

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Kiss Gábor

Android mérőrendszer erőforrás-kezelése

Diplomaterv

Konzulens

BUDAPEST, 2013

Tartalomjegyzék

[Összefoglaló 5](#_Toc375234600)

[Abstract 6](#_Toc375234601)

[1 Bevezetés 7](#_Toc375234602)

[2 Az erőforrás-kezelő rendszerkomponens 9](#_Toc375234603)

[2.1 A DroidLab mérőrendszer 9](#_Toc375234604)

[2.1.1 A rendszert építő eszközök 9](#_Toc375234605)

[2.1.2 A rendszer felépítése 11](#_Toc375234606)

[2.2 A kvóta rendszer 14](#_Toc375234607)

[2.2.1 Erőforrás használat szabályozása egyéb rendszerekben 15](#_Toc375234608)

[2.2.2 Szolgáltatás-csoportok és kvóták 20](#_Toc375234609)

[3 Gamification 25](#_Toc375234610)

[3.1 A gamification fogalma 25](#_Toc375234611)

[3.2 Gamification kategóriák 26](#_Toc375234612)

[3.2.1 Internal gamification 26](#_Toc375234613)

[3.2.2 External gamification 27](#_Toc375234614)

[3.2.3 Behavior change gamification 27](#_Toc375234615)

[3.3 A gamification eszköztár 29](#_Toc375234616)

[3.3.1 Az elemek hierarchikus piramisa 29](#_Toc375234617)

[3.3.2 A „PBL hármas” 31](#_Toc375234618)

[3.4 A játékosok típusai 34](#_Toc375234619)

[3.4.1 A Bartle modell 34](#_Toc375234620)

[4 Tervezés, rendszerkomponensek implementációja 37](#_Toc375234621)

[4.1 Erőforrást kezelő komponens 37](#_Toc375234622)

[4.1.1 Processzor használat szabályozás 38](#_Toc375234623)

[4.1.2 Plugin szabályozás 53](#_Toc375234624)

[4.1.3 Profilozás 60](#_Toc375234625)

[4.2 Gamification elemek implementálása 62](#_Toc375234626)

[5 Tesztelés 63](#_Toc375234627)

[6 Összegzés 70](#_Toc375234628)

[Irodalomjegyzék 71](#_Toc375234629)

Hallgatói nyilatkozat

Alulírott **Kiss Gábor**, szigorló hallgató kijelentem, hogy ezt a 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. 21.

...…………………………………………….

Kiss Gábor

Összefoglaló

Mindennapos életünk részévé váltak a különböző mobil készülékek. Ezek közül is kiemelt szerep jutott az okostelefonoknak, hiszen számos egyéb eszköz funkcionalitását átvették az évek során. Hála a technológia folytonos fejlődésének a gyártók a versenyképes készülék előállítása érdekében egyre változatosabb és nagyobb teljesítményű alkatrészeket integrálnak az eszközeikbe. Ennek ellenére a legtöbb felhasználó csak nagyon ritkán szokta készülékének maximális teljesítményét kihasználni a használati idő nagy részében.

Az ilyen szabad kapacitások kiaknázását szeretné megoldani az általunk kezdeményezett, DroidLabnak elkeresztelt merőrendszer. A projektben a fő feladatom volt a megfelelő erőforrás-kezelés kialakítása, melynek célja, hogy végje az adott eszközt és tulajdonosát a mérések által okozott esetleges kellemetlenségektől. Ez a fő témája diplomatervemnek is.

Az erőforrás-szabályozás megvalósítása mellett rám hárult a feladat, hogy a gamification („játékosítás”) módszertanának megismerése után ajánlást tegyek a megfelelő felhasználó-bázis illetve mérési körülmények kialakítására.

Abstract

The mobile devices have a very important role in our lives. Cell phones are the most used multifunctional tool today. The device manufacturers are integrating the newest and most powerful device parts into their products, because they want to stay in the market. But the most users can’t utilize these new resources effectively.

Our goal is to realize a testbed-like system, called Droidlab, which exploits these unused resources. As member of the team, my job was to create a resource-management subsystem. This component’s task is to protect the user and the device from the disadvantages of measurements (e.g.: performance decrease). This is the main subject of my work.

My secondary job was to study the gamification methods and use them to design a business-like model, which assists the project in evolving a proper user base and measurement environment.

# Bevezetés

Az Internet és az egyéb vezetékes illetve vezeték nélküli hálózatok teljes mértékben megváltoztatták életünket. Számos olyan szolgáltatás vált mindenki számára elérhetővé, melyről akár egy évszázada csak álmodni mertünk volna (például: kilométereket áthidaló videó hívások vagy vásárlás). A megnövekedett felhasználói igények szükségessé teszik a meglévő megoldások optimalizálását vagy újabb technológiák fejlesztését. Természetesen egy-egy újítás teljes körű bevezetése előtt számos tesztelési és finomítási lépésen kell végighaladnia az adott technológiának. E folyamat általában valamilyen szimulációval szokott indulni, mely során becsült értéket nyerhetünk az adott technológia egyes tulajdonságaival kapcsolatban (például: teljesítmény vagy skálázhatóság). A szimuláció során általában valamilyen leegyszerűsített modellt használunk, ami a valós környezet tényleges dinamizmusát nem tükrözi, így a szimuláció során kapott eredmények nem megfelelőek az adott technológia eredményességének bizonyítására.

Következő lépésként általában valamilyen emulációt szoktak használni a további vizsgálatok elvégzésére. Ebben az esetben már jóval komplexebb forgatókönyveket tudunk létrehozni, melyek sokkal jobban tükrözik az éles helyzetben elvárható eseményeket, így ez már megfelelőbb megközelítésnek mondható. Ennek ellenére számos olyan környezeti hatás (például aktuális felhasználók viselkedése, területi infrastruktúra) lehet, melyet még így sem tudunk kellőképp modellezni, ezért manapság egyre jobban elterjedtek az úgynevezett tesztrendszerek vagy idegen szóval kifejezve testbed-ek.

Egy ilyen tesztrendszer valamilyen kísérleti eszközök sokaságából áll, melyeken a valósághoz legközelebb eső méréseket tudjuk elvégezni. Az így kapott eredmények már alkalmasak a helytálló következtetések megállapítására. A tesztrendszerek sikerességét igazolja az is, hogy egyre több ilyen kezdeményezés indul el világszerte, például az ORBIT[1] (Open-Access Research Testbed for Next-Generation Wireless Networks) névre keresztelt vezeték nélküli hálózatot biztosító tesztrendszer, vagy a PlanetLab[2], mely többféle hálózati szolgáltatással kapcsolatban is képes tesztkörnyezetet biztosítani. Azt az eddigiek alapján beláthatjuk, hogy különböző tesztrendszerek pozitív hatással vannak a technológiai fejlődésre.

A mobil eszközök egyre fontosabbá válnak mindennapi életünkben. Az eszközgyártók folyamatosan növelik készülékeik teljesítményét, és ezeket az erőforrásokat a felhasználók nem használják ki teljes mértékben. Felmerült az ötlet, hogy e szabad kapacitásokat jó lenne valahogyan kutatási illetve fejlesztési célokra felhasználni. Az előbbi elképzelést továbbgondolva született meg egy olyan mérőrendszer megvalósításának ötlete, mely a már szóba hozott tesztrendszerekhez hasonlóan működik és az említett szabad erőforrásokat használja fel. A projektet DroidLabnak neveztük el.

Feladatom az előbb említett mérőrendszer megvalósításában a készülékek erőforrásainak egységekbe szervezése, ezen egységek (későbbiekben kvóták) kiajánlása a felhasználóknak és végül a felhasználók által beállított mennyiségek betartatása a rendszerrel. E mellett feladatom még úgynevezett gamification módszereket felhasználva olyan megvalósítás megtervezése és részleges vagy teljes implementálása, mely a megfelelő felhasználó bázis kialakítását szolgálja, illetve motiválja az addigi felhasználókat minél több erőforrás megosztására.

A dolgozatom következő fejezetében bemutatom röviden a mérőrendszer egyes komponenseit illetve felépítését, ezt követően részletesen kitérek az eszközökkel kapcsolatos erőforrásokra hozzájuk tervezett kvótákra. A 3. fejezetben ismertetem a gamification fogalmát és a legfontosabb alkotó elemeit.

# Az erőforrás-kezelő rendszerkomponens

Ebben a fejezetben bemutatom magát a mérőrendszert, majd beszámolok pár eddig megvalósított erőforrás-szabályozási módszerről, végezetül a kvóta rendszerrel kapcsolatos fontos információkat ismertetem.

## A DroidLab mérőrendszer

Az általunk megvalósított rendszer célplatformja az Android lett. A többi, piacon levő alternatívával összehasonlítva az Android rendelkezett a feladat számára legkedvezőbb tulajdonságokkal. Ezek közül a legfontosabb három tulajdonságot emelném ki:

* nyílt forráskódú, jól dokumentált, nagy fejlesztői és felhasználói körrel rendelkezik
* jelenleg az egyik legelterjedtebb platform, megjelenése óta népszerűsége folyamatosan növekszik
* legváltozatosabb az eszköztára (telefon, tablet, tv, játék konzol…)

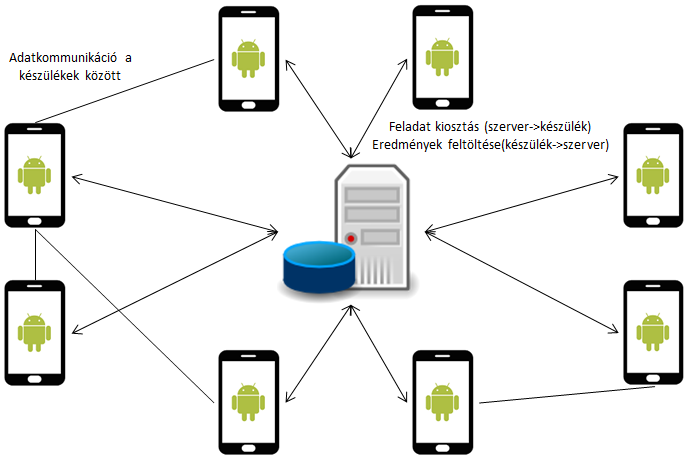
### A rendszert építő eszközök

A DroidLab részben eltér a hagyományos tesztrendszerektől a hálózatot alkotó eszközök szempontjából. A megszokottól eltérően nekünk nem áll szándékunkban egy izolált, általában laboratóriumi környezetbe beágyazott eszközállományt kialakítani, hanem egyszerű eszköztulajdonosok (későbbiekben felhasználó) készülékeit szeretnénk erre a célra felhasználni néhány általunk írt alkalmazás segítségével. Ez a megoldás azért ideálisabb a hagyományos megvalósításnál, mert így nem kell külön az eszközöket megvásárolni a hálózat kialakításához, ami általában egy hasonló projekt költségvetésének nagy részét szokta jelenteni. Megoldásunk mellett szól az az érv is, hogy az így végzett mérések sokkal reálisabbak lesznek, hiszen a mért adatok a mindennapos használat során születnek, nem pedig a már említett izolált környezetben.

Természetesen az előnyök mellett hátrányai is vannak ennek az elképzelésnek, hiszen sokkal kiszámíthatatlanabb lesz az adott hálózat, mivel a teljes irányítás az adott felhasználó kezében van. E mellett személyi jogokkal kapcsolatos gondok is felmerülhetnek, a későbbi kellemetlenségek elkerülése végett a rendszer tervezésénél ezeket megfelelő súllyal figyelembe kell venni. Végezetül kezdeti nehézségként felmerülhet a meglelő felhasználó bázis kiépítése is, erről még később részletesen is szó lesz (lásd: 3.2.2 External gamification).

A mérésekre használt készülékek mellett szükség van egy központi vezérlő egységre is. Ez a rendszerkomponens nem vesz közvetlen részt a mérések tényleges elvégzésében, legfőbb feladatai közé tartozik a készülékek és a hozzájuk tartozó adatok számon tartása, az elvégzendő feladatok kiosztása a rendelkezésre álló szabad eszközökre és az elvégzett mérések eredményeinek fogadása.

Az elhangzottak alapján a rendszer fizikai topológiájának egy vázlatos sémáját az 2.1. ábrán látható módon képzelhetjük el.



2.1. ábra: A rendszer fizikai topológiájának egy vázlatos sémája

Láthatjuk, hogy az összes merőrendszert alkotó készüléknek kapcsolatban kell lennie a központi egységgel, viszont ennek a kapcsolatnak nem kell feltétlen állandó, „online” kapcsolatnak lennie. Elképzelhető olyan mérési szituáció, amely nem követeli meg ezt az helyzetet. Viszont a feladatok kiosztásánál és az adatok feltöltésénél mindenképpen kell valamilyen rádiós hálózati összeköttetés a megfelelő működés érdekében.

Az ábráról jól látszik, hogy lehetséges közvetlen kapcsolat kialakítani kettő vagy több mérőműszer között is. Erre általában csak valamilyen speciális mérés során lehet szükség, például Bluetooth erőforrás segítségével szeretnénk mérni átviteli sebességet. A Bluetooth kommunikáció kialakításához legalább két hardware-sen támogatott eszköz kell, amiből következik, hogy ezt egy merőműszer segítségével nem lehet megoldani. Ilyenkor van szükség az említett közvetlen kapcsolatra. Az ilyen mérések lebonyolítása sokkal nagyobb koordinációt igényel, mint azoké, melyeket egymástól független mérő eszközökön el lehet végezni.

### A rendszer felépítése

A fizikai topológiához hasonlóan a szoftveres megvalósítást is két jól elkülöníthető részre oszthatjuk, egy szerverre és egy kliens alkalmazásra. A szerver a már említett központi vezérlő egységen fut és fő feladata a már bemutatott funkciók szoftveres megvalósítása, ezek közül is a legnagyobb hangsúlyt a kliens információk megfelelő karbantartására helyezve. Emellett még a mérőrendszert mérési célokra igénybevevő személyek (későbbiekben kutatók) is a szerver egy erre szolgáló felületén tudják a mérési feladataikat feltölteni. A tervek szerint a felhasználóknak is kialakítunk egy felületet, amin a saját adataikat és profiljukhoz tartozó információkat tudják megtekinteni, vagy egyes adatokat pótolni illetve módosítani. A szerver funkcióit a projekt kezdetekor egy PHP website valósította meg, de ezt a megoldást Google Appengine[[1]](#footnote-1) környezetre cseréltük le.

Magán a mérő eszközökön futtatott szoftvert tekintjük a már említett kliensnek. A kliens szinte minden készüléken futtatható, két elvárást támasztunk az eszközökkel szemben:

* rendelkezzenek hálózati hozzáféréssel
* érjék el a Google szolgáltatásait

Az előbbi magától értetődik, ha visszagondolunk a rendszert alkotó eszközök ismertetésénél leírtakra. Ez a kritérium nagyon kevés esetben jelenthet problémát, az olcsóbb készülékekben is van Wi-Fi modul, mely segítségével vezetéknélküli hálózatra képes csatlakozni. Utóbbi elvárásra azért van szükség, mivel a klienshez tartozó alkalmazásokat a Google Play[[2]](#footnote-2) digitális tartalomszolgáltatón keresztül szeretnénk terjeszteni a könnyű elérés és frissítés érdekében, az online bolt által nyújtott ellenőrzési és hitelesítési lehetőségeket kihasználva.

Mivel az egyik fő szempont a mérőrendszer tervezésekor a felhasználók szabad döntési jogának biztosítása volt, ezért a kliens alkalmazást teljesen modulárisra terveztük, így három egymástól jól elkülöníthető részre bontottuk.

#### Pluginok

Az Android platform eléggé sajátos erőforrás kezelési mechanizmussal rendelkezik. Ennek lényege, hogy az adott erőforrást használni kívánó alkalmazásnak rendelkeznie kell a megfelelő hozzáférési joggal (idegen szóval permission-nel), hogy az adott funkció hibamentesen működjön. Ez még nem jelentene problémát, a gondot a hozzáférési jogok kezelése okozza. A felhasználó az adott alkalmazás megfelelő működéséhez szükséges jogokat csak a telepítési folyamat kezdeti szakaszában engedélyezheti, és mindenképpen el kell fogadni az összes igényelt kérelmet, hogy a telepítés sikeres legyen. Az engedélyek megtagadása a folyamat automatikus megszakításával egyenlő. Emellett a megadott jogokat a későbbiekben nincs lehetősége a felhasználónak visszavonni, csak az alkalmazás törlése által.

Beláthatjuk, hogy ez a megoldás nem felhasználóbarát. Mivel az általunk tervezett rendszer egy adott készülék majdnem minden lehetséges erőforrását szeretné kihasználni, így ha csak egy komplex alkalmazásba sűrítenénk az összes funkciót az előbb ismertetett mechanizmus miatt minden szükséges jogot egy időben kellene engedélyezni a DroidLab futtatásához. Ez teljes mértékben megfosztaná a felhasználót annak eldöntésétől, hogy mely erőforrásokat hajlandó a rendelkezésünkre bocsájtani. Ennek kiküszöbölése érdekében olyan felépítés mellett döntöttünk, amely az egyes szolgáltatás-csoportokat pluginokként valósítja meg. Ez a kialakítás biztosítja az erőforrásokhoz definiált funkciók könnyű bővítési lehetőségét illetve a jövőbeli új erőforrások rendszerbe való integrálását.

Minden plugin önálló alkalmazásként képzelhető el, így csak az adott csoport működéséhez szükséges jogokat kell a felhasználónak biztosítania. Ezen kívül a felhasználóknak lehetőséget kínálunk az adott plugin által biztosított funkciók egyéni korlátozására is az erre a célra készített beállítási felületen.

Az egyes csoportok a későbbiekben részletesen bemutatásra kerülnek (lásd: 2.2.2-es fejezet).

#### Modulok

A kutatók által fejlesztett méréseket hívjuk moduloknak. A modulok a tesztelési fázisban Java nyelven készülnek. A moduloknak olyan futtatható kódrészleteknek kell lennie, melyek felett a készülékeken futó kliensnek teljes ellenőrzése van. Ezért a modulok nem kommunikálhatnak közvetlenül az Android rendszerrel, nem indíthatnak saját szálat, nem iratkozhatnak fel rendszer-eseményekre.

A biztonsági követelményeknek a Java kód nem felel meg minden szempontból. Különböző trükkökkel a Java által biztosított biztonsági konténer kijátszható és tetszőleges kódrészlet futtatható lenne, többek közt olyan is, mely magát a klienst vezérelné, így teljes ellenőrzést gyakorolna a felhasználó eszköze fölött. Ezért a modulok számára egy második, saját magunk által tervezett biztonsági konténert akarunk kialakítani a Java által nyújtott lehetőségen belül, hogy a fentebb említett követelmények mindenképpen biztosíthatóak legyenek. Ezért a modulok programozási nyelvét az éles rendszerben magasabb szintű script nyelvre, valószínűleg a LUA-ra fogjuk cserélni, de lehet, hogy egy saját fejlesztésű nyelv fogja felváltani.

A modulok a pluginok funkcióit felhasználva végeznek méréseket. Ezek a mérések vagy periodikusak, vagy eseményvezéreltek, a mérendő tulajdonság vagy jelenség jellegéhez igazodva. A kétféle viselkedés a modul kódjában elkülönül. Elnevezési konvenciókkal biztosítjuk a kétféle működés megvalósítását.

A modulok számára a kliens állandó háttértárat biztosít a részeredmények tárolására, és egy naplózási lehetőséget az eredmények rögzítésére.

A kutatók megadják, hogy hány készüléken, milyen mennyiségű erőforrásra van szükségük, a vezérlő ezeket az igényeket párosítja a felajánlásokkal, majd küldi a modulokat a megfelelő eszközökre.

A modulok funkcionalitásukból adódóan nem rendelkeznek saját felhasználói felülettel, ennek ellenére lehetőséget biztosítunk az adott eszközön futtatott modulok listás megjelenítésére és egyes listaelemeket kiválasztva az adott felhasználó információt szerezhet a kiválasztott modulról, illetve szükség esetén akár meg is szakíthatja annak futását.

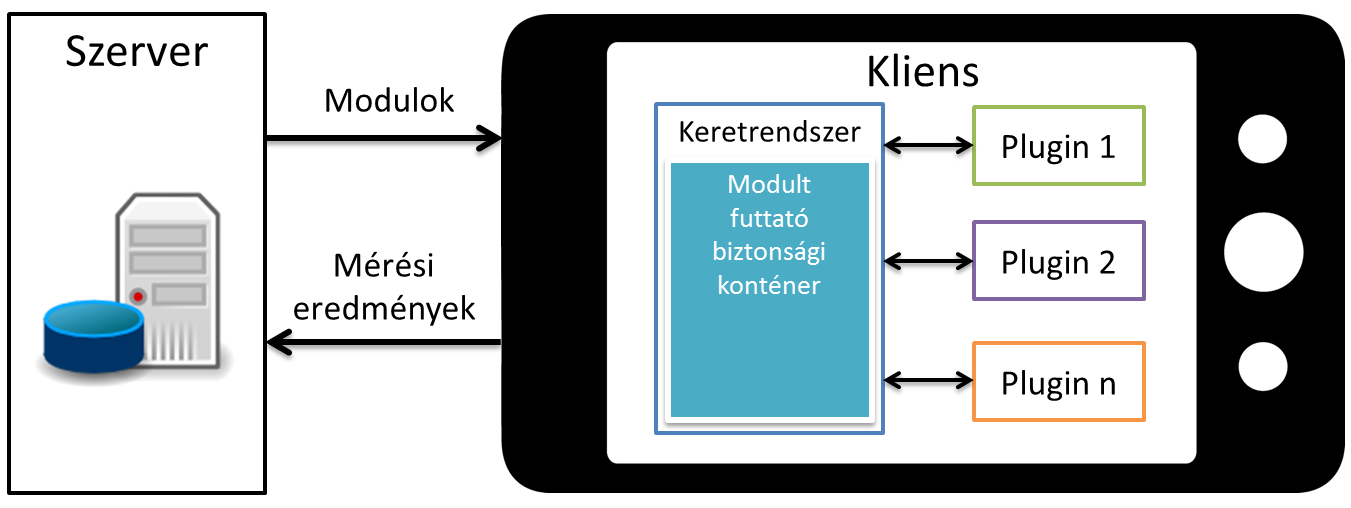
#### A keretrendszer

A DroidLab alapfunkcionalitását megvalósító alkalmazást nevezzük keretrendszernek. A keretrendszer feladata a telepített pluginok megkeresése és igény szerinti frissítése vagy letöltése. A keretrendszer tölti le, indítja el és állítja le a modulokat, tölti föl a mérési eredményeket.

Minden plugin definiálhat generikus eseményeket és generikus adatokat, amikre a keretrendszer feliratkozik, és az így kapott információkat összegyűjtve továbbítja a szerver felé.

A keretrendszer felülete általános híreket és statisztikákat közöl a rendszerről. Ezen a felületen tudnak a hálózat üzemeltetői kommunikálni a felhasználóval, felhívások, hibaüzenetek és egyéb visszajelzések közvetíthetőek a felhasználók felé.

Ezek alapján a mérőrendszer szoftveres felépítésének egy vázlatos sémáját az 2.2. ábrán látható módon képzelhetjük el.



2.2. ábra: A mérőrendszer szoftveres felépítésének egy vázlatos sémája

## A kvóta rendszer

Nyugodtan kijelenthetjük, hogy a mobil készülékek jelen technológia mellett sajnos még rendelkeznek néhány szűkös erőforrással. Ezek közül talán tápellátás az, ami a minden felhasználó számára kiemelt fontosságú. Könnyen belátható, hogy egy olyan kliens, amely az üzemidő drasztikus csökkentését eredményezné, az egész eszköz-hálózat kiépítését kivitelezhetetlenné tenné. Egy felhasználó sem lenne hajlandó ilyen tulajdonsággal rendelkező alkalmazás telepítésére. A felhasználók bizalmának elnyerése érdekében mindenképpen szükség van a mérőrendszerrel kapcsolatos korlátok létrehozására és annak biztosítására, hogy ezeket a rendszer ténylegesen be is tartja. Azt már láthattuk, az Android jogosultság kezelési rendszere több kívánnivalót hagy maga után, de sajnos a platform nem biztosít lehetőséget a telepített alkalmazások erőforrás-használatának korlátozására sem. E hiányosság pótlása az én fő feladatom.

Az eddig már kiderült, hogy a mérőrendszer használhatósága a felhasználók számától és az általuk felajánlott erőforrásoktól függ. Bár ez egy nem túl elterjedt modell, számos más területen érvényesülő, felhasználók szabad erőforrásait felhasználó projekt létezik, mely az elképzelés kivitelezhetőségét igazolja. Ezek közül talán a legismertebb a világsikerű SETI@home[3] projekt. A SETI@home egy tudományos kísérlet, amely az Internethez csatlakozó számítógépeket használja a földönkívüli élet kutatásához. A részvételhez mindössze egy ingyenes program futtatására van szükség, ami rádióteleszkóp-adatokat tölt le és elemez ki, mindezt kizárólag egy képernyőkímélő futtatása alatt. Így ténylegesen csak akkor használják az asztali gépek kapacitását, ha a tulajdonos már jó ideje nem használta azt komolyabb feladat végzésére. Ezzel a módszerrel sikerült a kezdetben használt, speciális szuperszámítógépet helyettesíteni résztvevő gépek szabad kapacitásával. Természetesen a célunk nem egy ilyen elosztott szuperszámítógép létrehozása a DroidLab segítségével (inkább egyfajta szenzorhálózatként tekintünk a mérőrendszerre), de a SETI@home sikere mindenképpen bíztató a saját projektünk megvalósíthatósága szempontjából.

Mivel a DroidLab-hoz hasonló elképzelés még nem született a tudományos világban, ezért kénytelen voltam más jellegű rendszerekben megvalósított erőforrás-kezelést megvizsgálni az Android hiányosságát pótló megoldás keresése közben.

### Erőforrás használat szabályozása egyéb rendszerekben

Mivel az Android platform Linux kernellel rendelkezik, ezért kézenfekvő volt ezen operációs rendszerrel kapcsolatos lehetőségeket megvizsgálni. A változatos felhasználási területeknek köszönhetően számos erőforrás szabályozási módszert sikerült találnom. Természetesen ezek a megoldások a mobil eszközök és az asztali gépek közötti különbségek miatt nem adhatnak tökéletes megoldást, de a közös erőforrásokkal kapcsolatban jó kiindulási alapot nyújthatnak.

#### setrlimit[4]

A setrlimit() egy olyan rendszerhívás, mely segítségével folyamatonkénti erőforrás korlátozást tudunk beállítani. Minden ilyen korlátozáshoz egy érték pár tartozik, mely egy névleges (soft limit) és egy tényleges (hard limit) határértékből áll. A soft limit értéket próbálja a kernel fenntartani, ez tulajdonképpen egyfajta intervallumot definiál az erőforrással kapcsolatban. A hard limit ennek az intervallumnak a felső korlátja, amit a rendszer semmiképp nem enged túllépni. Ezeket az értékpárokat egy úgynevezett rlimit struktúrában tárolja a rendszer. A setrlimit segítségével számos erőforrás korlátját beállíthatjuk ezek közül a fontosabbak:

* RLIMIT\_CORE: a folyamat által létrehozható visszaállítási fájl maximális mérete byte-okban
* RLIMIT\_CPU: a folyamat által használható maximum processzor idő másodpercben kifejezve (csak soft limitként értelmezhető)
* RLIMIT\_DATA: a folyamat által létrehozható összes adat maximális mérete byte-okban
* RLIMIT\_FSIZE: a folyamat által létrehozható fájl maximális mérete byte-okban
* RLIMIT\_NOFILE: a folyamat által használt fájl leírók maximális száma
* RLIMIT\_STACK: a folyamathoz tartozó maximális stack mérete byte-okban
* RLIMIT\_MEMLOCK: a folyamat által RAM-ba (Random Access Memory) zárható memória maximális mérete byte-ban
* RLIMIT\_AS: a folyamathoz tartozó virtuális memória maximális mérete byte-ban

Maga a rendszerhívás a Linux 2.6.36-os verziója óta elérhető, előtte egy úgynevezett vlimit() függvény segítségével lehetett hasonló funkciót elérni.

#### cpulimit[5]

A cpulimit egy 2006-ban, Angelo Marletta nevezetű úriember által létrehozott projekt. Lényege, hogy egy egyszerű program segítségével a különböző folyamatok processzor használatát szabályozhatjuk. Az setrlimit-tel ellentétben itt nem a processzor időre adhatunk korlátot, hanem beállíthatunk egy százalékban kifejezett értéket a processzor használatra. Egy konkrét példán ezt bemutatva: ha van egy egymagos gépünk, amelyen egy folyamathoz a cpulimit segítségével beállítottuk, hogy 50%-os maximális kihasználtságot érhet el, akkor ebben az esetben az adott folyamat másodpercenként legfeljebb 500 milliszekundum processzor időt használhat fel. Ahány maggal rendelkezik az adott számítógép, annyiszor 100%-ot oszthatunk szét a folyamatok között (például egy négymagos gép esetén összesen 400%-ot).

Maga az alkalmazás teljes mértékben a rendszer felhasználói módjában fut, így a Linux ütemezőt egyáltalán nem zavarja. A program lényege, hogy a /proc könyvtárban tárolt folyamat statisztikákat és a felhasználó által beállított értékeket figyelve az alkalmazás jelzések segítségével folyamatosan megállítja vagy folytatja az adott folyamatot.

A cpulimit alkalmazást minden 2.2 vagy e feletti Linux rendszeren használhatjuk, viszont rendszer szintű integrálása nincs neki, a felhasználónak kell integrálnia saját rendszerébe, ha használni akarja. Így alapvetően az Android rendszer sem tartalmazza és a dolgozat írásának időpontjában még nem készült el a Google platformjával kompatibilis cpulimit kiegészítés.

#### quota[6]

A quota egy fájlrendszer-használati korlátozás, melyet felhasználónként vagy csoportonként érvényesíthetünk. A korlátokat két szempont szerint is meghatározhatjuk: korlátozhatjuk a lehetséges csomópontok (inode-k) számát, illetve meghatározhatjuk az egyes felhasználókhoz vagy csoportokhoz tartozó lemezblokkok számát is. Hasonlóan a setrlimithez, itt is definiálhatunk soft illetve hard limiteket is. Soft limitet elérve a rendszer figyelmeztetést küld a felhasználónak, a hard limit elérése esetén a kernel nem engedi a további szabad hely használatát. A módszer hátránya, hogy csak az ext2 típusú fájlrendszereken alkalmazható. A quota lehetőségét engedélyezni kell a kernelben, és csatolási pontonként a /etc/fstab-ban.

Az 1.3.8x-as Linux verzió óta a qouta integrálva van a kernelben.

#### grsecurity[7]

A grsecurity tulajdonképpen olyan Linux kernel foltok (patch-ek) gyűjteménye, mely a Linux ismert biztonsági réseit hivatott javítani. GNU General Public License alatt adták ki és bárki számára ingyenesen elérhető.

A grsecurity több nagyobb komponensből épül fel, ezek közül számunkra az az egy fontos, amelyik egy teljes szerepalapú hozzáférési jogosultság szabályozó rendszert (Role-based access control (RBAC)) valósít meg. A RBAC előnye a hagyományos Unix hozzáférés-vezérlési listákhoz képest, hogy az egyes felhasználók és folyamatok jogosultságait a minimálisra tudja csökkenteni, így egy esetleges támadás esetén a támadónak sokkal kisebb esélye van sikerrel járni, mint normál esetben.

A RBAC-nek köszönhetően az erőforrás szabályozást is szerepekhez köthetjük. A grsecurity logikailag ugyanúgy működik, mint a setrlimit rendszerhívás. Ebben az esetben is soft és hard limiteket állíthatunk és a grsecurity is minden Linux erőforrást támogat. Két főbb eltérés van a setrlimit-el szemben:

* az erőforrások megnevezése: mint azt láthattuk alapvetően a Linux erőforrásokat az RLIMIT előtaggal használjuk általában. A grsecurity kapcsán ezt az előtagot RES-re cserélték (például a RLIMIT\_CPU helyett RES\_CPU megnevezéssel hivatkozhatunk a processzor időre).
* a setrlimit esetén a legtöbb esetben byte-ban illetve másodpercben adtuk meg az adott korlátokat, a grsecurity esetén már nagyobb mértékegységű értékeket is megadhatunk a megfelelő toldalék segítségével (például az 5K 5 kilobyte méretkorlátot jelent)

A projektet 2001 februárjában indították és az első kompatibilis kernelverzió a 2.4.1-es volt. A kernel szintű integrálása elég körülményes, csak az úgynevezett „vanilla”, hagyományos kernelekhez (ez azt jelzi, hogy senki sem változtatott rajtuk) adhatjuk hozzá. Ennek ellenére, több Androiddal kapcsolatos próbálkozás is napvilágot látott (AniDroid, Guardian-rom), de ezeknél a ROM-oknál a használható készülékek típusai nagyon korlátozottak, és az új funkciók közül nem az számunkra fontos komponensek használhatóvá tételére fektették a fejlesztők a hangsúlyt, így a grsecurity által nyújtott lehetőségekre sem alapozhatjuk a projektünket.

#### cgroups[8]

A cgroups (control groups) a Linux kernel egy olyan szolgáltatása, mely segítségével egyszerűen szabályozhatjuk és számlázhatjuk különböző folyamatcsoportok erőforrás-felhasználását. A cgroups segítségével különböző csoportokat hozhatunk létre, melyek hasonló tulajdonságokkal rendelkező folyamatokból állnak. A csoportokat a rendszer hierarchikus fa struktúrába rendezi. A gyermek csoportok öröklik a szülők számára beállított korlátokat. Természetesen lehetőség van további szűkítő korlátozások megadására is. A kernel több vezérlő segítségével menedzseli a csoportokkal kapcsolatos információkat. A projekt elsődleges célja egy egységes interfész létrehozása volt, mely a következő feladatok elvégzését támogatja:

* Erőforrások korlátozása (processzor idő, memória használat, háttértár IO)
* Priorizálás: adott folyamat csoportnak nagyobb használati jog biztosítása
* Számlázás: felhasznált erőforrások mérése például későbbi értékesítés céljából
* Kontrollálás: az adott rendszer felügyelete

A cgroups-t 2006-ban a Google mérnökei kezdték el fejleszteni. Először a 2.6.24-es kernel verzióba integrálták a megoldást, azóta számos új szolgáltatással és vezérlővel bővült a projekt. Az Android operációs rendszerben is megtalálható ez a funkció. Viszont a fejlesztők úgy tervezték a rendszert, hogy ezt a funkciót a rendszer módosíthatja, felhasználói szinten csak olvashatóak a szükséges fájlok és könyvtárak, ennek megváltoztatásához speciális feltételek kellenek az adott készülékkel kapcsolatban, és mivel ezek nem minden készüléken biztosítottak, így ezt a megoldást a projekt kapcsán nem használhatjuk.

#### PAM (Pluggable Authentication Modules) [9]

A PAM tulajdonképpen olyan megosztott könyvárak sokasága, amely lehetővé teszi a helyi rendszergazda számára annak eldöntését, hogy milyen módszerrel történjen a felhasználói azonosítás a két érintett fél között (felhasználó és alkalmazás). A projekt célja egy ilyen funkciót megvalósító szoftver létrehozása és a hozzá szükséges biztonsági valamint azonosítási modulok fejlesztésének elkülönítése. A kezdeti funkciókat a későbbiekben bővítették, így már lehetséges a rendszerben levő felhasználók erőforrás szabályozása is hála a pam\_limits modulnak.

A modul által megvalósított szabályozás eltér az eddig bemutatott megoldásoktól. Az előző projektekben az erőforrás korlátok megadása közvetlen módon történt, ezzel ellentétben a modul segítségével csak közvetve módosíthatunk. A modul segítségével megjelölhetünk egy alternatív limits.conf nevű konfigurációs fájlt, mely felírja az alapértelmezett korlátokat tartalmazó fájlban foglaltakat. Magát a régi fájlt nem módosítja a modul, így probléma esetén visszaállítható a beállítás az eredeti értékekre.

A PAM megvalósítás először a 3.0.4-es Red Hat Linuxban jelent meg 1996-ban. Később az X/Open UNIX részeként standardizálták. Mára számos más operációs rendszer is integrálta a különböző Linuxokon kívül (például a FreeBSD).

A 2.2.1-es fejezeten belül több, különböző Linux operációs rendszerekben használt erőforrás-szabályozás megoldást ismertettem. A projektek között több olyan van, amit a saját feladatunk számára is részleges, vagy némi átalakítás után teljes megoldást jelenthetne a felvázolt hiányossággal kapcsolatban. Viszont azt vegyük észre, hogy ezen megoldások mindegyike kernel szintű módosítást igényelne (kivéve a cpulimit-et, de az ezzel kapcsolatos problémát a 2.2.2.1 a) részében ismertetem). Elviekben kivitelezhető lenne valamelyik megoldás integrálása az Android rendszer kernelébe, de az eddigi tapasztalataim szerint az ilyen jellegű feladatok óriási szakmai tudást és erőforrás kapacitást igényelnek, a folyamat nagy valószínűséggel még a Diplomatervezés tárgy keretein is túlmutatna. Emellett, ha mégis sikerülne megoldást találni rövid időn belül, akkor is a DroidLab projekt szempontjából nem lenne használható, mivel a kernel módosítása mindenképpen egy saját ROM létrehozását vonná maga után, amit a mérésben résztvevő eszközökre telepíteni kellene. Egy ROM csere egy egyszerű felhasználó számára komoly feladatnak minősül, így nem várható el, hogy a mérés kedvéért mindenki lecserélje készüléke firmware-t.

Következésképpen olyan megoldást kell találni a problémára, amely szoftveresen a kliensen belül kivitelezhető, és a felhasználónak nem okoz extra tevékenységet a használatának biztosítása.

### Szolgáltatás-csoportok és kvóták

Az általunk tervezett erőforrást korlátozó megoldást kvóta rendszernek neveztük el. A kvóták testesítik meg a felhasználó által felajánlott erőforrásokat. A kvóták a pluginokra jellemzőek, lesznek olyan pluginok, melyekre egyáltalán nem vonatkozik kvóta és lesznek olyanok is, melyek többféle kvótát is definiálnak. A felhasználó az általa biztosítani kívánt kvóta mennyiségeket az erőforráshoz kapcsolódó plugin felhasználói felületén szabályozhatja, hasonlóan a már korábban említett funkcionális korlátozáshoz. A felhasználó döntésének megkönnyítése érdekében egyes erőforrásokhoz néhány tipikus értéket választunk ki. Így a felhasználók érezni fogják a kvóták nagyságrendjét és tudatos döntést tudnak hozni.

A modulok is rendelkeznek kvótákkal, ezeket azonban nem a felhasználó szabályozza. A központi vezérlő feladata, hogy a felhasználók által felajánlott kvótákhoz modulokat rendeljen.

A kvóta rendszer megvalósításához szükség volt a lehetséges használható erőforrások számba vételére, ezeknek pluginokba rendezésére illetve a kvóták jellegének és gyakoriságának meghatározására (ahol ez szükséges). Úgy láttam megfelelőnek, hogy hogy a legtöbb erőforrást egyéni pluginban kell megvalósítani, mivel így használható ki maximálisan a moduláris kialakításból származó előny.

#### Eszközmonitorozó pluginok

Az eszközmonitorozó pluginok közé tartoznak azok a megvalósított pluginok, melyeknek feladata, hogy az adott mérőeszköz fizikai állapotáról információt gyűjtsenek. A pluginokhoz nem terveztem külön kvóták definiálását, de ha a későbbiekben esetleg problémát okoz a felhasználóknak e pluginok energia fogyasztása, akkor a lekérdezések gyakoriságára vonatkozó korlátozási lehetőséget fogunk biztosítani.

1. **Memória-használat**: a plugin segítségével feljegyezhetővé válna a készülék memória kihasználtságának értéke. Az ilyen jellegű beállítási lehetőségek a projekt kezdeti szakaszában nem kerülnek implementálásra. Későbbiekben, ha lesz rá igény és kapacitás, akkor egy frissítés keretein belül pótoljuk őket (az ehhez szükséges kvótákkal együtt).
2. **Akkumulátor statisztika**: az Androidnak külön menedzser osztálya van az akkumulátor számára. A plugin segítségével számos adatot tudhatunk meg a készülék akkumulátorával kapcsolatban (például a töltöttségi szint, vagy a töltés típusa). Ezeket az adatokat más pluginok által szolgáltatott információkkal párosítva hasznos következtetéseket vonhatunk le például a felhasználói szokásokkal kapcsolatban.

#### Lokációs és hálózati információkat szolgáltató pluginok

A mobil eszközök legnagyobb előnye a nevükben is benne levő mobilitás, hordozhatóság. Ennek köszönhetően ezek a készülékek folyton velünk vannak, ez alkalmassá teszi őket a lokációs és hálózati információk gyűjtésére. Mivel az adatok nyeréséhez használt eljárások viszonylag energiaigényesnek mondhatóak, ezért a lekérdező műveletek számát és gyakoriságát mindenképpen korlátozni kell az kvóta rendszer segítéségével.

1. **GPS (Global Positioning System)**: GPS vevő ma már minden mobil eszközben megtalálható, mivel a vevő kedvező ára és kellően kis mérete ezt lehetővé teszi. Kezdetben GPS-t csak a helymeghatározásra alkalmazták, majd megjelentek a GPS alapú navigációs alkalmazások, jelenleg pedig az úgynevezett helyfüggő szolgáltatások kerülnek előtérbe. A vevő a Föld körül keringő műholdakkal kommunikál és a gyűjtött adatokból hozza létre a szükséges információt. Az általunk tervezett plugin elsősorban a készülék pozícióját és a műholdakkal kapcsolatos információkat képes mérni az idő függvényében. Ezekből utána más származtatott adatokat is képesek vagyunk előállítani, például két rövid időn belül mért egymás utáni pozícióból lehetséges sebességet becsülni. Jelenlegi tervek szerint a pluginhoz tartozó kvóták számát napi limitben fogjuk meghatározni és a lekérdezések közt eltelt időt pedig percekben.
2. **cella információ**: egy aktív mobil eszköz és a szolgáltató hálózata folytonos kommunikációban áll egymással. A kommunikáció során az adott cella számos információt szolgáltat a készülék számára (például térerő, operátor név, egyéb hálózati események). Az információk egy része automatikusan érkezik, a többit a készüléknek van lehetősége lekérdezni. A cellainformációk nem csak hálózati adatokat tartalmaznak, lehetséges például helymeghatározást is végezni ezek alapján. Bár ez nagy valószínűséggel nem lesz olyan pontos, mint egy GPS alapú, viszont elképzelhető olyan eset, amikor valamilyen ok folytán csak ez használható. Az általunk létrehozott plugin összes cella által szolgáltatott információ gyűjtésére alkalmas. Mivel az említett kommunikáció egyébként is része telekommunikációs szolgáltatás nyújtásának, ezért kisebb energia szükséges a használatához, mint a GPS esetén. Így a tervezett extra lekérdezések számát másodperces intervallumban tervezzük korlátozni.

#### Adatforgalmat bonyolító pluginok

Kétféle vezetéknélküli adatkommunikációt különböztetünk meg a mobil eszközökkel kapcsolatban. Vannak rövid hatótávolságú, kisebb adat sebességű megoldások illetve vannak nagy átviteli sebességű és hatótávolságú kommunikációt megvalósító szabványok. Amíg a rövid hatótávú megoldásoknál a maga a használat, ami legjobban energia igényes, addig a nagy hatótávolságú megoldások esetén maga a forgalmazott adat, aminek korlátokat kell szabni a kvóta rendszer segítségével.

1. **hálózati kommunikáció**: a hálózati kommunikációért felelős pluginban a két nagy hatótávolságú szabvánnyal, 3G-vel és a Wi-Fi-vel kapcsolatos funkciókat valósítjuk meg. A két szabványt nem választottuk szét, mivel nagyon sok közös tulajdonsággal rendelkeznek. Viszont teljes mértékben nem kezelhetjük őket egyenlőként. A 3G-t a mai világban mobil internetnek is nevezzük, ezt a szolgáltatást külön igényelheti a felhasználó a szoláltatójától, aki ezért külön díjat számláz fel. Ezért ennek az erőforrás használatának tökéletes szabályozása az egyik kardinális pontja a kvóta rendszernek. A pluginban definiált kvóták elsősorban az adatforgalom méretének korlátozását szolgálják.
2. **Bluetooth**: a Bluetooth egy 1994 óta létező, rövid hatótávolságú, adatcseréhez használt vezetéknélküli szabvány. Alkalmazásával számítógépek, mobiltelefonok (telefonkihangosítók) és egyéb készülékek között tudunk rádiós kapcsolatot létrehozni. A plugin segítségével e kapcsolatok tulajdonságait szeretnénk mérni. Az ilyen jellegű információk számos jelenleg is futó kutatással kapcsolatban (például késleltetés-toleráns hálózatok) jól hasznosíthatóak. A Bluetooth kapcsolat kiépülése több egymástól jól elkülöníthető fázisra bontható (például eszköz felderítés vagy párosítás). Ezek a fázisok külön funkciókban lesznek megvalósítva. Ezekhez a funkciókhoz definiálunk kvótákat, melyek az adott funkcióban végzendő tevékenység napi korlátját határozzák meg.
3. **NFC** **(Near field communication)**: a NFC, rövid hatótávú kommunikációs szabványgyűjtemény okostelefonok és hasonló (általában mobil) eszközök között, egymáshoz érintéssel vagy egymáshoz nagyon közel helyezéssel (maximum pár centiméter) létrejövő rádiós kommunikációra. NFC nagyon alacsony sebességű adatátvitelt tesz lehetővé, de a kapcsolat extrém gyorsan jön létre két NFC kompatibilis eszköz között. A plugin ebben az esetben is a kommunikáció adatait rögzítené. Mivel az adatátvitel jelenti az NFC-vel kapcsolatos szűk keresztmetszetet, ezért az átviteli fájl méretét kell korlátozni.

#### Felhasználó orientált pluginok

Ebbe a csoportba tartoznak azok a pluginok, melyek konkrétan a felhasználóval kapcsolatos adatok gyűjtését végzik. Ezeknél a pluginoknál nem a különböző erőforrások használata jelenti a problémát, hanem a személyes adatok kezelése és védelme. A kliens kialakítása során az egyik elsődleges szempont a felhasználók anonimitásának biztosítása volt, ennek ellenére ezeknél a pluginoknál a definiált funkciók korlátozásának megfelelő megvalósítása létfontosságú követelmény. Indokolt kvóta definiálását nem láttam szükségesnek ezeknél a pluginoknál.

1. **Telefónia**: telefónia alatt a felhasználó telekommunikációs szokásainak összességét értjük. A megvalósított plugin segítségével vizsgálhatjuk az adott telefonon folytatott hívások illetve SMS-k (Short Message Service) gyakoriságát, vagy a két előbb említett esemény között eltelt időt, áltagos híváshosszakat különböző napszakokban…
2. **Alkalmazások**: a létrehozott plugin célja, hogy a mérőkészülékre telepített alkalmazásokról minden lehetséges információt kinyerjen (telepített alkalmazások száma, csomagok neve, hozzáférési jogok…).
3. **Kontaktok**: a definiált plugin a kontaktok és a készüléken használt közösségi hálózatokból nyert információkat gyűjti későbbi statisztikai és egyéb célú felhasználás céljából

#### Egyéb készülékfüggő szenzor plugin

Utolsó csoportként következzen az a speciális plugin, amely az eddig nem ismertetett szenzorok felhasznosítását valósítja meg. Ezek a szenorok nem találhatóak meg minden eszközben, általában készülék illetve gyártó függőek. Az Android rendszer lehetőséget biztosít az ilyen szenzorok felderítésére. A felderítés után a kvóta rendszerrel már beállítható a megfelelő korlát mennyiség.

Ezzel végére értem a mérőrendszerrel kapcsolatos információk ismertetésének. A következő fejezet fő témája azon eszközök részletes ismertetése lesz, melyek segítségével a felhasználókkal kapcsolatos rendszerigények (részvétel és erőforrás-megosztás) rendelkezésre állását szeretnénk növelni.

# Gamification

Ebben a részben ismertetem a gamification-nel kapcsolatos fogalmakat módszerben rejlő lehetőségeket.

## A gamification fogalma

A gamification egy olyannyira új tudományág, hogy eddig még nem sikerült mindenki által elfogadott, általános definíciót megfogalmazni vele kapcsolatban. A leggyakrabban a következő leírással próbálják meghatározni a lényegét: gamification-ön azt a folyamatot értjük, mikor játékokra jellemző elemeket és játéktervezési mechanizmusokat használunk valamilyen nem játékhoz köthető környezetben[10]. Csak akkor beszélhetünk ténylegesen a gamification-ról, ha az előbbi leírásban elhangzottak mindegyike teljesül.

Mielőtt a módszer részleteivel is megismerkednénk érdemes tisztázni pár dolgot a gamification-nel kapcsolatban, mivel sokak számára nem világos mit is jelent pontosan e kifejezés. Gyakran keverik össze más játékokhoz köthető fogalmakkal. Először is a gamification nem szándékozik minden tevékenységet játékká változtatni, sőt számos szempontból pont az ellenkezőjét szeretné elérni. A tényleges cél, hogy az alkalmazott módszerek által egy adott cselekvés sokkal élvezetesebbé váljon a cselekvő számára, vagy sokkal nagyobb motivációt érezzen az adott tevékenység elvégzésére. A gamification alkalmazásáról van szó akkor is, ha egy unalmas feladat kapcsán sikerül egy olyan részfeladatot találni, melyen különböző játékelemeket alkalmazva izgalmasabbá válik a munka. Tulajdonképpen a gamification felfogható egy hatékony eszköztárként, mely segítségével különböző valós (általában üzleti vonatkoztatású) problémákat oldhatunk meg.

A leggyakrabban a gamification-t az úgynevezett komoly játékokkal (serious games) tévesztik össze. A két kifejezés közti legnagyobb különbség, hogy az előbbivel ellentétben az utóbbi ténylegesen egyfajta játékot takar, melyet valamilyen virtuális környezetben generált probléma megoldásának gyakorlására hoztak létre. Ezzel ellentétben a gamification esetén az adott problémákat a valóságban kell megoldani.

## Gamification kategóriák

A gamification három fő kategóriára osztható attól függően, hogy mit szeretnénk elérni az alkalmazásával. Ez a három kategória a következő: internal („belső”), external („külső”) és behavior change („viselkedést változtató”) gamification.

### Internal gamification

Internal gamification-ről általában valamilyen vállalat vagy szervezet kapcsán beszélhetünk, gyakran szokták vállalati gamification-nek is nevezni. Ebben az esetben a gamification célja a szervezeten belüli termelékenység javítása az alkalmazottak motiválása segítségével.

Az internal gamification-nek két fontos sajátossága van a másik két kategóriához képest. Az első, hogy a résztvevők már tagjai egy meghatározott csoportnak, ezáltal egy bizonyos szinten ismerik egymást, gyakran közös érdek vezérli őket. Erre jó példa a Microsoft Honosítás Minősítési Játék (Microsoft Language Quality Game[11]). Röviden összefoglalva: a cégen belüli hálózaton elérhetővé tettek egy alkalmazást, mely segítségével a még fejlesztés alatt álló Windows7 különböző honosításaiban lehetett hibát keresni. A dologban az az érdekes, hogy az alkalmazottak önkéntesen vettek részt a hibák keresésében, pedig nem járt jutalom az extra munkáért. Ez azzal magyarázható, hogy az alkalmazottak érdeke is a jó minőségű szoftver kiadása, nemcsak a cégé. Az indítvány olyan sikeres lett, hogy egy hónap leforgása alatt több mint 4600 alkalmazott 6000-nél több hibát jelentett be, melyek közül több száz tényleges javítást igényelt.

A második jellegzetesség az, hogy a motivációs késztetésnek összhangban kell lennie a cégen belüli menedzselési és jutalmazási rendszerrel. Az Microsoft példája azért is volt ennyire hatékony, mert a résztvevők nem erre a munkára voltak felbérelve. Tulajdonképpen mondhatjuk azt, hogy a céghez fűződő loyalitásuk miatt vettek részt a hibakeresésben, nem pedig azért, mert a megélhetésük függött tőle. Természetesen más motiváció is létezik a munkáltató iránt táplált hűségen kívül, például a munkatársak közötti versengés vagy egy új képesség megtanulásának lehetősége.

A DroidLab számára az internal gamification a kvóták méretének növelésében játszik szerepet. Az általunk megvalósított megoldás célja a projektben résztvevő felhasználók ösztönzése arra, hogy az általuk korábban beállított értékeket növeljék.

### External gamification

External gamification-t általában valamilyen marketing céllal létrehozott alkalmazások kapcsán szoktak alkalmazni. A cél leendő ügyfelek gyűjtése és meglévő vásárlók megtartása, márkához fűződő kötődésük növelése.

Az external gamification-re jó példa egy kaliforniai napilap, a Record Searchlight története. Az utóbbi időben a társadalom olvasási szokásai nagymértékben megváltozott, papír alapú újságok egyre jobban a háttérbe szorulnak és a digitális terjesztésű tartalmak kerülnek előtérbe. A Record Searchlight menedzsment részlege is belátta, hogy nem tud ez ellen az új trend ellen küzdeni és stratégiát váltott. Új célként kitűzte, hogy egy stabil olvasói közösséget épít ki magának egy saját weboldalt használva. A legnagyobb kihívást az jelentette, hogy azt oldalt ritkán látogató felhasználókat a közösség aktív és megbecsült tagjává tegyék. A probléma megoldásaként bevezettek egy úgynevezett jelvényrendszert. Bár elsőre nem úgy tűnhet, de a jelvényeknek nagyon jelentős motivációs hatása lehet. Három hónap alatt az oldal látogatottsága 25%-kal növekedett és a cikkekhez írt hasznos kommentek száma is gyarapodott. A példa tanulságát levonva az external gamification egy gazdag eszköztárat biztosít a felhasználói szokások megértéséhez és a motiváció befolyásolásához.

A tervezett mérőrendszerrel kapcsolatban is fontos az external gamification, hiszen létfontosságú a megfelelő felhasználói kör kialakítása a működés szempontjából. A DroidLab kapcsán az elsődleges terv egy jelvényrendszer implementálása, ami teljes mértékben a mérőrendszerhez lesz alakítva. A megszerzett jelvényeket a felhasználó a tervek szerint saját készülékén is és a webes felületen is megtekintheti. Emellett terveink között van a közösségi oldalak bizonyos szintű integrációja is, mely segítségével az adott felhasználó közzé tehetné az elért eredményeit. Természetesen a jelvényrendszeren kívül számos más eszköze létezik az external gamification-nek, melyeket egy részét mi is alkalmazni fogjuk. A további lehetőségeket a 3.3-as fejezetben láthatnak.

### Behavior change gamification

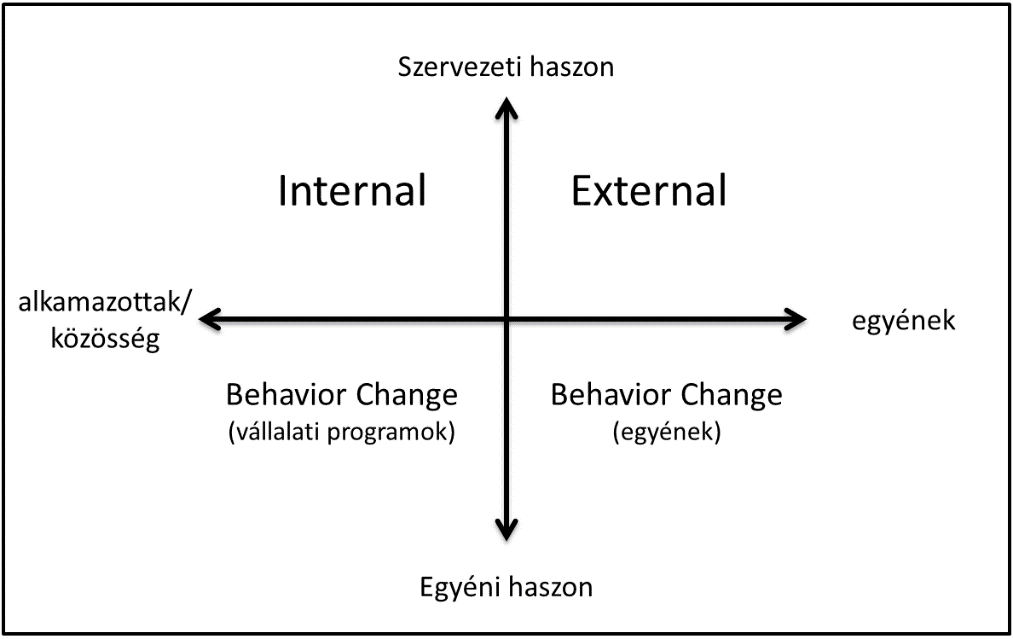
Behavior change gamification alkalmazása során arra törekszünk, hogy minél nagyobb társadalmi körökben új, hasznos viselkedési szokásokat terjesszünk el. Ilyen hasznos szokás alatt számos dolgot érthetünk, következzen néhány példa:

* egészséges életmód folytatása (egészséges étkezés, napi sportolás)
* új oktatási módszerek bevezetése
* hatékony megtakarítási technikák alkalmazása
* környezettudatos viselkedés elsajátítása

Hosszútávon ezeknek a változtatásoknak jelentős pozitív hatásai lehetnek. Az egészséges életmód elterjedése köveztében feltehetőleg csökkenne a túlsúlyos emberek és az egészségügyi ellátásra szorulók száma, az oktatási reformok következtében növekedne az oktatás hatékonysága, a megfelelő pénzhasználat elsajátítása következtében tudatosabb pénzügyi döntések születnének és környezettudatosság elterjedésével csökkente az egy főre jutó ökológiai lábnyom értéke. Az ilyen jellegű kezdeményeket az állam vagy valamilyen nonprofit szervezet szokta támogatni.

A projektünk szempontjából ez a gamification típus nem annyira releváns, mint ez előző kettő. Nem áll szándékunkban a felhasználók bármilyen jellegű szokásait megváltoztatni. Esetleges hasznot akkor hozhat, ha az egész elképzelést kisebb csoportra vetítjük le és nem állandó, csak ideiglenes viselkedésbeli változást szeretnénk elérni (például a már említett Bluetooth mérés kapcsán). A projekt kezdeti szakaszában ezzel az eshetőséggel nem nagyon számolunk, viszont a megvalósítás úgy lesz kialakítva, hogy szükség szerint kisebb módosítások bevezetése mellett könnyen megvalósítható legyen.

Az fejezet zárásaként következzen egy ábra, melyen a különböző kategóriák egymással való kapcsolata látható (3.1. ábra):



3.1. ábra Gamification kategóriák közötti kapcsolatok

## A gamification eszköztár

Ahogy az az eddig bemutatott példákból illetve az általam adott „definícióból” is megállapítható, a gamification sikeres használatának egyik feltétele a megfelelő játékelemek kiválasztása és helyes alkalmazása. Ezt még a folyamat kezdeti szakaszában érdemes megtenni, de a megfontolt döntés meghozásához ismerni kell a lehetőségeket. Az fejezet további részében az eszköztár bemutatása következik.

### Az elemek hierarchikus piramisa

A játékelemeket három hierarchikusan egymásra épülő csoportra oszthatjuk. Minél nagyobb egy adott csoport szintje, annál kevesebb elemet tartalmaz.

#### Játék dinamizmusok

A játék dinamizmusok olyan játékra és a játékosra vonatkozó, időben változó folyamatok és módszerek, melyeknek köszönhetően a játék (vagy a játékosított tevékenység) élvezhetőbb lesz. A dinamizmusok elemek hierarchiájának legfelsőbb szintjén találhatók. A legfontosabb dinamizmusok a következőek:

1. Korlátok (korlátozások vagy kényszerített kompromisszumok)
2. Érzelmek (kíváncsiság, csalódottság, boldogság)
3. Elbeszélés (egy következetes, folyamatos történet)
4. Fejlődés (a játékos karakterének fejlesztése)
5. Kapcsolatok (társadalmi interakciók, bajtársiasság, altruizmus)

#### Játék mechanizmusok

A játék mechanizmusok szabály alapú rendszerek, melyek cselekvésre ösztönzik a felhasználót. A szakirodalom tíz fontos mechanizmust szokott kiemelni:

1. Kihívások (rejtvények és egyéb megoldásra váró feladatok)
2. Véletlen (elemi véletlenszerűség)
3. Vetélkedés (nyertesek és vesztesek)
4. Együttműködés (összedolgozás másokkal a cél elérése érdekében)
5. Visszacsatolás (információnyújtás a játékos tevékenységéről)
6. Erőforrásgyűjtés (hasznos és gyűjthető dolgok szerzése)
7. Jutalmak (juttatás valamilyen tevékenységért)
8. Tranzakciók (játékosok közötti kereskedelem)
9. Körökre osztás (játékosok cselekvésének korlátozása)
10. Győzelmi feltételek

Mindegyik mechanizmus valamilyen módon kötődik legalább az egyik dinamizmushoz az előbb felsorolt öt közül. Például egy figyelmeztetés nélkül felugró ablak (2. mechanizmus) kíváncsiságot (2. dinamizmus) válthat ki a felhasználóból. Ebből az is következik, hogy a mechanizmusok a dinamizmusok alatt helyezkednek el a hierarchián belül.

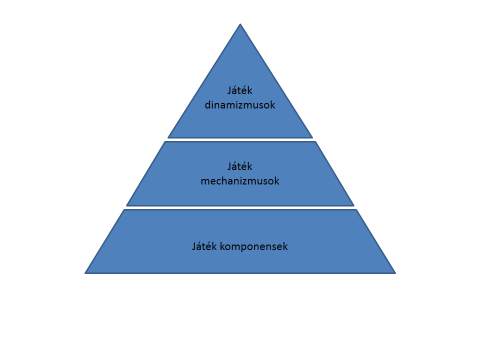
#### Játék komponensek

A hierarchia legalsó szintjén tálalhatóak a játék komponensek. Ezek a játékelemek adnak példákat és módszereket arra, hogy valósíthatjuk meg azokat a dolgokat, amelyeket a dinamizmusok és mechanizmusok reprezentálnak. Tizenöt fontos komponens különböztetünk meg:

1. Eredmények (meghatározott célok)
2. Avatárok (játékos karakterének vizuális reprezentációja)
3. Jelvények (eredmények vizuális reprezentációja)
4. Főellenség harcok (kifejezetten nehéz kihívás az adott pálya zárásaként)
5. Gyűjtemények (gyűjthető jelvények illetve tárgyak összessége)
6. Harc (meghatározott csaták)
7. Feloldható tartalmak
8. Ajándékozás (erőforrás-megosztás lehetőségének biztosítása)
9. Ranglisták (a játékos fejlődésének és eredményeinek vizuális megjelenítése)
10. Szintek (meghatározott lépcsőfokok a játékos fejlődésében)
11. Pontok (játék fejlődés numerikus reprezentációja)
12. Küldetések (előredefiniált kivívások konkrét célokkal és jutalmakkal)
13. Szociális hálózatok (a játékosok játékon belüli kapcsolatainak reprezentálása)
14. Csapatok (meghatározott játékosok egy csoportja, akik egy közös cél elérése érdekében együtt dolgoznak)
15. Virtuális javak (névleges vagy valódi értékkel bíró játékbeli tőke)

A mechanizmusokhoz hasonlóan a komponensek mindegyike egy vagy több felsőbb rétegbeli elemhez szorosan kapcsolódik.

A 3.3.1 fejezet zárásaként következzen egy vizuális reprezentációja az ebben a részben elhangzottaknak (3.2. ábra).



3.2. ábra: Az elemek hierarchikus piramisa

### A „PBL hármas”

A PBL a Points (pontok), Badges (jelvények), Leaderboards (ranglisták) játékelemek rövidítése. Vegyük észre, hogy mindhárom felsorolt elem a komponensek közé tartozik. Azért érdemes a PLB-vel külön foglalkozni, mivel az esetek többségében egy jó kiindulást biztosít egy gamification-t alkalmazó saját rendszer kialakításához.

A PBL nagyon hatékony és praktikus komponens, ha az ember megfelelően használja. De figyelembe kell venni azt is, hogy nem minden esetben garantálja az elvárt eredményt, megvan a maga korlátai. Ebből kifolyólag, ha maximálisan ki szeretnénk használni a gamification által nyújtott előnyöket, akkor szükséges egyéb komponensek igénybe vétele.

Mindezek ellenére a gyakorlat azt mutatja, hogy érdemes ezzel a három komponens vizsgálatával kezdeni a gamification folyamatát. Ehhez pedig tisztában kell lenni az egyes komponensek előnyeivel és hátrányaival.

#### Pontok

A pontok az egyik legalapvetőbb elemek közé tartoznak az olyan emberek motivációjával kapcsolatban, akik szeretnek dolgokat gyűjtögetni. Emellett nagyon egyszerűen lehet vele az adott játékos teljesítményét is mérni, így versengő felek közötti mérőszámként is hatékonyan alkalmazható. Természetesen egyéb más módon használhatjuk még a pontokat. A szakirodalom hat különböző esetet határozott meg, hogy milyen funkcionalitást tölthetnek be a pontok a gamification során:

1. **A pontok eredményt fejezhetnek ki**. Ez az egyik leggyakrabban alkalmazott használati mód. Ebben az esetben a pontok informálják a játékosokat, hogy hogyan végzik a tevékenységüket, például egy magasabb pontszám jelentheti azt, hogy az adott játékos többet játszott, vagy ügyesebben végezte a dolgát, mint az, akinek kevesebb pontja van.
2. **A pontok meghatározhatják, hogy mikor győztünk (ha létezik győzelmi feltétel).** Néha egyes folyamatok teljesítéséhez szükséges feltételeket pontokban határozzák meg a fejlesztők.
3. **A pontok kapcsolatot teremthetnek a végzett folyamat és az elérhető jutalmak között**. Számos gamification-t alkalmazó rendszerben a felhasználók az általuk gyűjtött pontokat valós vagy virtuális jutalmakra válthatják be.
4. **A pontok visszajelzést biztosíthatnak a játékosoknak**. A pontok segítségével nagyon egyszerűen hatékonyan kivitelezhető egy olyan visszacsatolást megvalósító mechanizmus, mely az adott játékosokat tájékoztatja, hogy milyen jól haladnak az adott tevékenységben.
5. **A pontok azonos tevékenységet végzők egyéni eredményeiről tájékoztatást nyújthatnak**. Ezek az értékek az résztvevők képességeit reprezentálhatják.
6. **A pontok adatokkal szolgálhatnak a fejlesztők számára**. A játékosok által elért pontok egyszerűen megfigyelhetőek és tárolhatóak. Később ezeket az adatokat a fejlesztők elemezhetik, a kapott eredményeket felhasználhatják a rendszer hatékonyságának növelésére.

Ha tisztában vagyunk, hogy mit szeretnénk elérni a gamification során, akkor könnyen meghatározhatjuk, hogy a pontokat milyen módon használhatjuk fel. Viszont ügyelni kell, hogy ne várjunk el többet a pontok funkcionalitásaitól, mint amire valójában képesek.

#### Jelvények

A jelvények már párszor szóba kerültek az elmúlt fejezetek során, most következzek egy kicsit részletesebb bemutatás. A jelvények voltaképp a felhasználó által elért eredmények vizuális reprezentációi. A jelvények és a pontok általában szoros kapcsolatban állnak egymással, gyakran definiálnak a fejlesztők olyan jelvényeket, melyek bizonyos mennyiségű pontok eléréséért járnak.

Kutatások szerint egy jól megtervezett jelvényrendszer öt motivációt növelő jellemzővel rendelkezik:

1. A jelvények egyfajta célt adnak a felhasználónak, akinek azáltal, hogy e célok elérésére törekszik, növekszik az adott tevékenységgel kapcsolatos elkötelezettsége.
2. A jelvények útmutatást nyújtanak a rendszer által nyújtott lehetőségekről a felhasználónak. Fontos, hogy ebben az esetben a felhasználó az általa még nem birtokolt jelvényeket is meg tudja nézni, illetve hogy e jelvények mellé fel legyen tűntetve a megszerzésük feltétele.
3. A jelvények a felhasználó által kedvelt dolgokat jelzik, illetve a tényleges tevékenységekkel kapcsolatos teljesítményekért járnak.
4. A jelvények egyfajta státuszszimbólumként szolgálnak.
5. A jelvények elősegítik a hasonló érdeklődést mutató személyek csoportba rendeződését.

A jelvények legfőbb tulajdonsága a rugalmasságuk. A jelvények számosságának és típusainak csak a fejlesztők képzelete és a játékosított tevékenység szab határt. Ebből következőleg a jelvények használatával sokkal változatosabb felhasználói kört tudunk megszólítani, mintha egy egyszerű pontrendszert vezettünk volna be. Bár a kategóriáknál bemutatott példában ez nem látszott, a jelvényeket érdemes internal gamification során is alkalmazni.

#### Ranglisták

A PBL utolsónak bemutatott tagja a ranglisták. A három bemutatott komponens közül ez a legproblémásabb, mivel gyakran „kétélű fegyver” válhat belőle. Egyrészről, vannak olyan játékosok, akik tudni szeretnék, hogy a többiekhez képest milyen teljesítményt sikerült elérniük. A ranglista erre a feladatra tökéletes. Nagyon sok játékost hajtja a versenyszellem, gyakran motiváló hatást fejthet ki az a tudat, hogy csak pár ponttal van lemaradva rangsorban előtte lévő embertől.

Másrészről, ha az adott játékos látja, hogy legjobbaktól milyen messze áll a rangsorban, az könnyedén elveheti a kedvét az adott tevékenység folytatásától. Éppen ezért nagyon fontos, hogy ne egyetlen statikus rangsort készítsünk, sőt a legjobb megoldás, ha néhány szempontot figyelembe véve összetett ranglistákat hozunk létre.

## A játékosok típusai

Ahhoz, hogy a gamification motiváló erejét felismerjük és megfelelő módon hasznosítani tudjuk, nem elégséges csupán a módszer eszköztárával tisztában lenni, szükséges a sikerhez a felhasználók tipikus viselkedési kategóriáinak ismerete is. E témával kapcsolatban is több említésre méltó elmélet is született, én ezek közül kettőt tartottam fontosnak, melyeket a fejezet további részében szándékozok bemutatni.

### A Bartle modell

Richard Allan Bartle egy angol professzor, aki a munkásságának javát a játékok világának szentelte, sokan az MMO (massively multiplayer online, nagyon sok szereplős interneten játszható) játékipar egyik úttörőjének tartják. 1996-os publikációjában[12] ismerteti a játékosok típusaival kapcsolatos elméletét. Az általa alkotott modell létrehozásakor tipikus játékosok által kedvelt dolgokat és a játékosok ezekhez való viszonyulását vizsgálta, és a kapott eredmények után a következő négy eltérő beállítottságú játékos csoportot definiálta.

#### Törtető (Achiever)

Az ilyen típusú játékosok elsődleges célja, hogy a játék keretein belül elérhető különböző javakból minél nagyobb mennyiséget szerezzenek. Ha a játék erre lehetőséget biztosít, akkor igyekeznek a játékon belüli magasabb szintek hatékony elérésére is. Számukra a játék mélyebb felfedezése csak szükséges kellemetlenség új erőforrások és javak gyűjtéséhez. A másokkal való kommunikációt általában egyfajta relaxációs tevékenységként űzik és az esetek többségében arra törekednek, hogy olyan hasznos információkat szerezzenek, amelyek a saját javaik gyarapodását segíti elő. Másik játékos elpusztítására csak akkor folyamodnak, ha a saját érdekeik úgy kívánják, például ellenkező esetben a másik játékos kapna helyettük jutalmat. A törtető játékosok általában azokat a játékokat szeretik, amelyek hosszú játékidő mellett folytonos fejlődési lehetőségeket biztosítanak mind a karaktert mind a javakat tekintve.

#### Felfedező (Explorer)

Ezt a játékos típust a felfedezés öröme készteti a játék folytatására, ezekre a játékosokra van a játék által nyújtott világ a legnagyobb hatással. Legfőbb tevékenységük a számukra biztosított területek felderítése, a különböző rejtett tartalmak (idegen szóval eater-eggs) és játékban maradt esetleges hibák észlelése. Szeretnek rájönni, hogyan működik a számukra biztosított, új virtuális világ. Nagyon sokat számít nekik a játékkal kapcsolatos minden apró részlet. A felfedező beállítottságú játékosok számára a különböző javak gyűjtése unalmas dolognak számít, egyedül akkor folyamodnak erre a tevékenységre, ha egyes területek csak bizonyos javak megléte után elérhetőek. A szociális lehetőségeket általában új kipróbálható ötletek szerzésére használják fel. Az ilyen típusú játékosok számára más játékosok elpusztítása nem okoz morális problémát, ennek ellenére kerülni szokták ezt a szituációt, mivel könnyen problémájuk lehet belőle, hiszen egy-egy felesleges gyilkolás egy hosszú játékmenet során könnyen megtorlásba végződhet. A felfedezők számára tervezett játékoknál mindenképpen ügyelni kell arra, hogy biztosítva legyen minden játékos számára az, hogy a saját tempója szerint haladhasson. Emellett kellő tartalommal és megfelelő szabályrendszerrel könnyen biztosítható az adott játék sikere az ilyen típusú emberek körében.

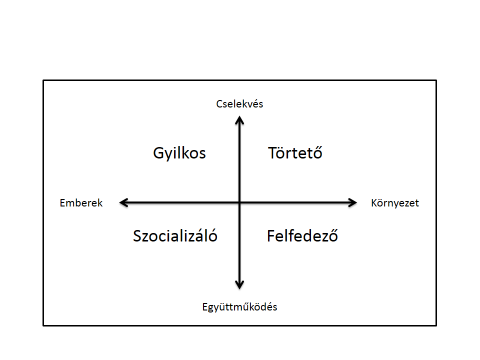
#### Szocializáló (Socialiser)

A szocializáló típusú játékosokat elsősorban az emberek érdeklik és legfőbb örömforrásuk a más játékosokkal kapcsolatos interakció. Az adott játék számukra csak másodlagos, szükséges dolog új emberek megismeréséhez. Nagyon fontos számukra a játékon belüli kapcsolatok ápolása (együttérzés más emberekkel, viccelődés, szórakoztatás). A legtöbb szocializáló játékos számára az az igazi jutalom a játékkal kapcsolatban, ha minél jobban megismerheti játékostársait. A javak gyűjtése számukra csak akkor fontos, ha ennek segítségével a közösségben elfoglalt helyüket tudják megerősíteni. A felfedezés számukra csak azért fontos, hogy a megszerzett információ birtokában megfelelő társalgó partnerek lehessenek a többi játékossal. A gyilkolás számukra csak nagyon ritka esetben elfogadható, például ha az adott pusztítás jelentéktelen, vagy ha megtorlás egy kedves barátért. Az ilyen típusú játékosokra specializált játékoknál egyetlen dologra kell odafigyelni, arra az ismerkedés lehetőségét a maximális szinten biztosítsuk a játékosok számára.

#### Gyilkos (Killer)

Olyan játékosok tartoznak ebbe a csoportba, akiket a mások feletti dominancia izgat, mások legyőzése késztet a játék folytatására. Az ilyen játékosok gyakran csak azért támadnak meg másokat, hogy végignézhessék az ellenfél karakterének halálát. Minél nagyobb szenvedést okoznak a megsemmisített ellenfélnek, annál jobban élvezik a dolgot és nem boldogítja őket annál a tudatnál semmi jobban, mint hogy valahol egy húsvér ember nagyon dühös azért, amit tettek vele és nem tud mit tenni ez ellen. Számukra csak azon játékban található javak gyűjtése a fontos, amelyek segítségével karakterüket fejleszthetik, hogy a későbbiekben minél nagyobb pusztítást és mészárlást vihessenek véghez. A játék felfedezését is csak az új célpontok találásáért és az új gyilkolási módszerek megismeréséért tartják fontosnak. A szociális csatornákat is legtöbb esetben csak az előző áldozataik gúnyolására használják, vagy az új kiszemeltek szokásairól próbálnak meg információt szerezni illetve ritkább esetben praktikákat váltanak más hasonló beállítottságú játékosokkal. A gyilkos számára a kellő mennyiségű és megfelelő képességű ellenfél a legfontosabb, ezek biztosításával garantálható az ilyen beállítottságú játékosok érdeklődésének fenntartása.

Az előbb bemutatott játékos csoportok egymáshoz fűződő kapcsolata az alábbi ábrán látható (3.3. ábra):



3.3. ábra: Bartle által meghatározott játékos típusok

# Tervezés, rendszerkomponensek implementációja

Ebben a fejezetben ismertetem az elkészült projekttel kapcsolatos, rám szabott feladatok megvalósításának lépéseit beleértve a tervezést és a tényleges implementációt. Mivel ez egy csoport munka, amin több hallgató, beleértve magamat is, párhuzamosan dolgozik egyszerre, ezért a teljes keretrendszer részletes megvalósításának bemutatására nem kerül ebben a dolgozatban sor, csak azokat a fontos elemeket fogom röviden ismertetni, amellyel a saját munkám során dolgoznom kellett. Ennek ellenére úgy gondolom, hogy a 2.1.2 fejezetben ismertetett információk elegendő tudást biztosítanak a fejezet további részében leírtaknak teljes körű megértéséhez.

Már a bevezetésben ismertettem, hogy melyik az a két fontos és logikailag összefüggő feladatkör, amelynek problémáit kellett megoldanom a diplomaterv készítése alatt. Emellett a feladatkiírás szerint a modulok erőforrás igényének becslésével is foglalkoznom kellett a tervezés során, így az aktuális fejezetben erre is részletesebben ki fogok térni. Most pedig kezdődjön a tervezés és implementálás folyamatának bemutatása.

## Erőforrást kezelő komponens

A 2.2 fejezetben már megismerkedhettünk az erőforrás kezeléshez köthető szükséges elméleti információkkal. A fejezet során bemutattam az alap koncepciót, amit szerettünk volna megvalósítani a mérőrendszerrel kapcsolatban, illetve részleteztem a dolgozat megírásának pillanatában létező, esetlegesen használható megoldásokat is. Ezeket tanulmányozva be kellett látnunk, hogy mindenképpen saját, egyedi megoldást kell megvalósítani a feladat megoldására. Első lépésként definiálni kellett a saját szabályozó rendszertől elvárt technikai részleteket:

* a tervezett rendszerkomponens támogassa a pluginokhoz kitalált kvóta mennyiségeket, és tartsa be annak határait
* emellett a modulok processzor használatát is szabályozza minél nagyobb pontossággal
* megfelelő felhasználói felületet biztosítson a szabályozás menedzseléséhez a készülék tulajdonosának

A telefon egyedi erőforrásaival ellentétben a gyári Android rendszer nem biztosít hasonló szintű szabályozási lehetőséget a processzort tekintve. Ezért a szabályozó komponenst további két részre osztottam, egyik a processzor használatáért másik a pluginokban megvalósított metódusok megengedett számú meghívásáért felelős. Most következzen először a processzor használat szabályozásának a bemutatása.

### Processzor használat szabályozás

Mint azt már korábban említettem a processzorral kapcsolatos gyári beállítási és szabályozási lehetőségek elég szegényesek, így az Android által nyújtott opciók között nem találtunk számunkra használható megoldást. Az egyetlen megmaradt lehetőség az volt, hogy valamilyen módon ki tudjuk használni a platform Linuxhoz (Unixhoz) fűződő kapcsolatát. Azt a 2.2.1-ben láthattuk, hogy számos lehetőség lett volna, amelyet alkalmazhatnánk, ha egy az egyben megegyezne az asztali gépek Linux rendszere a telefonon futtatottéval, de sajnos ez nem így van. Viszont a számítógépeken használható módszerek működésének vizsgálata során sikerült olyan megfigyeléseket tenni, amelyek ötletet adtak egy saját módszer kialakításához.

A Linux alatt számos olyan rendszerparancs van (például a top vagy a ps), amely az adott rendszeren futó folyamatokat monitorozza. Ezeknek a parancsoknak is valahonnan nyernie kell azokat az adatokat, amelyeket a felhasználó számára megjelenít. Az említett parancsok dokumentációját megvizsgálva könnyű volt megtalálni a szükséges információk forrását. Minden Unix alapú operációs rendszernek van egy proc elnevezésű pszeudo-fájlrendszere, melyben a folyamatokkal kapcsolatos információkat tárolja a rendszer. Ezt szokás szerint a /proc helyen lehet elérni az adott operációs rendszer könyvtár szerkezetében. Érdemes megjegyezni, hogy a struktúra tartalmára csak olvasási engedélyt biztosít a Linux, de ez számunkra bőven elegendő. Minden folyamathoz tartozik egy saját könyvtárstruktúra, mely gyökérkönyvtárának neve a folyamat azonosítója (idegen szóval process identifier, pid). Így, ha ismerjük ezt az azonosítót, számunkra is elérhetővé válnak a tárolt adatok. Egy-egy folyamathoz tartozó fájlrendszerben nagyon sok információt tárol a rendszer, a teljes lista és hozzá tartozó magyarázatok a [13]-ban olvasható, én most a dolgozatban csak azokat fogom ismertetni, amelyek fontosak a saját megoldással kapcsolatban:

* /proc/uptime: két adatot tárolnak ebben a fájlban, az első az adott rendszer futási idejét (másodpercben), a második pedig az idle állapotban töltött időt tárolja (szintén másodpercben)
* /proc/[pid]/task: ez a könyvtár a Linux 2.6.0-test6-os verziójától érhető el, tartalma tulajdonképpen annyi alkönyvtár, ahány szállal rendelkezik az adott folyamat. Minden szálnak hasonló könyvtár struktúrát tart fenn a rendszer, mint a folyamatoknak, amíg az adott szál fut. Egyes elemek közösek a szálak fájlstruktúrájában a szülő folyamatéval (például a folyamat munkaterületére mutató szimbolikus link), de az adatok nagy része az adott szállal kapcsolatosan egyediek. A projekt szempontjából ez azért nagyon fontos, mivel a keretrendszer a futtatott modulokat, egyéni szálba indítja el a saját folyamatán belül, és ennek a könyvtárstruktúrának köszönhetően egyedi adatokat szerezhetünk a modulokkal kapcsolatban.
* /proc/[pid]/stat (/proc/[pid]/task/[tid]/stat): ezekben a fájlokban az adott folyamat vagy szál állapot információit tárolja a rendszer (a már említett ps rendszer parancs is közvetlen innen nyeri az információt). A stat összesen 44 különböző adatot tárol, ebből számunka következő három releváns:
  + utime(14): az adott idő, ameddig az adott folyamat/szál ütemezve volt a rendszerben felhasználó módban.
  + stime(15): az adott idő, ameddig az adott folyamat/szál ütemezve volt a rendszerben kernel módban.
  + starttime(22): az az idő, amikor az adott folyamatot/szálat indította a rendszer a legutóbbi rendszerfelállástól számítva.

Ezen információk tudatában már mi is képesek vagyunk a processzor használattal kapcsolatos számításokat végezni, a fenti adatok ismeretében például kiszámolhatjuk az adott folyamat/szál átlagos processzor használatát a következő módon:

\* 100 %

Miután meglett az, hogy hogyan nyerhető információ a modulok processzor használatával kapcsolatban, már csak azt kellett kitalálni, hogyan lehetséges a szabályozást véghezvinni. Az esetleges lehetőségeket végigmérve arra jutottam, hogy a legjobb és egyben legegyszerűbb megoldást választom, azaz addig kell altatni a túl aktív modulok szálait, amíg processzor használatuk a megengedett érték alá vissza nem csökken. A működőképes szabályozó rendszer megvalósításához három részfeladatot kellett megvalósítani.

#### A keretrendszer modul kezelésének kiegészítése

A leglényegesebb szempont az volt a megvalósítás során a helyes működés biztosítása mellett, hogy az újonnan integrált részek logikailag is illeszkedjenek a meglévő forráshoz, hogy esetlegesen a később projekthez csatlakozó kollégák könnyebben átlássák a rendszer egészét. Így kezdő lépésként a modulokkal kapcsolatos részt tanulmányoztam át, melynek során arra jutottam, hogy a legkézenfekvőbb megoldás egy új osztály és a hozzátartozó interfész létrehozása, amely tárolja és elérhetővé teszi a processzor használat kontrollálásához szükséges adatokat. A tervek szerint a következő tárolókat kell létrehozni az új osztályban:

* a modul szálának azonosítóját tartalmazó változó
* egy olyan változó, amely annak jelzését szolgálja, hogy az adott modul túllépte-e a beállított határt
* egy tároló, amelyben a régebbi mért adatokat tároljuk ideiglenesen

Az Android SDK (Software Developer's Kit) segítségével könnyen lekérdezhetjük egy adott készüléken futó szál egyéni azonosítóját, amelyet egy egyszerű int változóban tárolhatunk későbbi felhasználás céljából. Az esetleges nehézséget az okozza, hogy ezt csak az adott szál futása alatt maga a szál teheti meg, ebből kifolyólag meg kell oldani, hogy minden modul kötelezően lekérdezze a saját azonosítóját legalább egyszer az indítása után. Ezzel a problémával részletesebben a 4.1.1.3-ban fogok foglalkozni a továbbiakban. A szál azonosítójához egy beállító illetve egy lekérdező függvényt kellett implementálni.

Az állapotot jelző változót több módon is meglehetett volna valósítani (például legegyszerűbben egy boolean változóval), de a lehetőségeket mérlegelve a java-ban is implementált Semaphore osztály mellett döntöttem. Ennek legfőbb oka, hogy ezt sokkal bonyolultabb manipulálni mások által, mint a többi lehetséges megoldást, emellett gyakran alkalmazzák szálak korlátozásával kapcsolatban is. A Semaphore osztály általunk inicializált példányához hozzárendelhetünk tetszőleges mennyiségű engedélyt (idegen szóval permit-et), amiket később az adott példányon keresztül menedzselhetünk. Erre a feladatra számos saját függvénye van az osztálynak, melyről nekünk külön nem kell gondoskodni. A megoldás során minden egyes modulhoz egy saját, egy darab engedéllyel rendelkező Semaphore-t példányosít a keretrendszer, melynek az a feladata, hogy futás közben szabályozza a szál altatásáért felelős kódrészlet lefutását. A saját osztályba ezzel kapcsolatban három függvényt kellett implementálni, amelyek segítségével az említett, osztályhoz tartozó függvényeket tudjuk egyszerűen elérni. Egy az engedély lefoglalásáért felelős, egy a felszabadításáért, az utolsó pedig arról ad információt, hogy milyen állapotban van éppen az engedély.

A 4.1.1-ben bemutatott átlagos processzor használat kezdetben megfelelő monitorozási lehetőségnek bizonyult, de később a tervezés során sajnos rá kellett jönnöm, hogy ez a kiszámolt érték csak addig reprezentatív, amíg a megfigyelt szál futása folyamatos, és nem zavarja meg egyéb tényező, mint például az általam tervezett megoldásban az altatás. Ugyanis az ilyen esetekben az adott érték az idő függvényével ugyan csökkenni fog, de nem olyan ütemben, ahogy a valóságban történik. Hasonló a helyzet, miután újra aktívan kezd futni az altatott szál, annak ellenére, hogy akár a teljes processzor teljesítményt kihasználhatja, az átlagos érték csak viszonylag lassan fog növekedni, ami az esetünkben nem megengedhető. Ezért egy új megoldást kellett kitalálni a processzor használat mérésére. Végezetül arra jutottam, hogy használható vizsgálati módszer lenne az, ha meghatározott, állandó időközönként lekérdezné a keretrendszer a modul összesített ütemezett idejét (a már említett utime és stime összegét) és a két egymás után lekérdezett érték különbségéből illetve az állandó értékből megahatároznánk, hogy az adott szál az adott idő hány százalékában használta a processzort. Ahhoz, hogy a megfelelő mértékű szabályozást meg tudjuk valósítani, nem elegendő, ha csak az legutolsó két értéket használja a rendszer a vizsgálat során. Ezt a következő esetet végiggondolva láthatjuk be a legegyszerűbben: képzeljünk el egy nagy processzor igényű futó modult és egy alacsony processzor használat korlátot. Ha csak az utolsó két érték különbségét vizsgálnánk, akkor azt történne, hogy egymást követően cserélődnének a futó és altatásban levő időszakok a modullal kapcsolatban, mivel futás alatt biztos túllépi a határt, a másik esetben meg nagyon alacsony lesz a különbség. Ez azt jelentené, hogy áltagosan csak maximum a felére tudtuk lecsökkenteni a processzor használatot. Ez lehet, hogy a beállított korlátnál még mindig nagyobb, így elviekben helytelenül működne a szabályozás. A hasonló esetek kiküszöbölése érdekében a rendszer csúszó ablak szerűen figyeli a futó modulokat, és a közelmúltban mért adatokból számít átlagos értéket, így csillapítva az előbb említett lehetséges ingadozást. A szükséges adatok tárolására szolgál a saját osztályomban definiált tároló. A tényleges implementáció során egy float értékeket tároló listát használtam. A méret kiválasztásánál az volt az elsődleges szempont, hogy az előbb felvetett probléma ne jelenjen meg, de ennek ellenére túl nagy se legyen, mert akkor változás észlelése lehet túl lassú. A kezdeti implementáció során a méretet tizenötre definiáltam, a későbbi tesztelés során ez megfelelőnek tűnt, így a jelenlegi megoldásban ez az érték maradt. A lista karbantartásához négy saját függvényt implementáltam az osztályomban. Az első segítségével a listához adhatunk hozzá újabb tagot, a második a lista méretét adja vissza, a harmadikkal a lista adott helyen álló elemét kérdezhetjük le, az utolsó feladata pedig az, hogy ha lista mérete maximális, akkor a legrégebbi adatot törli a listából, helyet biztosítva az újnak (itt jelenik meg a csúszó ablak jelleg).

Miután a tervezett osztály és a hozzá tartozó interface implementálása megtörtént, a következő feladat ezzel a résszel kapcsolatban az új elemek keretrendszerbe történő integrációja volt. Először hozzá kellett rendelni az új komponenseket a meglévő Modul osztályhoz. Ennek legmegfelelőbb módja a Modul osztály paraméter listájának a saját osztállyal való kibővítése volt, illetve importálni kellett az új interface-t is. Ezek után minden olyan létező osztályt módosítani kellett, ahol a Modul konstruktorát használják. Végezetül a ModuleManager osztályban létre kellett hozni olyan tárolókat, amelyek a betöltött modulok új adatait tárolják.

Mindezen módosítást elvégezve a meglévő keretrendszer alkalmassá vált a processzor használat szabályozásához szükséges adatok megfelelő tárolására és menedzselésére. Ezek után folytathattam a munkát következő részfeladattal.

#### A processzor használatot szabályozó szál

Az adatok tárolásával még önmagában nem érünk el semmit, ha nincs meg a megfelelő feldolgozás. Mivel a modulok processzor használata nem kifejezetten eseményvezérelt, mint például a metódusok meghívása, így a megfelelő működés biztosítása érdekében folyamatosan vizsgálni kell a futó modulok adatait. Ebből következik, hogy először egy olyan részt kellett találni a keretrendszer felépítésében, amelyben a folytonos futás lehetősége, mint alap tulajdonság, biztosított.

Maga a keretrendszer úgy lett megtervezve, hogy legyen egy olyan service is implementálva, ami folyamatosan fut a háttérben és szükség szerint menedzseli a keretrendszerrel kapcsolatos teendőket (például: elérhető pluginok összegyűjtése, szükséges rendszerkomponensek indítása). A létező forrásban ezt a szerepet a CoreService tölti be. Ez a service megfelelő implementálási helyet biztosít az ellenőrző szálnak.

Miután meglett, hogy hol érdemes megvalósítani az ellenőrző komponenst, következhetett a tényleges megvalósítás. Az ellenőrző szál tervezésekor készített folyamatábrát az alábbi ábrán (4.1. ábra) láthatjuk.



4.1. ábra: Az ellenőrző szál folyamatábrája

Mint az a folyamatábráról is leolvasható a tényleges ellenőrzés egy végtelen ciklusban történik, melyet egy while(true) ciklussal valósítok meg. Ez magában eléggé erőforrás igényes folyamat lenne, de az ellenőrzési ciklus végén történő egy másodperces altatás ezt a problémát megoldja és a futás során az ellenőrző szál csak minimális mértékben használja a készülék erőforrásait. Emellett az altatás alkalmazásával meg lett valósítva a már korábban említett periodikus adatlekérdezés is.

A folyamatos évközi konzultációk során felmerült az igény, hogy lehessen azt is szabályozni, hogy a beállított korlát a modulok processzor használatára egyenként vagy összesítve vonatkozzon. Így a tervezett szál folyamatábráját módosítani kellett ennek megfelelően, a módosításokat az alábbi ábrán (4.2. ábra) láthatjuk.



4.2. ábra: Módosított folyamatábra (részlet)

Természetesen a folyamatábrán csak a szabályozás logikai lépései vannak feltűntetve a könnyebb megértés és az átláthatóbb ábra miatt. Néhány olyan adat lekérdezésének feltűntetése kimaradt, amely magához a helyes működéshez szükséges (például a CoreService folyamat azonosítójának lekérdezése). A megvalósított szál az altatás előtt elvégzi az összes futó modullal kapcsolatban a felvázolt ellenőrzést és csak ezután zárja le az aktuális ciklust a szál altatásának meghívásával.

Miután az előzetes tervek alapján sikerült implementálni az ellenőrző szál első változatát, a rendszer készen állt a modulok processzor használatának korlátozására. Az egyetlen problémát az okozta, hogy ebben a verzióban a korlát értéke a kódban volt statikusan meghatározva. Így a következő lépés ennek megváltoztatása volt úgy, hogy futás közben tetszőleges értéket lehessen beállítani.

Maga a keretrendszer jelenlegi felhasználói felülete eléggé egyszerű és letisztult. Nem találtam megfelelő helyet a beállítási lehetőséget implementálásához a meglévő kódban, ezért arra a döntésre jutottam, hogy a létrehozok egy menüt az alkalmazáshoz, és azon keresztül lehet majd a beállításokat végrehajtani. Az Android SDK-ban létezik egy úgynevezett Preference osztály, mely segítségével lehetséges alapszintű beállításokhoz felhasználói felületet egyszerűen létrehozni. Természetesen nem kötelező ezt a lehetőséget használni, saját activity-t is készíthetünk erre a feladatra, de úgy gondoltam, hogy két egyszerű érték beállításához elegendő a fejlesztők által biztosított osztály használata. Rövid tervezés után elkezdtem implementálni a menüt. Ehhez először létre kellett hozni a szükséges PreferenceActivity-ből származtatott új activity-t, ami a beállítások megjelenítéséért felelős (miután kiválasztottuk a megfelelő menüpontot). Ezen felül még szükséges volt egy új xml fájl hozzáadására az erőforrás fájlokhoz, melyben a megjelenítendő elemeket kell definiálni, illetve a már létező fájlok egy részét is ki kellett egészíteni a szükséges új értékekkel. Végezetül a keretrendszer kezdő kezelőfelületéért felelős MainAcitivy-t kellett kiegészíteni, hogy az implementált menü elérhetővé váljon.

Úgy terveztem, hogy modulok vizsgálati módjának beállítását egy jelölőnégyzettel (idegen szóval checkbox) oldom meg. Alapértelmezett értékként az egyéni vizsgálatot állítottam be. A processzor használattal kapcsolatos korlát beállítását egy csúszka (idegen szóval slider) segítségével szerettem volna megvalósítani. Sajnos ez utóbbival akadt egy kis probléma az implementálás során, mint kiderült a Preference használata során nem lehet minden Android-ra jellemző elemet alkalmazni, így a tervezett menüt nem lehet megvalósítani az elképzelt módon. A felmerülő új probléma megoldásának keresése közben találtam egy külsős projektet, ami pont a számomra szükséges hiányosságot orvosolja. A könyvtár szerzője Jay Weisskopf, aki a MIT (Massachusetts Institute of Technology) hallgatója vagy munkatársa, legalább is forráshoz mellékelt engedélyből erre lehet következtetni (az egész könyvtár forrása és a hozzá tartozó engedély a [14] alatt megtalálható). Mivel az adott engedély nem tiltja a talált könyvtár használatát és kimondottan nem a tényleges szabályozási metódust segítené megvalósítani a talált forrás, úgy gondoltam, hogy a szóban forgó könyvtár használata nem rontaná a saját munkám megítélését. Emellett egyszerű és gyors megoldást biztosítana a hirtelen felmerült problémára. Miután sikeresen hozzáadtam a könyvtárat az meglévő saját projektünkhöz, egyéb probléma nélkül sikerült befejezni a processzor használat szabályozásához szükséges felhasználói felület megvalósítását. A végeredményt az alábbi két ábrán (4.3. ábra és 4.4. ábra) láthatjuk.

|  |  |
| --- | --- |
| 4.3. ábra: Beállítások menü | 4.4. ábra: Processzor használat mértékét szabályozó csúszka |

A felhasználói interfész hozzáadásával befejeztem a processzor használatot ellenőrző szál implementálását. Most már létezik az adattároláshoz szükséges struktúra, és az is megvan, hogy hol és milyen módon történik ezeknek a felhasználása, így a keretrendszer szempontjából nincs akadálya új funkció használatának. Egy utolsó dolog viszont még mindezek mellett szükséges, ahhoz hogy tényleg az elvártnak megfelelően működjön a szabályozás. Következzen most ennek ismertetése.

#### A betöltendő modulok kódjának kiegészítése

Mint azt korábban már említettem, sajnos van olyan szabályozáshoz szükséges adat, amit csak az adott szál képes lekérdezni futása alatt. Az ehhez szükséges kódot még a fordítás előtt hozzá kellene tenni a futtatásra szánt modulok forrásához. Emellett arra is szükség van, hogy a már említett altatást megvalósító részletet is hozzáadjuk a modulokhoz. Mivel az egész szabályozó rendszernek az egyik célja, hogy az esetleges kártékony céllal írt modulok sikeres működését meghiúsítsa, így a szükséges kódok beillesztését nem bízhatjuk a modulok fejlesztőinek becsületességére, kell valami automatikus módszer, amivel a kiegészítést elvégezhetjük.

A modulfejlesztőknek viszonylag könnyű dolguk van a fejlesztés folyamatát tekintve. Egyetlen .java kiterjesztésű fájlt kell nekik elkészíteniük és a tervek szerint, majd ezt a fájlt eljuttatva a központi szerverre, a szerver fog belőle a keretrendszer számára futtatható modult generálni. Ez azért fontos a feladat szempontjából, mert a szükséges módosításokat ezen a .java fájlon, vagy az ebből fordított binárison kell elvégezni.

A probléma felismerését követően az első lépés az elvárt funkciók meghatározása volt a tervezés során, a következő kritériumokat szabtuk meg a kiegészítővel kapcsolatban:

* könnyen beépíthető legyen a jelenlegi modul átalakító folyamatba
* automatikusan végezze el a módosításokat
* lehetőleg csak olyan helyre illesszen új kódot, ahol arra nagy valószínűséggel tényleg szükség van

Mivel ez az új, általunk tervezett szabályozási módszerhez köthető probléma, ezért a kezdeti tervezés során ezzel a feladatkörrel nem foglalkoztunk. Így a következő lépés a lehetséges megoldások keresése volt. Felmerült az ötlet, hogy egy saját fordítót kellene megvalósítani. Ezt a lehetőséget elég hamar elvetettük, mivel jobban utánanézve annak, hogy mivel jár pontosan egy ilyen munka, első látásra úgy tűnt, hogy túl nagy feladat lenne a jelenlegi teendők mellé, és arról sem tudtunk megbizonyosodni teljes mértékben, hogy egy saját fordítónak a szerveren történő használata megoldható-e. Emellett időközben két másik lehetséges, az előbbinél egyszerűbb megoldást sikerült találnom.

Az első ezek közül az úgynevezett Eclipse Java fejlesztői eszköz (JDT, Java Development Tools[15]). Ez egy olyan alprojektje az Eclipse-nek, amely kapcsán különböző Java programozási eszközöket fejlesztenek. Maga JDT projekt öt nagyobb különálló projektre bontható szét, számunkra ezek közül a Mag (idegen szóval Core), amelyik releváns. Segítségével az adott .java fájlt elemeire bonthatjuk és egy fa struktúrába rendezhetjük. Ezek után a létrehozott struktúrát módunkban áll megváltoztatni, kiegészíteni.

A másik lehetőség az úgynevezett Javassist. Ennek segítségével teljesen más oldalról közelíthetjük meg a problémát, mivel lényegében ez egy olyan eszköz, amely segítségével futási időben tudunk új osztályokat definiálni vagy módosítani, ha éppen erre lenne szükség. Kétféle API-t (Application Programming Interface, felhasználói program interfész) használhatunk, egy forrás szintűt és egy bájtkód szintűt. Az előbbivel lehetséges az adott .class fájl módosítása a tényleges bájtkód ismerete nélkül, viszont csak a Java nyelvben ismert alapelemekkel dolgozhatunk, az utóbbi segítségével pedig közvetlen módon szerkeszthetjük a .class fájlt, mint a többi hasonló szerkesztőben.

A rövid ismertetők után egy összehasonlító táblázat (4.1. táblázat) segítségével szeretném bemutatni, hogy melyek azok a fő szempontok, amelyek szerint vizsgáltam a két lehetőséget, illetve hogy ezekre a szempontokra milyen megoldást nyújt a JDK és a Javassist.

|  |  |  |
| --- | --- | --- |
| Szempontok | JDK | Javassist |
| Forrásfájl: | .java kiterjesztésű | .class kiterjesztésű |
| Új függvény hozzáadása a forráshoz: | Nehezen megvalósítható. | Könnyen megvalósítható, de korlátozottak a lehetőségek. |
| Forrás létező elemeinek szerkesztése: | A fa struktúra miatt könnyen megvalósítható. | Korlátozottan megvalósítható. |
| Egyszerű elem hozzá-adása adott helyre: | A fa struktúra miatt könnyen megvalósítható. | Megvalósítható, de fennáll a kódfelülírás veszélye. |
| Egyszerű ciklusok detektálása: | Könnyen megoldható a kikeresésük, emellett mó-dosítható a tartalmuk is. | Nincs rá támogatott mód. |
| Változtatások mentése: | Nehezen megvalósítható. | Könnyen megvalósítható, ha a szintakszis ellenőrzés nem talál hibát. |

4.1. táblázat: A JDT és Javassist összehasonlítása

Látható, hogy mindkét módszernek megvan a maga erőssége és gyengesége. A JDK segítségével sokkal pontosabban tudjuk elemezni a kódot (elemi szinten) és a meglévő részeket módosítani, de az új részek hozzáadásával problémák adódnak. Ezzel ellentétben a Javassist az új elemek létrehozásában előnyösebb, ellenben az eredeti részeket bonyolult vele manipulálni. Könnyen észrevehető, hogy az egyik gyengesége a másik erőssége, így adja magát az ötlet, hogy használjuk mindkettőt az adott forrás megfelelő formátumával kapcsolatban. Sajnos ez nem lehetséges, mivel csak adott sorrendben tudjuk használni őket a forrásfájlok miatt (előbb a JDK és utána a Javassist), és a kötött sorrend következtében nem tudjuk az előnyöket kihasználni. Emellett a Javassist használata során nem hivatkozhatnánk a 4.1.1.1-ben ismertetett osztály elemeire, mivel ezek nem részei az alap Java szótárnak és a szintakszis ellenőrzés alatt hibát dobna ezekre a részekre a fordító. Ezért elkezdtem próbálkozni a JDK által nyújtott lehetőségek tesztelésével és az esetleges gyengeségek kiküszöbölésével.

A legfőbb problémát a szükséges két új függvény hozzáadása okozta. A sikeres implementációhoz követni kellett volna a struktúra felépítését és elemenként felépíteni az adott részeket, ami már egy egyszerűbb függvény esetében is komoly feladatnak bizonyult. Be kellett látnom, hogy így megvalósítani az új részeket túl sok időt emésztett volna fel és úgy döntöttem, hogy erre a célra kisegítő megoldást keresek. Először megírtam mindkét függvényt, hogy tisztába legyek vele pontosan mit is kellene hozzáilleszteni a modulok forrásához. Mivel a rendszer készen állt a szabályozás használatára, így egy manuálisan kiegészített modullal leteszteltem a megírt függvényeket. Miután meggyőződtem a helyes működésről tovább folytattam a keresést a hozzáadás problémájának megoldására. Sajnos nem találtam semmilyen használható megoldást, így végül úgy döntöttem, hogy egyszerű sztring műveletek segítségével hozzáfűzöm a forrásfájlhoz a szükséges két függvény implementációját. Ez nem egy túl elegáns megoldás, de a jelen esetben használható, mivel a szóban forgó függvények elég egyszerűek, így nem okoz olyan nagy problémát átalakítani őket a szükséges formátumra, illetve egy helyre megoldható a beillesztésük. Már csak egy olyan fix pont kellett, amely minden modul forrásában közös. Ezt elég hamar sikerült megtalálni, mivel mindegyik modul egy osztályt definiál, így a választásom a záró ’}’ karakterre esett. Most már minden szükséges részlet megvan ahhoz, hogy a kód hozzáadását meg lehessen valósítani. A hozzáadás végző alkalmazás folyamatábrája a következő (4.5. ábra):



4.5. ábra: A hozzáillesztés folyamatábrája

Az ábrán látható ellenőrzés azért szükséges, hogy megakadályozzuk a szükséges kód többszöri beillesztését. Ez ugyanis fordítási hibát okozna, mivel a hozzáadott függvények többszörösen lennének implementálva. Az ellenőrzést úgy végzi a folyamat, hogy megvizsgálja az beszúrandó kód egy részlete segítségével, hogy a beolvasott fájl tartalmazza-e az adott részletet. Ha igen, az nagy valószínűséggel azt jelenti, hogy nem első alkalommal szeretnénk módosítani az adott fájlt. Sajnos ez a megoldás így minimális kijátszási felületet biztosít arra, hogy esetlegesen meglehessen kerülni az ellenőrzéshez szükséges részek hozzáadását, viszont ehhez pontosan kell ismerni az átírást végző program kódját, ellenkező esetben elég kicsi a valószínűsége, hogy pontosan kitalálják ellenőrző részletet.

A két szükséges függvény beillesztése után elhárult minden akadály, ami a JDT használatát meggátolta. A JDT nyújtotta lehetőségeket arra használtam, hogy módosított forrásból kialakított struktúrába beszúrjam az újonnan hozzátett függvények metódushívását. A struktúra elemeinek végigjárásához egy úgynevezett ASTVisitor osztályt kellett példányosítani, és minden egyes elemtípusra, amellyel kapcsolatban módosítást szeretnénk végrehajtani, egy visit(elemtípus) függvényt kellett definiálni. Az adott függvény törzsében határozhatjuk meg, hogy mit csináljon az alkalmazás, ha a struktúrában a paraméternek megfelelő elemet talál. A két beillesztett függvény két teljesen eltérő feladatot valósít meg, így különböző szempontok szerint kellett kiválasztani a beszúrásuk helyét. A szál azonosítójának beállítását végző függvényt minél hamarabb kell meghívni, hogy az ellenőrzés elkezdődhessen. Viszont arra is kell figyelni, hogy ha nem a megfelelő helyen (például a futás megkezdése előtt, az inicializáló függvényben) végezzük el a metódushívást, akkor feleslegesen módosítottuk a forrásfájlt. Végezetül úgy láttam helyesnek, hogy két helyen érdemes meghívni az azonosítót beállító függvényt, a run, illetve az onTimerEvent függvény elején. Az előbbivel kapcsolatban sok magyarázni való nincs, amikor ezt a függvény meghívódik, biztos elindul a modul futása is. Az utóbbi egy, a keretrendszerben definiált függvény, mely segítségével különböző időzítő vezérelt eseményeket tudunk létrehozni a modullal kapcsolatban. Használatával lehetséges olyan modulokat is megvalósítani, amelyek nem rendelkeznek run függvénnyel, ezért szükséges az onTimerEvent esetében is a függvényhívás. Az nem okoz gondot, ha esetleg többször is meghívásra kerül az adott függvény, mivel egy szál azonosítója futás alatt nem fog megváltozni, és a beállítást végző függvény úgy lett megvalósítva, hogy észlelje az újrahívást, és ne változtassa meg az először beállított értéket.

Az altatást megvalósító függvény kapcsán már bonyolultabb volt meghatározni, hogy hol érdemes beszúrni a metódushívást. Az túl pazarló lett volna, ha minden egyes elem után beszúrja az alkalmazás a metódushívást, emellett megtörténhet az az eset is, hogy olyan helyre sikerülne a beszúrást elvégezni, ahol később az futás alatti hibát okozna. A tökéletes megoldás létrehozásához mélyreható kódelemzést kellene megvalósítani, amire a JDT nem alkalmas, ezért végig kellett gondolni, hogy mi az, ami veszélyt jelenthet és a JDT segítségével orvosolható. Arra jutottunk, hogy a legnagyobb problémát a rossz szándékú fejlesztők ciklusok segítségével tudják okozni, így első körben különböző ciklusok törzséhez kellene a metódushívást hozzátenni. Szerencsére a JDT segítségével a ciklusokat könnyen megkereshetjük a struktúrában és viszonylag egyszerűen hozzáadhatjuk a szükséges elemet a ciklus belsejéhez. Emellett úgy láttam helyesnek, hogy az azonosítót lekérdező kód után is érdemes a másik függvényhívást is hozzáadni, mivel az onTimerEvent segítségével lehetséges periodikus eseményeket is definiálni, amit akár egy ciklusként is értelmezhetünk.

Egyetlen dolog maradt hátra a modulok módosításával kapcsolatban, és ez nem más, mint az újonnan módosított változtatások elmentése. Mint azt korábban említettem, ezzel is nehézségek vannak a JDT-vel kapcsolatban. A célra létezik elviekben egy ASTRewrite elnevezésű osztály, de se a hivatalos, se a mások által készített oktatási anyagok segítségével nem sikerült rendesen használni. Szerencsére észrevettem, hogy a JDT által használt CompilationUnit tartalma majdnem egyezik a beolvasott forrásfájl tartalmával, az egyedüli különbség, hogy az előbbiből hiányoznak az üres sorok és a kommentek. Az JDT használata során azt tapasztaltam, hogy változtatások után az új elemek is belekerülnek ebbe az osztályba, így az ASTRewrite-al való próbálkozás helyett, inkább a módosítások elvégzése után kiírattam a CompilationUnit tartalmát egy fájlba, rövidre zárva az utolsó fennálló problémát a processzor szabályozásával kapcsolatban.

Utolsó lépésként a modul átíróval kapcsolatban létrehoztam egy olyan .jar fájt, amely segítségével ellehet végezni a bemutatott függvényhozzáadás és metódushívás kiegészítést egyetlen parancs segítségével. Ehhez csak a le kell futtatnunk az adott .jar fájlt a parancssorban, paraméterül megadva a módosítandó modul forrás és cél elérési útját. Ennek segítségével könnyen ki lehet egészíteni a jelenlegi modul átalakítás folyamatát.

Ezzel befejeztem a processzor használat szabályozásával kapcsolatos elemek megvalósításának bemutatását. Az alfejezet zárásaként következzen egy szekvencia diagram, ami 4.1.1-ben ismertetett rendszerkomponensek kommunikációját szemlélteti.



4.6. ábra: Processzor használat ellenőrzésének szekvenciadiagramja

### Plugin szabályozás

A pluginokkal kapcsolatban elég sok információt megismerhettünk a 2.1.2.1-es és 2.2.2-es fejezeteknek köszönhetően. Sajnos a projekt nem a tervezett ütemben haladt, így korábban ismertetett kvóta rendszer implementálása nem történt meg az eltervezett módon. Ennek ellenére a jelenleg elérhető minta plugin segítségével próbáltam egy olyan szabályozási megoldást létrehozni, ami a későbbiekben könnyen kiegészíthető, ha elkészülnek a keretrendszerhez írt egyéb pluginok is. A megvalósítandó plugin szabályozási feladatokat leszűkítettük az adott plugin metódushívásainak korlátozására. Ezzel biztosítva lett, hogy az elkészült megoldás a többi plugin esetében is alkalmazható lesz.

A munkát most is, hasonlóan a modulkezelés kiegészítésénél, a pluginokhoz kapcsolódó források tanulmányozásával kezdtem. Ennek eredményeként megállapítottam, hogy a metódushívásokat és ezek eredményeit a keretrendszer intentek segítségével közvetíti a különböző komponensek között. Az intentek úgy is példányosíthatóak, hogy az adott példánynak egy úgynevezett akciót adunk paraméterül, amelyet egyfajta azonosítóként használhatunk fel az intenttel kapcsolatban. Ebben az esetben az intent többi paramétere null értékű lesz. A hozzárendelt akció segítségével meg tudjuk szabni, hogy mely broadcastreceiver-ek dolgozzák fel az adott intentet. Ehhez a megfelelő broadcastreceiverek kapcsán az adott akcióra intent-szűrőt kell definiálni. Az intentekhez különböző extra értékeket adhatunk hozzá, a megfelelő szűrővel rendelkező broadcastreceiver ezeket az adatokat ki tudja nyerni az beérkezett intentből.

A projektben két broadcastreceiver van, amely a keretrendszerrel kapcsolatos intenteket feldolgozza. A fő alkalmazás részeként ott van a PluginIntentReceiver, mely az eredmények feldolgozásában játszik szerepet. A másik nem kimondottan a keretrendszer része, hanem egy speciális úgynevezett Android könyvtár projektben van megvalósítva. Ezt a megoldást általában akkor szokták alkalmazni, ha egy adott forrást több másik Androidos alkalmazás kapcsán is fel szeretnének használni. Ennek a lépésnek az egyedüli hátránya, hogy az adott projektet a továbbiakban nem lehet önálló alkalmazásként fordítani. A keretrendszerrel kapcsolatos Android könyvtár projektben egyetlen BroadcastReceiver-ből származtatott osztály van, mely a következő feladatokat látja el:

* a pluginokkal kapcsolatos beérkező intentek feldolgozása
* az extra adatok kinyerése után a megfelelő plugin metódusának meghívása
* válasz intent legenerálása és elküldése

Ezt az osztályt PluginCommunicationInterface-nek nevezik. A különböző metódusok implementálása nem itt történik. Erre minden pluginnak létezik egy saját osztálya, amelyet ebből az osztályból kell származtatni. Az adott plugin megfelelő működéséhez szükséges szűrőket a saját broadcastreceiver osztályának Manifest részletében kell definiálni.

Miután sikerült megérteni a pluginokkal kapcsolatos részek működését, következhetett a szabályozórendszer megtervezése. A kommunikációs eljárás adott volt, így ezzel nem kellett külön foglalkozni. A szabályozás megvalósításának legjobb helyeként a PluginCommunicationInterface mutatkozott, mivel az itt végzett módosítások az összes később megvalósított pluginra is vonatkozni fognak, illetve az implementált metódusok hívása itt történik, ezért a legegyszerűbben itt tudjuk számon tartani a meghívásaik számát.

A tervezést szokásos módon az implementálási fázis követte, mely a következő fontosabb részekre bontható:

1. Értékek beállításához szükséges felhasználói felület kialakítása
2. A beállított értékek kiolvasása és intenté alakítása, intentek elküldése
3. Az adott plugin saját broadreceiverének felkészítése a korlátok fogadására
4. PluginCommunicationInterface kiegészítése a korlátok feldolgozásával és szükséges számlálókkal
5. Ellenőrzés és számlálók karbantartásának megoldása
6. Számlálók meghatározott időközönként történő újraindítása

Érdemes megjegyezni, hogy csak a lista első három eleme kapcsolódik konkrét plugin módosításához, a többi lépést az újabb pluginok implementálása során már nem kell végrehajtani.

Mint az a listából is látszik a tényleges implementálást szükséges felhasználói felület kialakításával kezdtem. Hasonló módon terveztem ennek a beállítási lehetőségnek a megvalósítását, mint ahogy azt processzor használat korlátozásával kapcsolatban tettem, így azt nem szeretném újra leírni, hogy milyen lépéseket kell tenni egy menü implementálásával kapcsolatban. Két fontos különbség van a korábban implementált menühöz képest. Az első, hogy ezt a menüt nem a keretrendszerből lehet elérni, hanem az adott plugin felhasználói felületén keresztül. A második dolog pedig az, hogy a szerkezeti felépítése sem egyezik az előzővel, mivel más funkciót szeretnénk megvalósítani vele. A dolgozat korábbi részében már elhangzott a kvóta rendszerrel kapcsolatban, hogy a felhasználók dolgának megkönnyítése érdekében csoportokat hozunk létre az adott korlátok meghatározására, így nem kell külön-külön értékeket beállítaniuk minden egyes elemmel kapcsolatban. A megvalósítás során négy különböző kategóriát definiáltam:

* szigorú korlát: öt darab függvényhívást enged meg a beállított időszakban
* közepes korlát: ötven darab függvényhívást enged meg a beállított időszakban
* laza korlát: kétszáz darab függvényhívást enged meg a beállított időszakban
* korlátlan: annyi függvényhívás engedélyezett, amennyire szükség van

Szerencsére az új menü megvalósítása során nem merült fel hasonló probléma, mint a csúszka kapcsán, a Preference osztály ListPreference elemét pont az ilyen jellegű menük létrehozásához találták ki. Így a modul összes metódusának definiáltam egy ilyen elemet a xml fájlban. Az egyes elemek azonosítójaként a metódusok nevét adtam meg, ez a későbbiekben az adatok kiolvasása és továbbítása során hasznos döntésnek bizonyult. Mivel nem tehető kötelezővé a felhasználó számára az, hogy egy plugin telepítése után az első dolga az összes új korlát beállítása legyen, ezért az elemek implementációja során alapértelmezett értékként a közepes korlátot definiáltam minden metódussal kapcsolatban. Így a szabályozás még azelőtt megfelelően működhet, mielőtt az adott felhasználó meghatározhatná a saját beállítását. Az elkészült új menüről készített képernyőmentéseket a következő ábrákon (4.7. ábra és 4.8. ábra) láthatjuk.

|  |  |
| --- | --- |
| 4.7. ábra: Plugin menü | 4.8. ábra: Beállítási lehetőségek |

A kezelőfelület kialakítása után következhetett a tényleges szabályozás megvalósítása. Ehhez először ki kellett olvasni a beállított adatokat. A Preference osztály az adatokat erre a célra fenntartott SharedPreferences tárolóba menti el. Az adott SharedPreferences egész tartalmát be lehet olvasni egy Map<String,?> típusú struktúrába. A Map első paramétere az elemek azonosítója, ami jelen esetben megegyezik a metódusok neveivel, a második paramétere pedig a beállított érték. Egy for ciklus során az összes metódusnév-korlát párost kiolvashatjuk ebből a srtuktúrából és a ciklus belsejében el is küldhetjük a megfelelő akcióval és extra adatokkal ellátott intentet. Erre a célra egy saját akciót definiáltam a keretrendszerrel kapcsolatban és hozzáadtam a minta plugin broadcastreceiveréhez a szükséges szűrőt. Ezek után már csak az intentek elküldését megvalósító részlet helyét kellett megtalálni a kódban. Ehhez számításba vettem, hogy melyek azok az események, amelyek után szükség van az intentek elküldésére. Az első ilyen eset a plugin indulásakor van, ezért a plugin saját háttér service-nek tartalmaznia kell a kódot. Ezen felül akkor kell még újraküldeni a korlátokat, ha a felhasználó megváltoztatta őket. Mivel a menü csak a plugin felhasználói felületéről érhető el, ezért az ehhez tartozó activity onResume függvényébe is beillesztettem a szükséges részt. Ezzel az utolsó kiegészítéssel befejeztem a konkrét plugin módosítását és következhetett a PluginCommunicationInterface kibővítése. Először is szükség volt két tárolóra a metódusnév-korlát és a metódusnév-számláló párosok tárolására. Úgy láttam legjobbnak, ha ezeket is SharedPreferences segítségével tárolom. Az adatkonténerek definiálása után létre kellett hozni azt a részletet, amely a saját akcióval paraméterezett intenteket észleli és a kiolvasott extra adatokat beleírja a tárolókba. A korlátok tárolóját minden esetben módosítja a program, de a számlálókét csak akkor, ha adott metódusnak még nem létezik számlálója a tárolóban. Ezzel kész lett a beolvasási fázis is, következő lépésként az ellenőrzés megvalósítása következett. Ehhez azt a részt kellett kiegészíteni, ami a metódushíváshoz tartozó intentet dolgozza fel. Az ilyen intentek extra tartalma tartalmazza annak a metódusnak a nevét, amelyiket szeretné meghívni az intentet kiváltó modul. Az alkalmazás ezt a metódusnevet kulcsként használva először megvizsgálja, hogy megtalálható-e a korlátok tárolójában, ha igen akkor ellenőrzi, hogy nem korlátlan-e a beállítás. Ha nincs benne vagy korlátlan, akkor az eredeti kód további része fut le, mint a módosítás előtt. Ha benne van és nem korlátlan, akkor ellenőrzi, hogy a számláló kisebb-e mint a beállított korlát. Ha igen, újfent az eredeti kód fog lefutni, és a metódushoz tartozó számláló eggyel növekszik. Ha a számláló elérte a limitet, akkor az adott függvény nem hívódik meg, hanem helyette egy hamis eredmény generálódik, ami függvényhívás eredményét helyettesíti, és az alkalmazás ezt használja a válasz intent létrehozásakor. Így a korlátozást a rendszer nem érzékeli, csak a kapott eredmény tartalmán fog a változás látszani.

A plugin metódus hívásokat szabályozó rendszer az eddigi módosítások után majdnem készen áll a működésre, egyetlen dolog hiányzik ahhoz, hogy normálisan használni lehessen. Ez pedig nem más, mint a számlálók újraindítása, hiszen e nélkül rövid időn belül elérnék a számlálók a beállított korlátokat (kivéve, ha korlátlanra van állítva) és csak az áleredményeket tartalmazó válaszok keletkeznének. Hasonlóan az processzor használatot szabályozó szál esetében a számlálókat újraindító megoldásnak is folyamatosan kell futnia. Annak ellenére, hogy elviekben a pluginoknak is van saját háttér service-ük, jobbnak láttam, ha a CoreService felelős ezért is. A megvalósítást az eddigi tapasztalatok alapján egyszerű volt kitalálni, a CoreService-be létrehoztam egy új szálat, ami a beállított időközönként elküld egy olyan akcióval paraméterezett intentet, aminek feldolgozásakor PluginCommunicationInterface minden számláló tárolóban található kulcshoz nulla értéket állít be. Az számláló lenullázási ciklusok idejét a keretrendszerhez tartozó beállítások menüben választhatjuk ki. Négy értéket lehetséges beállítani: másodpercenként, percenként, óránként vagy naponta történjen meg a számlálók lenullázása.

A keretrendszer módosított menüjét és az új beállítási lehetőséget az alábbi képeken (4.9. ábra és 4.10. ábra) láthatjuk.

|  |  |
| --- | --- |
| 4.9. ábra: Kiegészített baállítások menü | 4.10. ábra: Számlálók nullázásának periódus ideje |

A periodikus lenullázás megvalósításával tényleg befejeződött a plugin metódus hívások szabályozásának az implementálása.

A két szabályozó rendszer együttesen eleget tesz azoknak az elvárásoknak, amelyeket az erőforrás kezelő rendszerkomponenssel kapcsolatban megköveteltünk. Az új implementált részeknek köszönhetően a keretrendszer használata sokkal biztonságosabb lett illetve sokkal jobban azt az érzetet kelti a felhasználóban, hogy övé marad a tényleges irányítás a készülékével kapcsolatban.

Mielőtt ténylegesen áttérnénk a másik nagy feladatkörrel kapcsolatos megvalósítás ismertetésére, jöjjön egy rövid kitérő a már korábban említett profilozással kapcsolatban.

### Profilozás

Profilozáson a keretrendszerrel kapcsolatban azt a tevékenységet értjük, amely során egy adott modul futási igényeit mérjük fel. Erre azért van szükség, mert a későbbiekben, mikor az egész projekt élesben működik, a központi szervernek ismernie kell a kiosztásra váró modul igényeit, hogy olyan készülékre juttassa el azt, amely korlátai mellett is képes a modul lefutni.

A modulok profilozását úgy képzeltük el, hogy lesz erre a célra egy dedikált készülék, amin a modulok a profilozásuk alatt lefutnak. A készülék a futás alatt különböző adatokat gyűjt a modullal kapcsolatban és ezeket az adatokat valamilyen könnyen értelmezhető formátumban eltárolja.

A profilozás tervezésekor az első verzióval a következő adatokat terveztük monitorozni a futtatott modullal kapcsolatban:

* összesített ütemezett idő
* maximális processzor használat
* metódus hívások száma

Maga a profilozás nagyban hasonlít a korábban bemutatott erőforrások szabályozásánál használt folyamatokhoz, az egyetlen különbség, hogy itt nem szeretnénk megszabni a vizsgált értékeknek felső határt, csak megfigyeljük őket. A hasonlóságból adódik, hogy a megoldás logikai menete is egyező lesz több helyen.

Úgy gondoltam, hogy a profilozás folyamatát a keretrendszeren belül kellene, megvalósítani, mivel így a pluginok komponenseitől független lenne a megoldás (elképzelhetőek olyan modulok is, amelyek nem használnak pluginokat futásuk során). Ezért a metódus hívások megfigyelésére nem a PluginCommunicationInterface-t használtam a profilozáshoz, hanem a korábban már említett PluginIntentReceiver-t, mely a keretrendszer részét képezi. Ezt a megoldást nem zavarja a metódus hívások szabályozása, mivel a korlát elérése után is generálódik válasz, csak a már említett hamis eredményt tartalmazza, de ez a profilozás szempontjából lényegtelen. Az itt implementált számlálót hasonlóan valósítottam meg, mint a szabályozásnál használtakat. Amikor egy új metódushívás válasz intentet érzékel a broadcastreceiver, akkor a feldolgozás mellett az adott metódushoz tartózó SharedPreferences-ben tárolt értéket eggyel megnöveli a PluginIntentReceiver. Egyéb szükséges kiegészítés nincs a metódus hívások monitorozásával kapcsolatban.

A maradék két megfigyelendő adatot az ellenőrző szál segítségével könnyű megszerezni. Az összesített ütemezési időt alapból minden ciklusban lekérdezi az ellenőrző szál így ezzel különösebb dolgom nem volt. A másik értékkel kapcsolatban is annyit kellett tenni, hogy minden ciklusban össze kellett hasonlítani az kiszámolt használat értéket az addigi maximális értékkel, és ha az új eredmény nagyobb, mint az addigi, akkor ezek után az újat tárolja az erre a célra dedikált változó.

A kívánt adatok méréséhez szükséges részletek implementálása után már csak azt kellett megoldani, hogy ezeket az adatokat egy helyre gyűjtve elérhetővé tegye a keretrendszer. Erre a fájlba kiíratást találtam a legegyszerűbbnek. Ezt a folyamatot is a CoreService-ben szerettem volna megoldani. Ehhez azonban először elkellett juttatni a service-nek a metódus hívásokkal kapcsolatos információt. Sajnos az eddig használt módszer, konkrétan az intenteken keresztül történő adatkommunikáció jelen esetben nem használható, hiszen ehhez a CoreService kapcsán nincs meg a megfelelő feldolgozó rész. Szerencsére a SharedPreferences tárolót létre lehet hozni olyan paraméterekkel, amelynek köszönhetően lehetőség van a létrehozó osztályon kívül olvasni a tartalmát. Így az információ megszerzésével kapcsolatos probléma is megoldódott. Az adatokat a CoreSerivce a telefonban található memóriakártyára menti a /profiling/profileddata.txt fájlba.

Bár a profilozáshoz köthető részek a felhasználóknak szánt végleges verzióban nem lesz benne, ideiglenesen hozzáadtam a projekthez a profilozást megkönnyítő plusz beállítási lehetőséget. A két extra funkció segítségével be- és kikapcsolhatjuk a profilozás folyamatát a tesztelés alatt, illetve újraindíthatjuk a metódus számlálókat, ha az éppen vizsgált modul befejezte a futását.

Bár ez eddig nem lett kihangsúlyozva, a profilozás folyamata során csak egy modulnak szabad egy időben futnia, hogy az elvárt eredményt megkaphassuk. De az előbb említett plusz funkcióknak köszönhetően a modulok egymás utáni profilozása csak minimális kellemetlenséggel jár.

Ezzel a rövid profilozással kapcsolatos résszel zárnám a fejezet erőforrás kezelésével kapcsolatos részét, a fejezet további részében a másik feladatkörrel kapcsolatos implementáció kerül bemutatásra.

## Gamification elemek implementálása

# Tesztelés

Ebben a fejezetben szeretném bemutatni, hogy az általam végzett módosítások helyesen működnek és a tőlük elvárt feladatokat teljesítik. Mint azt már korábban említettem a dolgozat írásakor létező pluginok száma elenyésző, így a tesztelésre használható modulok lehetőségei eléggé korlátozottak. A tesztelés során két fajta modult alkalmaztam, az egyik a meglévő minta plugin metódusait képes használni, a másik pedig intenzív processzor használatot generál. Ezekkel a célirányos feladatokat megvalósító modulokkal úgy érzem, hogy teljes mértékben be tudom mutatni az általam végzett munka helyességét.

A tesztelési esetek bemutatása előtt szeretnék kitérni a tesztelési körülményekre. A fejlesztés és a tesztelés alatt is a saját készülékemet használtam, melynek fontosabb adatait az alábbi táblázatban találhatóak (5.1. táblázat):

|  |  |
| --- | --- |
| Készülék típusa: | Samsung Galaxy S1 (GT-I9000) |
| Telefon állapota: | Rootolt, főzött rommal ellátott |
| Telepített android verzió: | 4.0.4 |
| Processzor magok száma: | 1 |
| Maximális órajel: | 1000 MHz |

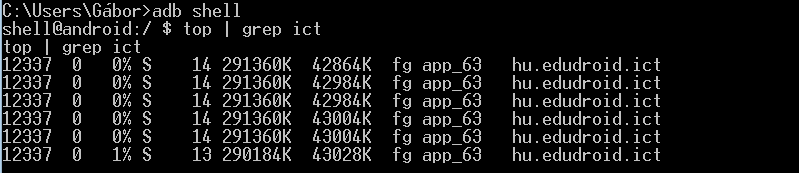
5.1. táblázat: Tesztkészülék fontosabb adatai

Az implementálási sorrendet követve először következzen a processzor használattal kapcsolatos szabályozás működésének bemutatása.

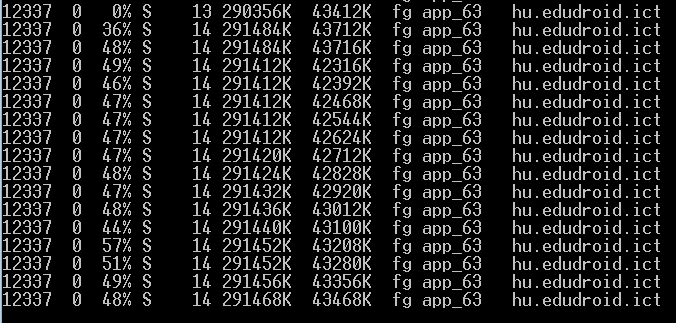
A processzor használatot tesztelő modul magját egy végtelenített while ciklus adja. Arra már korábban is utaltam, hogy ez egy elég erőforrás igényes folyamat. A modul .java fájlján először lefutattam a modul átírót, melynek az eredményét a futtatás előtti illetve a futtatás utáni kimásolt kód mutatja:

|  |  |
| --- | --- |
| public void run(){  while(true){  }  } | public void run(){  setTid();  threadSleeper();  while (true) {  threadSleeper();  }  } |

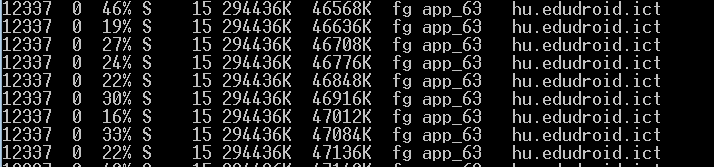
Látszik, hogy az átíró a tőle elvárt módon végezte a dolgát. Ez után már következhetett a keretrendszer által is futtatható modul létrehozása a megszokott módon. Ahhoz, hogy a processzor használattal kapcsolatos összes funkciót ki lehessen próbálni, nem elég csak egy modul. Ezért létrehoztam egy másik, ugyanazt a feladatot végrehajtó modult is, így jelenleg két olyan modul állt rendelkezésemre, amelyek közel maximálisan képesek kihasználni a processzort. Ahhoz, hogy a tesztelés hiteles legyen szerettem volna olyan módon igazolni a processzor használat mértékét, ami független a keretrendszertől. Szerencsére az Android SDK nem csak a fejlesztéshez tartalmaz eszközöket, hanem egyéb tevékenységekhez is segítséget biztosít. Így például az adb eszköz segítségével nyithatunk egy terminált usb csatlakozáson keresztül az adott Androidos készüléken, amin keresztül a Linuxban ismert parancsok segítségével megfigyelhetjük a telefonon futó folyamatokat. Én erre a célra a top parancsot használtam emellett a grep paranccsal szűrést végeztem, hogy csak a keretrendszer folyamatával kapcsolatos adatok jelenjenek meg.



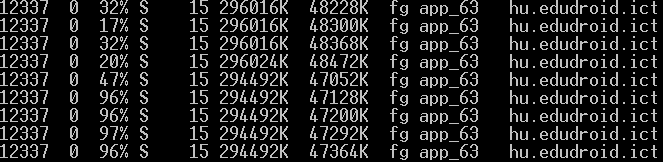
Jól látható, hogy a keretrendszer önálló processzor fogyasztása minimális (az ellenőrző szál is fut már). Ezt követően elindítottam az egyik elérhető modult a keretrendszeren keresztül.



Az volt tapasztalható, hogy a keretrendszer folyamatának processzor használata néhány mérési periódus alatt 50 % körüli értékre felugrott. Azért idáig, mivel a processzor szabályozás alap értékének ez az érték van megadva (a 4.4. ábra is ezt az állapotot mutatja). Egyelőre úgy tűnt, hogy a megfelelő módon működik a szabályozó rendszer. Következő lépésként a beállított korlátott csökkentettem a felhasználói felület segítségével, melynek következő lett az eredménye:



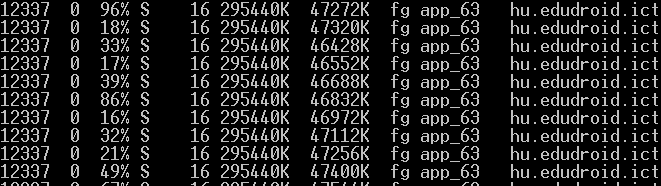
Az új értétek 20 % körül határoztam meg, és a terminálban mért értékek is azt mutatják, hogy a folyamat processzor használata is körülbelül erre az értékre csökkent vissza. Ezt követően a maximális értéket állítottam be a csúszka segítségével és ezután vizsgáltam a mért értékeket.



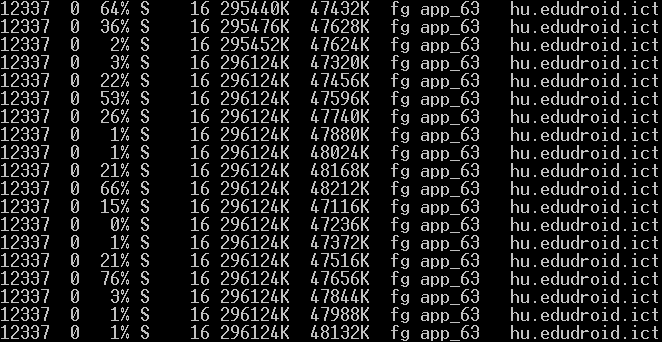
Ennek hatására nagyon rövid idő alatt felugrott az érték 100 % közelébe. Ez igazolja a korábban tett állításomat, miszerint a tesztelésre használt ciklus beavatkozás nélkül tényleg közel maximális processzor terhelést képes generálni. Továbbra is úgy tűnt, hogy megfelelően működik a szabályozó rendszer. Ezek után áttértem annak vizsgálatára, hogy több modul futása alatt is működik-e a szabályozás. Ehhez először visszaállítottam a kezdeti 50 % környékére a korlát értékét, majd elindítottam a második elérhető modult is:



Látható, hogy kezdetben a processzor használat értéke lecsökkent a visszaállított érték környékére, ezt követően a második modul is elkezdett futni és az érték ismételten felszökött a maximum környékére. Ez azzal magyarázható, hogy mindkét modul külön-külön használhatja a beállított értékig a processzor időt. Majd újfenten lejjebb állítottam a korlát értékét, melynek következtében a top által mért adatok is csökkenni kezdtek.



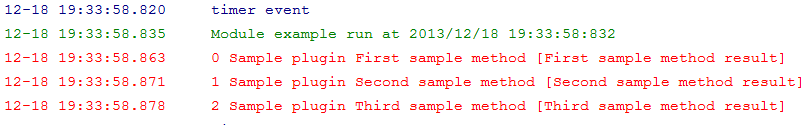
Azt tapasztaltam, hogy az ingadozás sokkal nagyobb, mint az előző esetben. Ez azzal magyarázható egyrészt, hogy a top által mért adatok nem külön a modulokra vonatkoznak, illetve a modul szálak altatása szinkronba kerülhet, és egy időben kezdik újra a futást, aminek hatására hirtelen megugorhat a processzor pillanatnyi használatának mértéke. Az átlagos használatot még így is láthatóan jól korlátozza a megvalósított szabályozás. Az utolsó ezzel kapcsolatos dolog, aminek a helyes működését meg kellett vizsgálni, a közös korlátos funkció működése volt. Ezért a beállításokban kikapcsoltam azt, hogy a korlátot modulonként vizsgálja a keretrendszer.



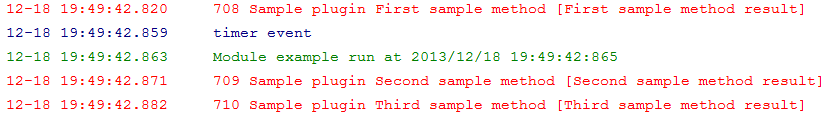
Az eredmény olyan szempontból nem meglepő, hogy az szabályozó rendszer ebben az esetben is végezte a dolgát. Viszont az megállapítható, hogy az összes modul egyszerre történő szabályozásának hatására a keretrendszer processzor használata kevésbé kiegyensúlyozottá válik, lökésszerű terheléseket okoz a processzor használattal kapcsolatban.

Ezzel az utolsó beállítással késznek találtam a processzort szabályozó rendszer tesztelését. Véleményem szerint az elvégzett teszt eredményeit értelmezve nyugodtan kijelenthető, hogy a rendszerkomponens megfelelő módon működik. Ezért át is térek a metódus hívást szabályozó rendszer működésének bemutatására.

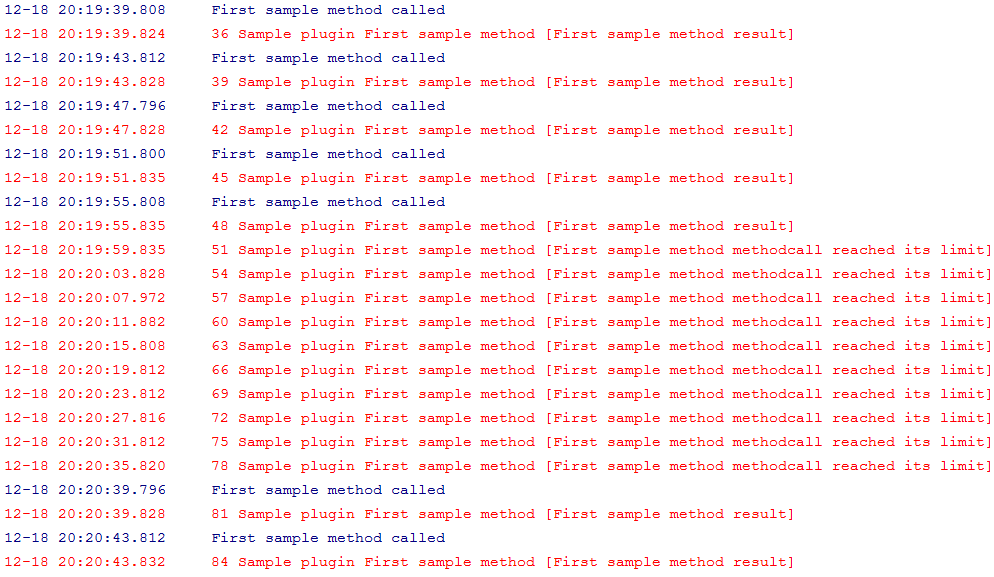
Ehhez először egy új modult kellett betölteni a keretrendszerbe, ami használni képes a minta pluginban definiált metódusokat. Ezek a függvények csak egy egyszerű szrtinget adnak vissza válaszként, melyből meg lehet állapítani, hogy melyik függvényt hívták meg. A maga a modul periodikusan fogja újrahívni a metódusokat. Egy ciklus során mindhárom elérhető metódus meghívódik. A ciklus periódus idejének négy másodpercet állítottam, ami azt jelenti, hogy percenként tizenöt alkalommal fogja a modul meghívni a metódusokat. Ez azt jelenti, hogy az alapértelmezett beállítások mellett (minden korlát közepesre állított és percenként újraindulnak a számlálók) a szabályozásnak nem kellene életbe lépnie.



Ha a számlálók újraindítása nem működne rendesen, akkor a hívások száma négy percen belül elérné a korlátot, így némi ideig futtatva a modult azt állapítottam meg, hogy egyelőre helyesnek bizonyult a metódusokkal kapcsolatos szabályozó rendszer is.



A korlátozás kikényszerítést a modul módosítása nélkül úgy oldottam meg, hogy az egyik metódushívás korlátját átállítottam szigorúra és az újraindítási időt meghagytam egy percen. Az elvárt viselkedés az, hogy öt hívás után a metódus nem kerül megvívásra és a válasznak meg kell változni az üzenetben, de a program futásának zavartalanul folytatódnia kell.



A log adataiból látszik, hogy a megjósolt módon történtek a dolgok. A számlálók sűrűbb újraindításával a korlát elérésekor kapott üzenetek megszűntek és végig a metódus hívás történt meg. A metódus hívással kapcsolatos teszt alapján véleményem szerint sikerült teljes mértékben igazolni a megvalósított szabályozó rendszer helyes működését.

A profilozás tesztelése annyiból állt, hogy mindkét modulra elvégeztem a folyamatot. Mivel a két szabályozó rendszer működése helyesnek bizonyult és profilozás megoldása az ezek során szerzett tapasztalatokra épült, ezért elegendőnek láttam a helyes működés igazolására ezt a módszert. Emellett a két modullal kapcsolatban is meg tudjuk becsülni magunktól a várható eredményt.

A processzor szabályozás során felhasznált modul profilozásának eredménye:

Summa scheduled time: 1816.0

Max CPU usage: 98.28571%

Ahogy az várható volt, a maximális processzor használat majdnem 100 % (profilozás alatt nem érvényesek a korlátok).

A plugint használó modul profilozásának eredménye (kb. 2 perc futtatás után):

Summa scheduled time: 74.0

Max CPU usage: 0.9285714%

First sample method called 32 times.

Second sample method called 32 times.

Third sample method called 32 times.

Mivel tisztában vagyunk vele, hogy percenként tizenöt metódushívás történik metódusonként, így a kapott eredmények helyesnek tűnnek. Az is hihetőnek tűnik, hogy a processzor használat 1 % körüli, hiszen az idő nagy részében altatás alatt van az adott szál a periodikusság miatt. Úgy gondolom, a mért adatok kellőképp bizonyítják a profilozási módszer helyes működését és ezzel lezártnak tekintem az erőforrás szabályozással kapcsolatos részek tesztelését.

# Összegzés

Irodalomjegyzék

1. ORBIT(Open-Access Research Testbed for Next-Generation Wireless Networks), URL: <http://orbit-lab.org/> (2013. május)
2. PlanetLab, URL: <http://www.planet-lab.org/> (2013. május)
3. SETI@home URL:, <http://setiathome.berkeley.edu/> (2013. május)
4. setrlimit, URL: <http://docs.oracle.com/cd/E19082-01/819-2241/setrlimit-2/>   
   (2013. május)
5. cpulimit, URL: <http://cpulimit.sourceforge.net/> (2013. május)
6. quota, URL: <http://unixhelp.ed.ac.uk/CGI/man-cgi?quota+1> (2013. május)
7. grsecutrity, URL: <http://grsecurity.net/index.php> (2013. május)
8. cgroups, URL: <https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt>   
   (2013. május)
9. PAM (Pluggable Authentication Modules), URL: <http://www.linux-pam.org/>   
   (2013. május)
10. S. Deterding, D. Dixon, R. Khaled és L. Nacke, „From Game Design Elements to Gamefulness: Defining "Gamification"” 2011.
11. Joshua Williams, Ross Smith, Score One for Quality!: Using Games to Improve Product Quality 2008
12. R. A. Bartle, „Hearts, clubs, diamonds, spades: players who suit MUDs” 28 augusztus 1996. [Online]. URL: <http://www.mud.co.uk/richard/hcds.htm>.  
    (2013. december)
13. Linux Programmer's Manual: PROC  
    URL: <http://man7.org/linux/man-pages/man5/proc.5.html> (2013. december)
14. Jay Weisskopf: AndroidSliderPreference projekt  
    URL: <https://github.com/jayschwa/AndroidSliderPreference> (2013. december)
15. Eclipse JDT, URL: <http://www.eclipse.org/jdt/> (2013. december)
16. Javassist, URL: [www.javassist.org/‎](http://www.javassist.org/‎) (2013. december)

1. https://developers.google.com/appengine/ [↑](#footnote-ref-1)
2. https://play.google.com/store [↑](#footnote-ref-2)