie, amelyik interfésztől adatokat szeretne biztosítani a mérések számára.

4.3 Core fejlesztők

A fejlesztők új mérő-pluginek tervezését és fejlesztését végzik, amelyek külön alkalmazásként telepíthetők az eszközökre. Az alkalmazások fejlesztése a tanszéki projekt keretein belül történik. A cél, hogy minél több plugin-alkalmazás készüljön, hogy a mérő-eszközök által nyújtott információkat elérhetővé tegyék a kutatók számára.

5 Megvalósítás

Ebben a fejezetben az általam elvégzett féléves munka kerül bemutatásra. Munkám során először megterveztem, majd megvalósítottam az egyes komponenseket, és az azok közti kommunikációs interfészeket.

Természetesen a korábban bemutatott és a projekt fejlesztése során használt platformokkal meg kellett ismerkednem, mielőtt az implementációra sor került volna. Az Önálló labor keretein belül már volt alkalmam Android rendszerre alkalmazást fejleszteni, illetve a PHP nyelvvel is foglalkoztam már. A Google App Engine platform viszont még teljesen új volt a számomra.

5.1 Teszt-plugin

A pluginek őssztálya egy külön projektben található, így új plugin létrehozásakor szükséges ennek a könyvtárnak az importálása is. Ez azért fontos, hogy a plugin interfész ne változzon attól, hogy egy fejlesztő új plugint tervez.

5.1.1 PluginCommunicationInterface

A PluginCommunicationInterface minden plugin őssztálya. Ez az osztály a BroadcastReceiver osztályból származik le. Két fő metódusa van: az `onReceive` és a `reportResult`. A keretrendszer felől érkező üzenetek az `onReceive` függvényt hívják meg, amelyben kétféle üzenettípust különböztetünk meg: `INTENT_ACTION_PLUGIN_POLL` és az `INTENT_ACTION_CALL_METHOD`.

INTENT_ACTION_PLUGIN_POLL

Ennek az üzenetnek a segítségével a keretrendszer feltérképezheti, hogy éppen milyen plugin-alkalmazások vannak telepítve az eszközre. Minden plugin egy broadcast üzenetben küldi vissza a saját tulajdonságait egy `INTENT_ACTION_DESCRIBE` üzenetként, amely tartalmazza a fontosabb plugin-információk leírását.

INTENT_ACTION_CALL_METHOD

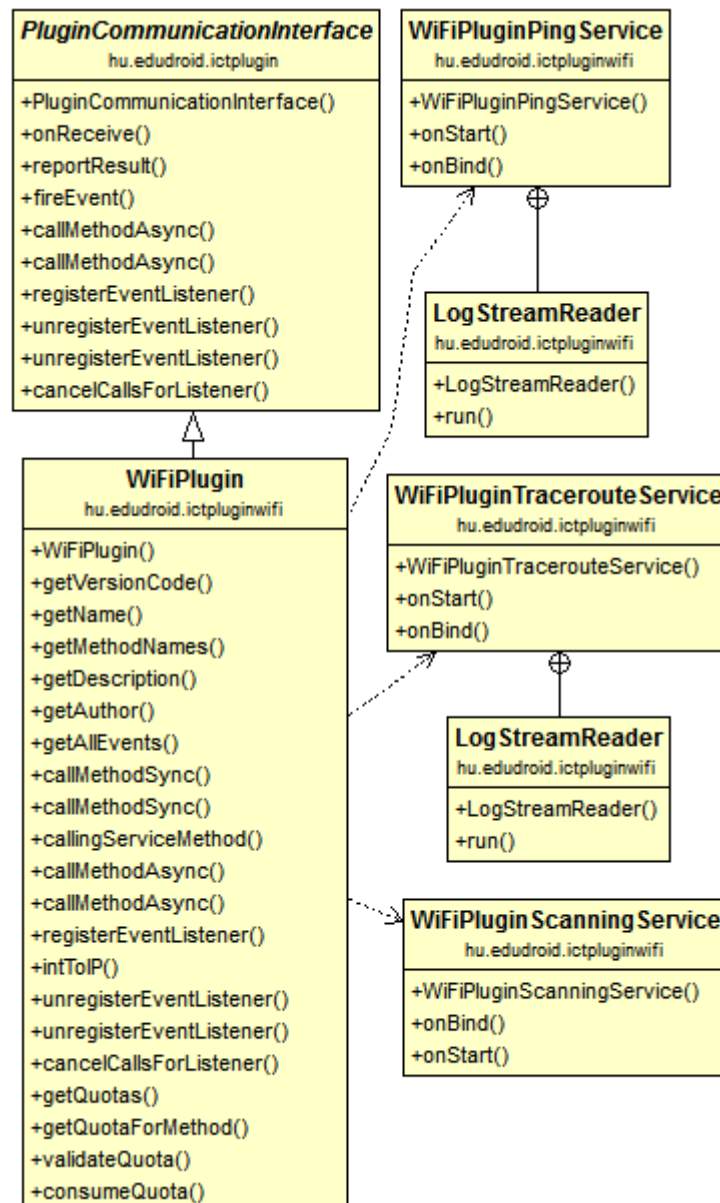
Ezzel az üzenettel lehet metódus-hívást kezdeményezni a plugin felé. Az üzenet tartalmazza a hívás azonosítóját, a metódus nevét, amelyet futtatni szeretnénk, és a metódus paramétereit. A válaszüzenetben a mérési eredményt küldjük vissza a

keretrendszer felé az `INTENT_ACTION_PLUGIN_CALLMETHOD_ANSWER` üzenettel.

5.1.2 WiFiPlugin

A rendszer tesztelésére egy példa plugin-t készítettem, amely hálózati méréseket készít a WiFi interfészen. A `WiFiPlugin` osztály a `PluginCommunicationInterface` osztályból származik le. A mérések megvalósítása a `callMethodSync` függvényben találhatóak. A mérni kívánt értékeket két fajtába sorolhatjuk. A pillanatnyilag elérhető állapotinformációk, és a hosszabb mérést igénylő információk. Pillanatnyi elérhető állapotinformációnak nevezzük azokat a mérési eredményeket, amelyek közvetlenül elérhetőek egy függvényhíváson keresztül. Ilyen például az eszköz által használt IP cím, vagy MAC cím. Ezek a függvények rögtön visszatérnek a megfelelő értékkel, így a választ egyből vissza is lehet küldeni a keretrendszernek. Mérést igénylő információknak nevezzük azokat a mérési eredményeket, amelyek megszerzéséhez egy külön szálon, hosszabb ideig tartó mérést kell végezni. Ezen mérések megvalósításához az Android által nyújtott `Service` szolgáltatást használtam.

Az általam készített plugin-ben három különböző háttérben futó mérést valósítottam meg. Szerver pingelés, traceroute futtatása, és elérhető WiFi hálózatok lekérése. Az Android API lehetőséget ad a fejlesztőknek parancssori hívások indítására, így a hívás kimeneteit stream-eken keresztül teszi elérhetővé. Ezzel a lehetőséggel élve valósítottam meg az említett két mérést. Az elérhető WiFi hálózatokat pedig az Android `WifiManager` osztályának segítségével kértem le. A következő ábrán a `WiFiPlugin` osztálydiagramját láthatjuk. Az ábrán megfigyelhető, hogy a hosszabb mérést igénylő feladatok külön `Service` osztályban vannak megvalósítva, amelyek többnyire a Linux rendszer felé intézett hívásokat kezelik.

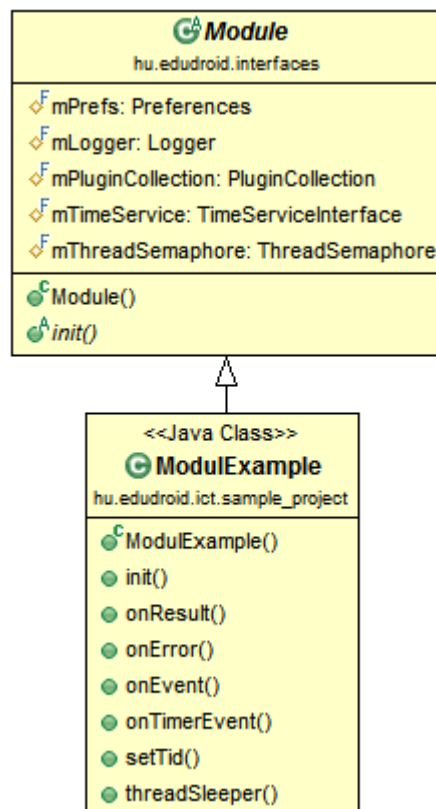


6. ábra – Plugin osztálydiagramja

5.2 Tesztmodul

Az absztrakt Module interfész megvalósítja PluginEventListener, PluginResultListener és a ModuleTimerListener interfészeket, továbbá tartalmaz két absztrakt metódust. Az init függvény a modul-inicializálást a run pedig a futtathatóságot valósítja meg. Feladataim közé tartozott egy példa modul létrehozása. Új modul létrehozásakor egy új java-projektet kell létrehozni, amelyben egyetlen osztály szerepel, amely a modult valósítja meg. Ennek az osztálynak a Module osztályból kell leszármaznia. A konstruktorban négy paramétert kap a modul: Preferences, Logger, PluginCollection és TimeServiceInterface. A

Preferences segítségével a modulhoz tartozó beállításokat lehet perzisztensen tárolni, például részeredményeket, állapotokat. A Logger a mérési eredmények fájlba írását teszi lehetővé. A log és a preferences közti különbség, hogy a logot a modul már nem olvashatja, a logok a mérés kimenetei. A PluginCollection az elérhető plugin-ek listáját szolgáltatja a modul számára, míg a TimeServiceInterface a modul időzítéséhez nyújt függvényeket. A modul az `init()` metódusban beállítja az időzítési paramétereket, illetve megszerzi a szükséges kommunikációs csatornákat az elérni kívánt plugin-ekhez a PluginCollection interfészen keresztül. A `run()` metódusban kell implementálni magát a mérést, amely a plugin egy-egy metódusát hívja meg. Végül a fentebb említett interfészek miatt a modul rendelkezik `onResult`, `onError`, `onEvent` és `onTimerEvent` metódusokkal is. Értelmszerűen a plugin-től vagy időzítőtől érkező események ide érkeznek be, majd ezekben történik meg az eredmények log-ba írása, illetve újabb mérések indítása vagy a futás időzítése.

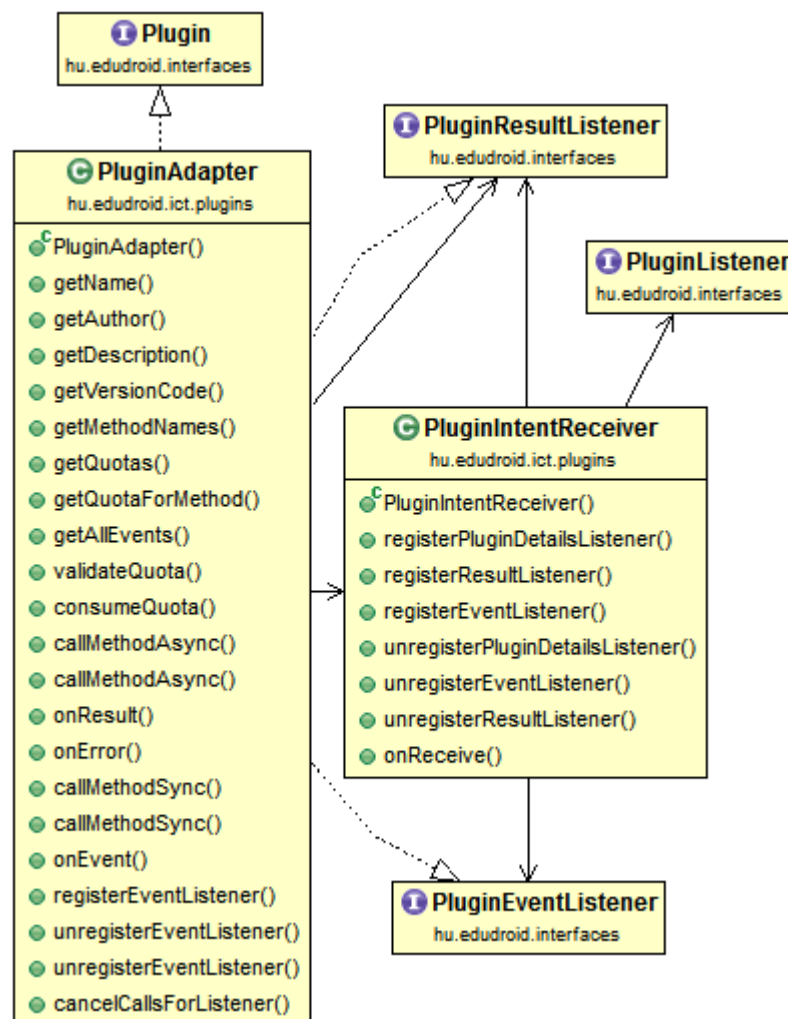


7. ábra - Modul osztálydiagramja

5.3 Keretrendszer és Plugin közötti interfész

PluginAdapter a keretrendszerbe

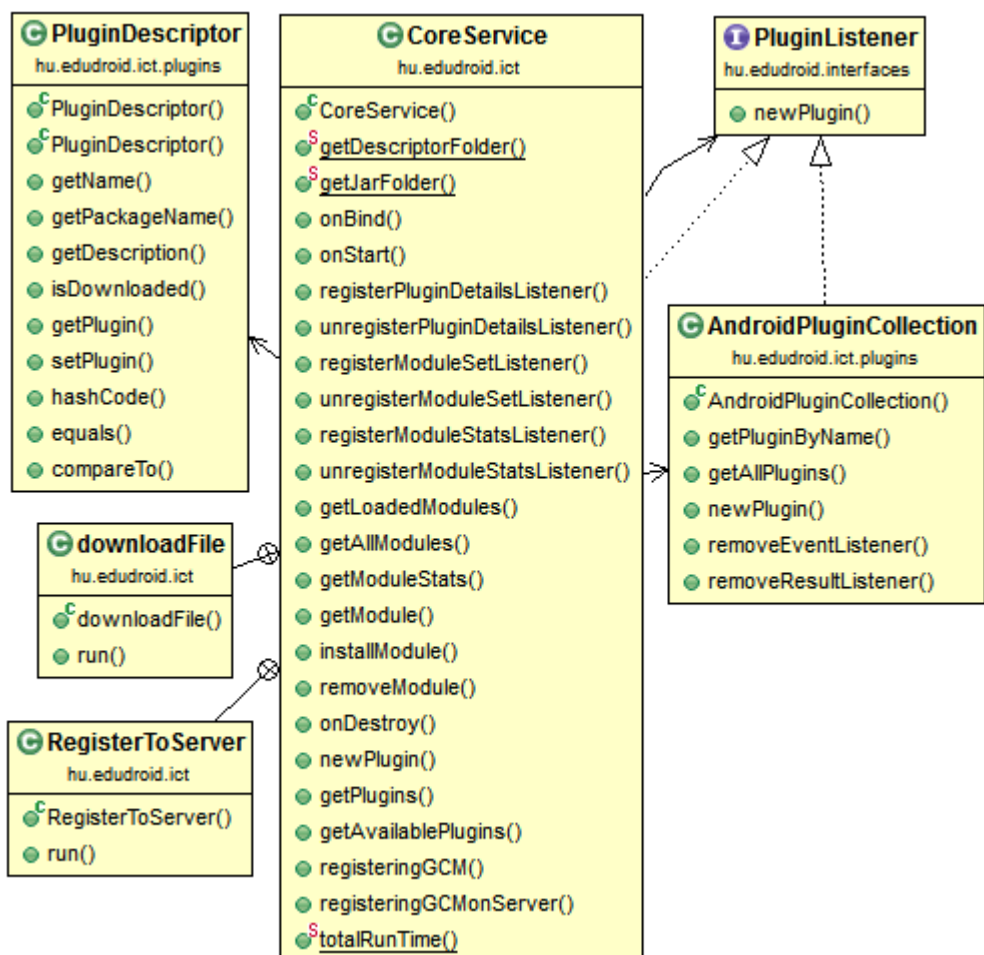
A keretrendszer felőli interfészt a PluginAdapter osztály valósítja meg a Plugin, PluginResultListener és PluginEventListener interfészek implementálásával. A callMethodAsync metódus először regisztrálja magát a válasz broadcast üzenetére, majd összeállítja a modultól kapott paraméterekből a plugin számára küldött `INTENT_ACTION_CALL_METHOD` broadcast üzenetet. A plugin válasza a PluginIntentReceiver osztályba érkezik meg, amely továbbadja a beregisztrált PluginAdapternek a mérési eredményeket, amely aztán továbbküldésre kerül a modulnak. Így tehát a PluginAdapter teremti kapcsolatot a modul és plugin között. Minden üzenetváltás ezen az osztályon keresztül történik. A következő ábrán a PluginAdapter és a PluginIntentReceiver osztály felépítését láthatjuk.



8. ábra - PluginAdapter és PluginIntentReceiver osztálydiagramja

CoreService létrehozása a keretrendszerben

A CoreService feladata, hogy háttérben futó feladatokat hajtson végre. Az alkalmazást az általam létrehozott BootingBroadcastReceiver osztály indítja el, amely regisztrálva van a BOOT_COMPLETED üzenetre. A végrehajtandó feladatok közé tartozik az alkalmazás elindulásakor történő modulbetöltés, telepített plugin-ek összegyűjtése broadcast üzenetek segítségével, regisztrálás a Google Cloud Messaging-re (továbbiakban GCM), az eszköz meta-adatainak frissítése a szerver felé, illetve az alkalmazások közti kommunikációhoz szükséges BroadcastReceiver-ek regisztrálása a keretrendszerben. Futás közben a modulok háttérben történő letöltését is végezheti. A CoreService osztálydiagramja az alábbi ábrán látható.



9. ábra - CoreService osztálydiagramja

Kommunikációs interfészek felállítása

A CoreService egyik feladata a megfelelő BroadcastReceiver-ek regisztrálása, másrészt itt történik a keretrendszer, a modul és plugin-ek közti kommunikációs pontok felépítése is.

Google Cloud Messaging regisztráció

Ezt a technológiát a szerver felől érkező üzenetek fogadásához használjuk. Ha egy kutató új modult szeretne futtatni az eszközön, akkor a szerver egy üzenetben közli az eszközzel, hogy a paraméterként megadott elérési útról töltsse le a modult, és futtassa, majd a mérési eredményeket töltsse fel a szerverre.

Meta-adatok frissítése a szerver felé

A mérőeszköz meghatározott időközönként frissíti a szerveren a saját meta-adatait. Ezek az információk azt tartalmazzák, hogy például rendelkezik-e az adott eszköz WiFi, GPS, Bluetooth, vagy celluláris interfésszel. Továbbá, hogy milyen verziójú Android operációs rendszer fut az eszközön. Ezen az interfészen a mérőeszköz tulajdonságait lehet továbbítani a szerver felé.

5.4 Modul ütemezése

A modulok futásának ütemezését a modul írója kérheti az `init` metódusban. Így a `run` függvényben megírt mérési sorozatot többször is megismételhető, bizonyos időközönként. A `ModuleTimeService` a `TimeServiceInterface`-nek megfelelően lehetőséget biztosít a modul egy konkrét időpontban történő, vagy periodikusan ismételt futtatására.

5.5 Module logger

A mérési eredményeket a modulok log-okba írják. Ehhez egy `AndroidLogger` nevű osztályt hoztam létre, amely a `Logger` interfészt valósítja meg. Megvalósítása nagyban hasonlít az Android által kínált logger-hez. Három különböző log-level hoztam létre: `error`, `debug` és `information`. Az `error` szint a méréskor fellépő error-üzenetek, a `debug` szint a fejlesztést segítő üzenetek, míg az `information` szint a mérési eredmények logba írását végzi. Minden modulnak saját `Logger`-re van, amelyet az inicializáláskor állít be magának. A log-ok a belső SD kártyára mentődnek az `ictdroidlab_log`

mappába, zip fájlként. A hálózati forgalom optimalizálása érdekében a logok csak egy bizonyos méret meghaladása után záródnak le, és kerülnek feltöltésre.

5.6 PHP web-szerver prototípus

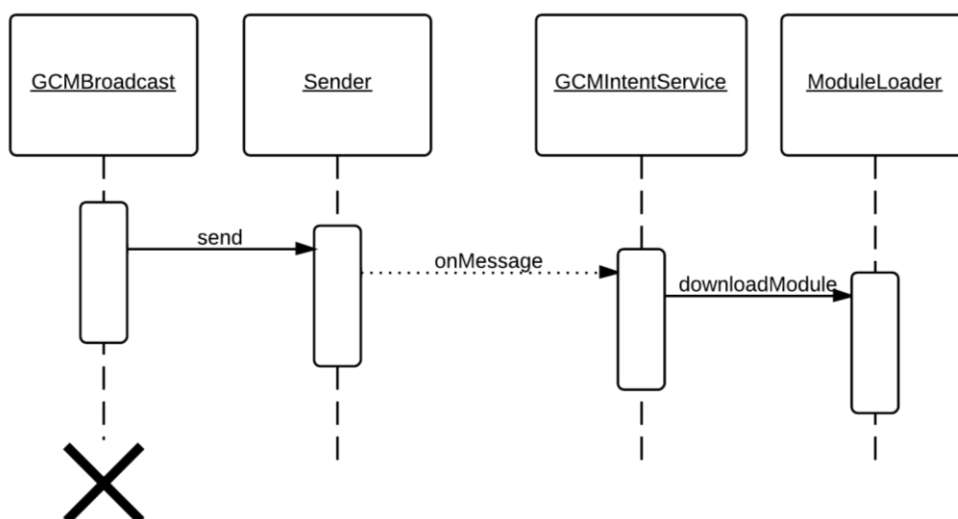
Feladatom közé tartozott a web-szerver kommunikációs interfészeinek megvalósítása és a felhasználói felület kialakítása az alapvető funkciók teszteléséhez. Munkám során először a PHP és MySQL alapú technológiával foglalkoztam. Olyan, már meglévő keretrendszert kerestem, amely képes a felhasználókat, a funkciókat és a feltöltött tartalmat megfelelően kezelni. Lehetőségek között szerepelt a Joomla, mint az egyik legnagyobb, ingyenesen is elérhető CMS (Content Management System) szolgáltatás. A projekt számára egy egyszerű, áttekinthető, és könnyen bővíthető rendszer kellett. A Joomla ennek az elvárásnak nem felelt meg. A sok további lehetőség mellett a WebsiteBaker nevű CMS mellett döntöttem, mivel megfelelt a fentebb említett elvárásoknak. Második lépésként a web-szerver beüzemelése következett, amelyet a tanszék biztosított. Az üzembe helyezés után még telepítésre került a phpmyadmin és a curl. A phpmyadmin webes felületet biztosít a szerveren futó mysql adatbázis kezeléséhez, a curl segítségével pedig HTTP kéréseket lehet generálni, amelyek szükségesek voltak a GCM megvalósításához. Ezek után megindult a weboldal fejlesztése PHP nyelven. A fejlesztés korai fázisában a PHP alapú webes keretrendszer gyorsan fejleszthető prototípus volt a web-szerveri funkciók teszteléséhez.

5.7 Felhő alapú web-szerver

Miután az alkalmazás hálózati modulját sikeresen teszteltem a PHP backend-del, munkámat a szerver GAE-re portolásával folytattam. Felhasználói interakcióra képes felületeket a Java Server Pages (JSP) segítségével hozhattam létre. Szerveren futó adatfeldolgozó alkalmazásokat pedig egyszerű java osztályokkal valósítottam meg úgy, hogy az osztályokat a `HttpServlet` osztályból származtattam le. Ennek két fő metódusát használtam, amelyek a GET és a POST hívásokat kezelik. A tartalmak elérhetőségének szabályozásához a Google saját azonosítási rendszerét használtam, így a CMS-ek által kínált tartalomkezelési szolgáltatást sikerült helyettesítenem. Ugyanakkor a projekt további feladatai közé tartozik a felhasználó- és tartalomkezelési mechanizmusok megtervezése és kialakítása.

GCMBroadcast Servlet

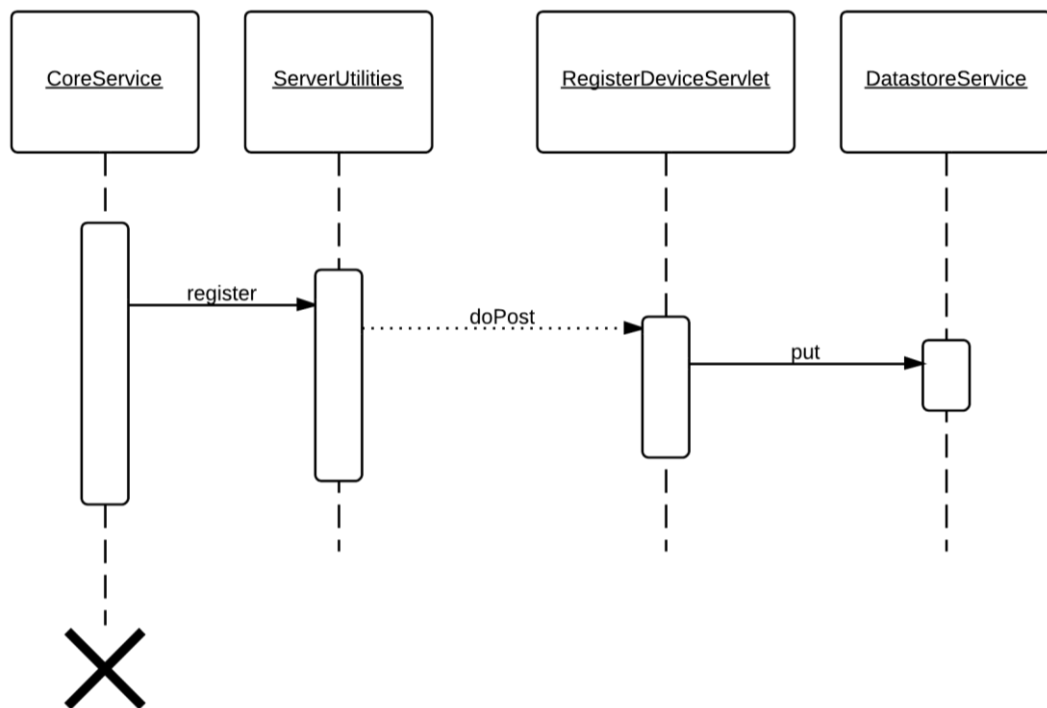
Ez a szervlet a GCM üzenetek küldéséért felelős. Paraméterként az értesítendő eszközök regisztrációs-azonosítóját és a letöltendő modul blob-azonosítóját kapja meg egy POST hívásban. A GCM üzenetben a letöltendő modul elérési útját adjuk meg, ahonnan az eszköz majd megpróbálja letölteni a modult. A címzettek pedig a paraméterként megkapott regisztrációs azonosítók listája. Az alábbi szekvencia-diagramon a modul letöltésének folyamata látható. A Sender osztály segítségével kiküldésre kerül a GCM üzenet, amely a keretrendszerbe érkezve meghívja a ModuleLoader downloadModule metódusát. Az üzenetben lévő URL címről pedig letöltődik a module az eszköz belső tárhelyére.



10. ábra - Modul letöltésének szekvencia-diagramja

RegisterDeviceServlet

Ennek a szervletnek a segítségével regisztrálhatnak az eszközök a szerverre. Az Android alkalmazás indulásakor egy POST hívásban küldi el az eszközhöz tartozó fontosabb értékeket, amelyeket a DataStore-ban rögzíttek. A rögzített paraméterek a következők: IMEI, GCM azonosító, SDK verzió, továbbá, hogy az eszköz képes e celluláris, WiFi, Bluetooth és GPS hálózatra kapcsolódni. A következő ábrán ennek a folyamatnak a szekvencia-diagramját láthatjuk.



11. ábra - Eszköz regisztrálásának szekvencia-diagramja

UnregisterDeviceServlet

Ez a szervlet azt a célt szolgálja, hogy ha egy eszköz törölni szeretné magát, akkor ezt megtehesse. Így már nem küldhető további mérés az adott eszközre. Paraméterként a törölni kívánt eszköz IMEI számát adhatjuk meg egy POST kérésben.

UploadModuleServlet

Ennek a szervletnek a segítségével a feltöltött modulok feldolgozása történik. Paraméterként a feltöltött fájlok blob azonosítóját kapjuk meg. A BlobstoreInputStream segítségével beolvassuk a desc fájlt, amely tartalmazza a modulhoz tartozó meta-adatokat JSON formátumban. A sikeres beolvasás után a DataStore-ba a következő értékeket mentjük le: a bejelentkezett felhasználó (aki a feltöltést végezte) e-mail címe és beceneve, a desc fájl és a jar fájl neve, a modul és csomag neve, a desc és a jar fájl blob azonosítója és a feltöltés dátuma.

ServeModuleServlet

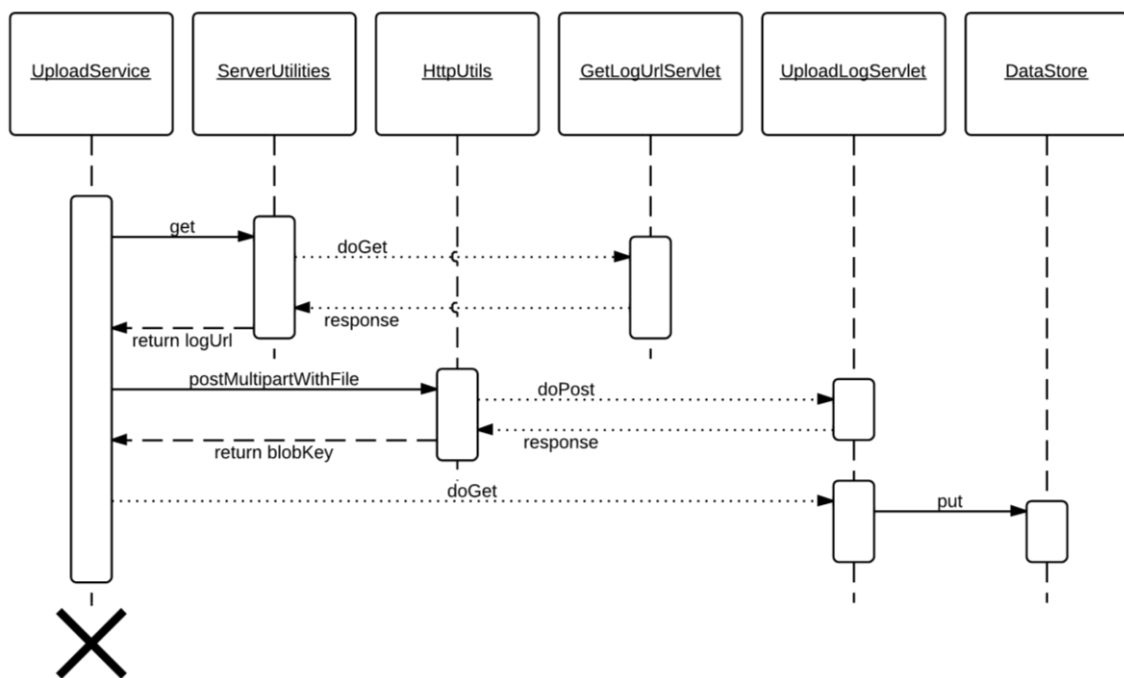
A modulok telefonra történő letöltésére szolgál ez a szervlet. Paraméterként a letölteni kívánt blob-azonosítót kapjuk meg. A fájl kiszolgálását a BlobStoreService serve metódusa biztosítja.

UploadLogServlet

Ebben a szervletben egy GET és egy POST metódus szerepel. A POST metódus hívása a log-fájl feltöltése után hívódik meg és válaszként a feltöltött fájl blob-azonosítóját adja vissza. A GET metódus paraméterül kapja a feltöltést végző készülék IMEI számát és a feltöltött fájl blob-azonosítóját, majd ezeket az értékeket lementi az éppen aktuális dátummal.

GetLogUrlServlet

A log feltöltéséhez szükség van egy generált URL-re ahhoz, hogy a fájlt fel lehessen tölteni a BlobStore-ba. Ezt az elérési utat a BlobstoreService createUploadUrl függvénye segítségével kérhetjük el, amelyet a kérés válaszként küldünk vissza a mobil eszköznek. Az alábbi ábrán látható szekvencia-diagram mutatja be a logok feltöltésének folyamatát. Az UploadService először lekéri a feltöltéshez szükséges URL címet a webszervertől, majd erre a címre küldi a megadott zip-fájlt. Sikeres feltöltést követően a kérés visszatér a blob-azonosítóval, amely egy újabb GET hívással mentésre kerül a DataStore-ba.



12. ábra - Log feltöltésének szekvencia-diagramja

ServeLogServlet

A log-ok elérhetővé tételéhez szükség van egy olyan szervletre, amely biztosítja a log-ok letöltését. Működése nagyon hasonló a ServeModuleServlet-hez.

Paraméterként a letölteni kívánt blob-azonosítót kapjuk meg és a BlobStoreService serve metódusával pedig kiszolgáljuk a kérést.

GetRegisteredPluginsServlet

A mobil-alkalmazásoknak lehetőségük van lekérni az éppen aktuális, elérhető plugin-ek gyűjteményét. Ez a szervlet egy JSONArray-ben küldi el a szerveren regisztrált plugin-ek listáját.

RegisterNewPluginServlet

Ez a szervlet a POST hívás paramétereiben kapott plugin-információkat menti el a DataStore-ba.

ListRegisteredDevices

Ez a JSP oldal az eddig regisztrált eszközök információt jeleníti meg egy táblázatban. Az alábbi ábrán egy éppen aktuális állapotát láthatjuk a regisztrált eszközöknek.

Imei	Sdk version	Cellular	Bluetooth	Wifi	Gps
004999010640000	15	1	1	1	1
353918059712006	17	1	1	1	1
356216043964040	15	1	1	1	1
357441052611329	17	1	1	1	1

13. ábra - Regisztrált eszközök

ListUploadedLogs

Ennek a JSP oldalnak a segítségével az eddig feltöltött logokat tekinthetjük meg és tölthetjük le. A következő ábrán ennek a képét láthatjuk.

IMEI	Date	->
353918059712006	Tue Dec 03 14:50:31 UTC 2013	Download
353918059712006	Tue Dec 03 12:17:39 UTC 2013	Download
353918059712006	Tue Dec 03 10:48:22 UTC 2013	Download
353918059712006	Tue Dec 03 10:18:12 UTC 2013	Download

14. ábra - Feltöltött mérési eredmények

ListUploadedModules

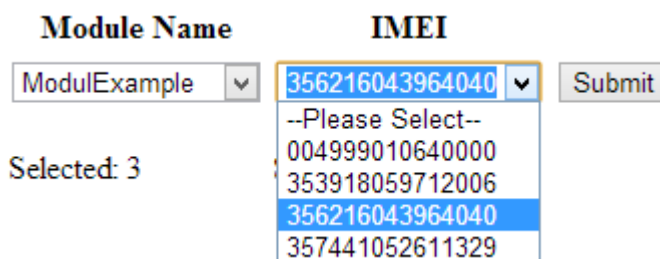
Ezen az oldalon az eddig feltöltött modulok információit tekinthetjük meg, ahogy az alábbi ábrán is látható.

Modul name	Jar file	Class name
ModulExample2	Sample.Module2.jar	hu.edudroid.ict.sample_project2.ModulExample2
ModulExample	Sample.Module.jar	hu.edudroid.ict.sample_project.ModulExample

15. ábra - Feltöltött modulok

ModuleSelector

A modulok készülékre juttatását ez a JSP oldal biztosítja. A felhasználó két legördülő menüből választhatja ki, hogy melyik modul melyik készülékre települjön. A kiválasztott modul és eszköz paraméterként adódik át a GCMBroadcast servlet-nek, amely értesíti az eszközt egy új, letöltendő modulról. Ezt az oldalt az alábbi ábrán láthatjuk.



The screenshot shows a web form with two dropdown menus. The first dropdown is labeled 'Module Name' and has 'ModulExample' selected. The second dropdown is labeled 'IMEI' and has '356216043964040' selected. Below the first dropdown, it says 'Selected: 3'. To the right of the second dropdown is a 'Submit' button. The IMEI dropdown menu is open, showing a list of options: '--Please Select--', '004999010640000', '353918059712006', '356216043964040' (highlighted), and '357441052611329'.

16. ábra - Futtatni kívánt modul és eszköz kiválasztása

UploadModule

Ennek a JSP oldalnak a segítségével tölthet fel új modulokat a kutató. Az oldalon két fájl-beviteli mező található. Az egyikben a modult leíró desc fájlt, a másikban pedig a megvalósítást tartalmazó jar fájlt kell megadni. A sikeres feltöltés után az UploadModule szervlet-re történik átirányítás, ahol beolvassa a desc fájlt, majd menti a korábban megadott adatokat az adatbázisba. A következő ábrán a modul-feltöltő form-ot láthatjuk.

JAR file: SampleModule.jar

DESC file: SampleModule.desc

17. ábra - Modul feltöltése

ListRegisteredPlugins

Ez a JSP oldal kilistázza a szerveren regisztrált plugin-ek listáját. Megjeleníti a plugin nevét, osztálynevét, verziószámát, leírását, és a feltöltés időpontját az alább látható módon.

Name	Class	Ver.	Descr.
WiFiPlugin	hu.edudroid.ictpluginwifi.WiFiPlugin	2	An upgrade to the wifi plugin
WiFiPlugin	hu.edudroid.ictpluginwifi.WiFiPlugin	1	WiFi plugin for wireless interface

18. ábra - Regisztrált plugin-ek

RegisterNewPlugin

Ennek a JSP oldalnak a segítségével új plugin-t regisztrálhatunk a szerveren. Itt meg kell adni a plugin nevét, osztálynevét, verziószámát, és leírását. A form feldolgozását a RegisterNewPluginServlet végzi, amely elmenti a plugin meta-adatait a DataStore-ba. A weboldalon található űrlap az alábbi ábrán látható.

Plugin name:

Plugin class-name:

Plugin version:

Plugin description:

19. ábra - Új plugin regisztrálása

6 Teszt-eredmények

A tesztelés során az elkészült rendszer komponenseinek működését vizsgáltam meg. A fejlesztés folyamán minden egyes felfedezett hiba javítva lett, így a fejlesztés végére az egyes komponensek az elvárásoknak megfelelően működtek. Külön funkcionális tesztrendszert nem hoztam létre, mivel ehhez nem rendelkezttem szükséges erőforrásokkal.

Az rendszer tesztelése manuálisan történt. Végighaladtam az összes komponens főbb feladatain és megvizsgáltam, hogy a működésük megfelelnek-e a követelményeknek. Ehhez természetesen szükség volt az általam korábban létrehozott pluginre és modulra is. Az Androidon futó funkciók helyességéről az Android által szolgáltatott LogCat segítségével, a web-szerveren futó funkciók helyességéről pedig a DataStore-ban és a BlobStore-ban létrehozott bejegyzések alapján győződtem meg.

6.1 Keretrendszer funkcióinak tesztelése

Eszköz regisztrálása a szerverre

Az alkalmazás indulásakor a mobil-eszköz frissíti a meta-adatokat a szerver felé. Egy POST hívásban küldi el a mobil IMEI számát, GCM azonosítóját, SDK verzióját, illetve hogy rendelkezik-e celluláris, WiFi, Bluetooth és GPS interfészekkel.

A tesztelés során a funkció az elvárásoknak megfelelt. Az eszköz regisztrálta magát a szerveren az alkalmazás indulásakor.

Kommunikációs interfészek felállítása

Ennek a funkciónak a feladata, hogy regisztrálja a megfelelő BroadcastReceiver osztályokat, hogy a komponensek képesek legyenek kommunikálni egymással.

Az alkalmazás indulásakor megfelelően regisztrálásra kerültek az osztályok és a broadcast üzenetek mindenhova megérkeztek. Így a teszt-eredmények megfeleltek az elvárásoknak.

Új modul letöltése

Az alkalmazás egy GCM üzenetben kapja meg a letöltendő modul elérési útját, amit aztán le is tölt az eszköz belső tárhelyére.

A GCM üzenetben megfelelő elérési út érkezett a mobil-eszköze és ezt követően az alkalmazás sikeresen le is töltötte a belső tárhelyre a modult.

Plugin-ek betöltése

Az alkalmazás indulásakor a keretrendszer egy broadcast üzenetet küld ki, amelyre minden telepített plugin-alkalmazás válaszol. Így a keretrendszer számára elérhetővé válnak a plugin-ek.

A broadcast üzenet megfelelően eljutott a plugin-alkalmazásokba. Válaszként a plugin alkalmazások elküldték az adott plugint jellemző meta-adatokat, amelyek az elvárásoknak megfelelően meg is érkeztek a keretrendszerbe, és regisztrálásra kerültek az `AndroidPluginCollection` osztályban.

Elérhető plugin-ek lekérése a szerverről

A keretrendszer bizonyos időközönként lekéri a webszerverről az elérhető plugin-ek listáját. A GET kérésre a válasz egy JSON tömbben érkezik, amely tartalmazza az adatbázisban tárolt plugin-ek nevét, osztálynevét, verzióját és leírását.

A kérés megérkezett a webszerverre, az adatbázisból lekért adatok pedig visszaküldésre kerültek egy JSON tömbben, így a működés megfelelt az elvárásoknak.

6.2 Plugin funkcióinak tesztelése

A plugin-specifikus mérések többnyire az Android rendszer által szolgáltatott információkat kéri le. Ezek vonatkozhatnak a mobil aktuális állapotára, vagy valamelyik interfész megfigyelésére. Ugyanakkor vannak olyan kérések, amelyek a Linux réteghez fordulnak mélyebb szintű információkért és éppen ezért futásidejük előre nem meghatározható.

A plugin hívásakor megadott metódus az elvárásoknak megfelelően lekérte az Android által szolgáltatott információkat. A Linux rendszer felé intézett kérések is visszatértek a kért és elvárt értékekkel, így a metódus-hívások működése megfelelt az elvárásoknak.

6.3 Modul funkcióinak tesztelése

Metódus-hívás

A modul a keretrendszertől kéri le az elérhető plugin-ek listáját, amelyen plugin-metódusokat képes hívni. A metódushívásban a metódus neve mellett a paramétereknek is szerepelnie kell. A keretrendszer egy broadcast üzenetben küldi el a plugin felé a metódus nevét és a paramétereket. A mérés lefutása után a plugin visszaküldi a keretrendszernek az eredményeket, amelyeket végül a modul is megkap.

A modul elérte a plugin-ek listáját, így képes volt metódus-hívást intézni a plugin felé. A broadcast üzenetek megfelelően el is jutottak a komponensekhez és a mérési eredmény megérkezett a modulhoz. Így a funkció az elvárásoknak megfelelt.

Modul ütemezése

A modulban lehetőség van a mérések többszöri futtatására. Lehet bizonyos időközönként, vagy egy adott dátumtól kezdődően meghatározott számú mérést futtatni.

A modulban beállított időzítési értékek alapján a modul az elvárásoknak megfelelően futott le.

Logok fájlba írása

A beérkező mérési eredmények egyből loggolva vannak. Egy modul futtatásakor egy ilyen log fájl keletkezik, amibe az eredmények kerülnek. A modul új futtatása során már új logba íródnak az eredmények.

A modul futása során a mérési eredmények loggolásra kerültek, így a funkció az elvárásoknak megfelelt.

6.4 Web-szerver funkcióinak tesztelése

Modulok feltöltése

A futtatandó mérések kódját egy jar kiterjesztésű fájl tartalmazza, amelyet a weboldalon keresztül lehet feltölteni. A feltöltést követően a mobil-eszköz képes azt elérni és letölteni a belső tárhelyére.

A weboldalon megadott fájlokat a szerver feltöltötte a BlobStore-ba és elérhetővé tette a mobil-alkalmazás számára azzal, hogy tárolta hozzá a blob-kulcsot a DataStore-ban.

Logok feltöltése

A lezárt logokat bizonyos időközönként, zip formátumban tölti fel a szerverre az alkalmazás egy POST hívás segítségével.

A funkció megfelelően működött, a log fájlok feltöltésre kerültek a szerverre. Ezek után a weboldalon keresztül letölthetővé, illetve kitömörítés után olvashatóvá is váltak.

GCM küldése az eszköznek

Új modul futtatása során a web-szerver egy GCM üzenetet küld az eszköznek, amelyben a futtatandó modul elérési útja található. Az alkalmazás a GCM üzenetet megkapva megpróbálja letölteni a modult elérhető internet esetén.

A futtatandó modul kiválasztása után a GCM üzenet megérkezett a kiválasztott eszközre és a modul sikeresen letöltésre került a belső tárhelyre, így az elvárásoknak megfelelt a funkció működése.

Új plugin regisztrálása

A weboldalon lehetőség van új plugin információinak a regisztrálására. Itt meg lehet adni a plugin nevét, osztályát, verziószámát, leírását. Majd ezek az információk a DataStore-ban kerülnek rögzítésre.

A megfelelő mezők kitöltése után a plugin információk rögzítésre kerültek a DataStore-ban, így az elvárásoknak megfelelően működött.

7 Összefoglalás

A projekten már korábban is dolgoztak hallgatók, ám a komponensek és a köztük lévő kommunikációs csatornák tervezése elmaradt. Az általam korábban Önálló labor tárgy keretein belül végzett hálózati interfészt vizsgáló plugin-alkalmazás forráskódjából történt átemelés.

Munkám során megterveztem, majd megvalósítottam a DroidLab mérő-alkalmazás prototípusát. Ez magába foglalja az egyes komponensek funkcióinak és kommunikációjának implementálását, illetve tesztelési célokra létrehozott példa-alkalmazások, komponensek megvalósítását is.

A prototípus megvalósításának eredménye egy olyan rendszer, amely képes bemutatni a DroidLab mérőrendszer alapvető működését, így demonstrációs célra megfelelő. A megvalósított plugin és modul segítségével méréseket lehet végezni a mobil-eszközök hálózati interfészén is. A rendszer webes felületén keresztül lehetőség van a mérési eredmények letöltésére, illetve újabb mérések futtatására a regisztrált eszközökön.

Ahhoz hogy az alkalmazás tágabb körökben is elérhető legyen, fontos a biztonsági kritériumok átgondolása és implementációja. A méréseket vezérlő és kvótázó komponens még teljesen hiányzik a rendszerből, így annak létrehozására is szükség van a továbbiakban.

Az eddig elkészült alkalmazás a tanszéken már a többi kutató számára is hozzáférhető a Google Play Store-on keresztül, illetve felsőbb éves hallgatók plugin-alkalmazásokkal kiegészíthetik és méréseket végezhetnek a hálózatban résztvevő készülékeken.

8 Köszönetnyilvánítás

Köszönöm tanszéki konzulensemnek, Lajtha Baláznak a sok segítséget és tanácsot, amit a fejlesztés, majd a dolgozat készítése közben adott, illetve diáktársaimnak, akik részt vettek a projektben. Továbbá szeretném megköszönni barátnőmnek, hogy annyi mindenben támogatott és türelemmel fordult hozzám a dolgozat készítése közben. Végül pedig szüleimnek szeretném megköszönni, hogy átolvasták a dolgozatot és felhívták a figyelmem egy két apróbb hibára.

Rövidítésjegyzék

CMS – Content Management System: Különböző tartalmak rendezett tárolására, megjelenítésére és visszakeresésére használható szoftver rendszer. Elsősorban portálrendszerekkel kapcsolatosan fordul elő. A portál megjelenítését, adminisztrációját biztosítja.

GAE – Google App Engine: Egy platform szolgáltatás webalkalmazások fejlesztésére és működtetésére.

GCM – Google Cloud Messaging: Egy olyan szolgáltatás, amely segíti a fejlesztőket, hogy üzenetet küldhessenek a szerver felől az Android operációs rendszert futtató mobil-eszközökre.

GPS – Global Positioning System: A GPS rendszer segítségével a földfelszín bármely pontján meghatározhatjuk a pontos helyzetünket, szélességi és hosszúsági koordinátáinkat.

HTTP – Hypertext Transfer Protocol: Gazdagépek, kiszolgálógépek és az egyéni felhasználó gépe közötti láncszöveges (hipertext) kommunikáció protokollja.

IMEI – International Mobile Equipment Identity Number: A GSM rendszerű mobil-készülékek elektronikus módon rögzített egyedi azonosítója. A készülék hátulján és a jótállási jegyen nyomtatott módon is megtalálható, 15 számjegyből álló számsor.

JAR – Java Archive: A JAR fájl olyan ZIP fájlformátumú fájl, amely java osztályokat és hozzájuk tartozó metaadatokat tartalmaz.

JSON – JavaScript Object Notation: Egy kis-méretű, szöveg alapú szabvány, ember által olvasható adatcserére.

JSP – Java Server Pages: Egy technológia, melynek segítségével a szoftverfejlesztő dinamikusan tud generálni HTML, XML vagy egyéb dokumentumokat HTTP kérésekre reagálva.

PHP – PHP Hypertext Preprocessor: A PHP általános szerver oldali szkript nyelv, melyet dinamikus weblapok megalkotására hoztak létre.

SDK – Software Development Kit: Szoftverfejlesztői csomag, amely API-t szolgáltat az adott platform fejlesztéséhez.

SQL – Structured Query Language: Strukturált lekérdezőnyelv, amely relációs adatbázis-kezelők lekérdezési nyelve.

URL – Uniform Resource Locator: Webcím, amely az interneten megtalálható bizonyos erőforrások (például szövegek, képek) szabványosított címe.

WiFi – Wireless Fidelity: Az IEEE által kifejlesztett vezeték nélküli mikrohullámú kommunikációt (WLAN) megvalósító, széleskörűen elterjedt szabvány (IEEE 802.11) népszerű neve.

XML - Extensible Markup Language: A W3C által ajánlott általános célú leíró nyelv, speciális célú leíró nyelvek létrehozására. Különböző adattípusok leírására képes és az elsődleges célja strukturált szöveg és információ megosztása az Interneten keresztül.

Irodalomjegyzék

- [1] Mobile Crowdsensing: Current State and Future Challenges Raghu K. Ganti, Fan Ye, and Hui Lei IBM T. J. Watson Research Center, Hawthorne, NY
- [2] Crowd-sourcing: <http://www.wired.com/business/2008/12/crowdsourcing-n/>
- [3] uTest: <http://www.utest.com/>
- [4] Wikipedia: <http://en.wikipedia.org/wiki/Wikipedia:About>
- [5] CreekWatch: <http://creekwatch.researchlabs.ibm.com/>
- [6] Időkép: <http://www.idokep.hu/>
- [7] CarTel: <http://cartel.csail.mit.edu/doku.php>
- [8] NeriCell: <http://research.microsoft.com/en-us/projects/nericell/>
- [9] ParkNet: Drive-by Sensing of Road-Side Parking Statistics; WINLAB, Rutgers; http://www.winlab.rutgers.edu/~suhas/SuhasMathur_Mobisys2010.pdf
- [10] OpenSignal Mobile App: <http://opensignal.com/>
- [11] DietSense: <http://www.dietsens.appspot.com/>
- [12] Jan Albrecht: The Java-Sandbox
<http://blog.datenwerke.net/p/the-java-sandbox.html>
- [13] Android: <http://developer.android.com/develop/index.html>
- [14] Android SDK: <https://developer.android.com/sdk/index.html>
- [15] Android Developer Tools: <http://developer.android.com/tools/index.html>
- [16] Google App Engine: <https://developers.google.com/appengine/>
- [17] DataStore: <https://developers.google.com/appengine/docs/java/datastore/>
- [18] BlobStore: <https://developers.google.com/appengine/docs/java/blobstore/>
- [19] GAE users: <https://developers.google.com/appengine/docs/java/users/>
- [20] Java szervlet: <http://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html>
- [21] Java Server Pages: <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
- [22] Google Cloud Messaging: <http://developer.android.com/google/gcm/index.html>