Debreceni Egyetem
Informatikai Kar

Alkalmazásfejlesztés Google Androidra

Témavezető
Dr. Fazekas Gábor
Ph.D., Egyetemi Docens

Készítette
Barabás Botond Imre
Programtervező Informatikus (BSc)

Debrecen
2011

A dolgozat írója hálás mindazoknak, akik miatt ott tarthat, ahol...

Köszönöm!

Tartalomjegyzék

Bevezetés

Valamikor a kétezres évek elején elindult az informatikában egy teljesen új irányzat. A Sidekick[1] és a Windows Mobile különböző verziói valószínűleg az alkotóik tudta nélkül elindították a mobilplatform forradalmát.

Manapság szinte mindenkinek mobiltelefonja van és az életünknek elválaszthatatlan része a mobilkommunikáció. Fanyar humorral közölte például a nyugat európai illetve az amerikai sajtó az egyiptomi majd líbiai felkelőkről készült képeket, amint mobiltelefonon beszéltek miközben másik kezükben fegyvereket tartottak.

Nagyon sok mobilfelhasználó érzi azt, hogy egy átlagos mobiltelefon képességei nem elégítik ki szükségleteit. Ezt a vágyat lovagolta meg az Apple az első iPhone-al, ami volt olyan okos és hasznos, mint a régi Windows Mobile alapú eszközök, ám mégis stílusosabb, mint a régi igazán méretes készülékek.

A mindennapi munkám során nem találkozom a mobilplatformmal, ezért kíváncsiságomat a szabadidőmben elégítem ki. Ismerem közelről a BlackBerry, az iPhone, és az Android világát, valamint az Microsoft új Windows Phone nevű platformját is.

A Google viszonylag új szereplőnek számít a mobil piacon, hiszen a Microsoft és a BlackBerry már lényegesen korábban foglalkozott a mobiltelefonok/PDAk operációs rendszerének fejlesztésével.

Mégis az ő platformjukra esett a választásom, hiszen ennyire gyors és ennyire nyitott fejlődést, a többi cég nem nyújt. Példálózhatnék azzal, hogy az Apple átláthatatlan szabályrendszerrel szűri az alkalmazásokat, vagy azzal, hogy a BlackBerry ma már kicsit öregecske és hiába kapkod levegő után, ha a többi igazán nagy platform is megveti a lábát az Enterprise szegmensben, akkor lassan feledésbe merül, vagy akár a Microsoft erősen vérszegény[2] próbálkozásával, amit Windows Phone-nak neveztek el.

Ám ezek helyett inkább bemutatom a platformot minél több aspektusból és az olvasóra bízom, hogy döntse el, megéri-e a legdinamikusabban növekedő és a felhasználóit maximálisan kiszolgáló platformba fektetni úgy időt, mint energiát.

[1] Amerikában a kilencvenes évek végén nagyon népszerű JavaME alapú eszköz

[2] a gyenge készülék eladások és a fejlesztők motiválatlansága miatt merem vérszegénynek nevezni a Windows Phone-t

A dolgozat végigvezeti az olvasót az Android történetén, a platform felépítésén, bemutatja a Dalvik virtuális gép tulajdonságait éppúgy, mint az alkalmazások építőköveit, életciklusának állomásait, valamint az SDK által nyújtott eszközök működését. Végül egy példaalkalmazást tekint át, felhívva a figyelmet a rendszer különböző eszközeire.

1. Az Android története

Andy Rubin, aki korábban az Apple Inc.-nek dolgozott majd később egy WebTV nevű projecten kezdett dolgozni, amit a Microsoft felvásárolt, 2000-ben megalapította a Danger Inc.-et ami legsikeresebb projectjeként a Danger OS nevű mobil platformra optimalizált operációs rendszert könyvelheti el. A Danger OS fut az Amerikában korábban nagyon népszerű Sidekick eszközökön, amiket a Windows Mobil-t futtató eszközökkel az okostelefon kategória megteremtőjének nevezhetünk. Ezekben az időkben a Windows mobile alapú eszközök még managerek zsebében/táskájában található PDA[3]-k voltak, a Sidekick pedig fiatalok kézben tartott chatelésre és sms-ezésre használt eszköz volt. A Sidekick operációs rendszere Java alapú volt, először a J2SE majd később a Java ME platformokon írt alkalmazásokat tudta futtatni.

Rubin 2003 októberében kilép az egyébként a Microsoft által szintén felvásárolt Danger Inc.-ből és megalapítja új cégét azzal a céllal, hogy "...okosabb mobil eszközöket, melyek többet tudnak a tulajdonosuk hollétéről és preferenciáiról..."[4] tervezhessen. Andy Rubinról és a Danger Inc.-ről többet olvashatunk a [1]ben és a [2]ben.

2005 augusztusáig a cég titokban dolgozott, mindössze annyit lehetet tudni, hogy mobil készülékekre optimalizált szoftveren dolgoznak. Augusztusban azonban a Google - ami ekkor már meghatározó cég az információ technológiában - megvásárolja a céget és azt is sikerül elérnie, hogy az összes fontos munkatársat megtartsa (Andy Rubin mai napig a Google-nek dolgozik, ma VP of Engineering beosztásban).

Rubin a Google-nél egy a Linux kernelen alapuló mobil operációs rendszeren dologzott csapata élén. A Google kihasználva a piaci erejét készülékgyártóknak és szolgáltatóknak reklámozta a készülő platformot azzal az ígérettel, hogy a szoftver frissítéseket gyorsan és gördülékenyen érkeznek majd a platformra.

2007 november 5.-én a Google az iparág számos fontos szereplőjével együtt megalapította az Open Handset Alliance-ot vagyis a nyílt telefon szövetséget, tagjai mobil telefon szolgáltatók (T-Mobile, Vodafone, NTT-Docomo), szoftvercégek (eBay,

[3]Personal Digital Assistant

[4]http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm

Google, Pocket Video, SVOX), kereskedelmi konzorciumok (Aplix), félvezető- (Qualcomm, Marvell, Intel, Nvidia) illetve készülék gyártók (HTC, LG, Motorola). Az OHA egyetlen, ám annál fontosabb terméke az Android, amit alapításakor jelentett be, ez egy teljesen nyílt forráskódú Linux alapú operációs rendszer, amit kifejezetten mobil eszközökre szántak. Az OHA tagjairól további információk találhatóak a [3]-on.

Szeptember 23.-án bemutatta az Android 1.0 verzióját, ám ekkor ez még nem egy telefonon, hanem inkább egy prototípus nyomtatott áramkörön futott[5].

Alig egy hónappal később a T-Mobile kiadta a G1 (Google 1 vagy HTC Dream) nevű telefonját, ez volt az első készülék, amin az Android futott. Érdekessége, hogy ennek fizikai billentyűi voltak, mert a rendszer e verziója nem támogatta az érintőképernyőn megjelenő virtuális billentyűzetet.

Fél évvel később megjelent az 1.5-ös verzió, melynek kódneve Cupcake (muffin) volt, az android verzióit innentől kezdve sütemények ábécé szerint növekvő betűivel jelölik.

Ennek a verziónak az újdonságai között szerepel a videó felvétel készítése, Bluetooth támogatás, youtube és picasa elérése illetve a GUI animációkkal való ellátása valamint a virtuális billentyűzetek támogatása.

Ismét fél év telik el, amikor az 1.6 vagyis Donut (fánk) megjelenik. CDMA hálózatok, nagyobb felbontású kijelzők és érintéses gesztusvezérlés támogatást hozott, valamint ezzel vált elérhetővé a Google ingyenes navigációs alkalmazása.

Egy hónappal később a 2.0, majd még egy hónappal később a 2.0.1, majd még egy hónappal később a 2.1 vagy Eclair jelenik meg. Nagyon sok hibajavítást tartalmaz, újdonságként megjelenik a mozgó háttér.

2010 májusában jelenik meg a Froyo (Frozen Yoghurt, fagyasztott joghurt) vagyis a 2.2. Fő újdonsága a JIT fordító, a Flash lejátszó, és a wifi hotspot funkció, valamint az alkalmazások vagy azok egyes részeinek a memória kártyára való telepíthetősége.

[5]http://www.2dayblog.com/images/2008/september/google_android.jpg

2010. december 6.-án jelenti be az OHA 2.3-as verziójú, Gingerbread fantázianevű verziót. Jelentős újításai a SIP VoIP, a nagyméretű kijelzők, több kamera, giroszkóp, barométer és az NFC technológia (Near Filed Communication) natív támogatása.
2011 áprilisának végén jelentette be az Open Handset Alliance a jelenleg legújabb verziót a 2.3.4-et, ami lényegében csak egy újdonságot hozott el: a GTalkba épített hang- és videóchat funkciót, ami nem titkoltan az Apple Face Time nevű hasonló szolgáltatása miatt került bele a rendszerbe.
3.0-ás verzióval, Honeycomb néven jelentettek meg egy speciális tabletekre vagyis tábla PC-kre optimalizált változatott, ami teljesen új grafikus megjelenést, kifejezetten nagy érintőképernyőkre optimalizált billentyűzetet, megújult Gmail és böngésző alkalmazást valamint a Fragment nevű SDK-be épített eszközt hozott az Android világba. A fragment nem más, mint egy olyan eszköz, ami több Activity-t tud összefogni és egyszerre megjeleníteni/kezelni. Fontos tulajdonsága a Fragmentnek, hogy visszafelé kompatibilis az összes többi Android verzióval.
Mivel a 3.0 újdonságait szeretnék a telefonjaikon is viszontlátni a felhasználók, ezért a Google, mint az OHA vezetője bejelentette, hogy az Android következő verziójában (aminek a száma nem, de a neve már ismert), az Ice Cream Sandwich-ben újra összefésüli a tabletekre és a telefonokra kiadott verziókat. Egyes iparági pletykák szerint valamikor 2011 közepén vagy végén várható a bejelentés, nagy eséllyel a 2011-es Google I/O-val egy időben.
Az Android különböző verzióinak újdonságairól még többet olvashatunk [4]-ben.
Abból, hogy milyen sok újdonságot mennyire rövid idő alatt építettek a mérnökök az Androidba látszik, hogy igazán komolyan gondolják a platform fejlődését és többek között ez az oka annak, hogy ma már a legelterjedtebb mobiltelefon platform az Android.

2. Az Android felépítése

Az Android mint operációs rendszer a Linux kernelre épül. A 2.3.3-as verzió, közkeletűbb nevén a Gingerbread például a 2.6.29-es verziószámú kernelt használja. Ez tartalmazza a különböző perifériák meghajtó programjait is, egyebek mellett a hang, videó, wifi, bluetooth, usb és flash memória vezérlőjét, valamint a kamera vagy kamerák és minden egyéb illesztő programjait.

Kernel szinten bele van építve a rendszerbe az energiaellátás vezérlése, mivel az Androidot már az alapoktól úgy építették fel, hogy szem előtt tartották azt, hogy az eszközök nagy része szinte mindig akkumulátorról üzemel. Ennél fogva nincs szükség magasabb szinten működő energiavezérlőre, ami hosszú távon lassabb működést eredményezne.

Egy, ennél magasabb szinten foglalnak helyett a különböző programkönyvtárak, melyeket a hatékonyság kedvéjért főleg C/C++ nyelveken írták a Google mérnökei. Itt találjuk a Surface Manager-t, ami a grafikus felületért felel, ez jeleníti meg a felhasználók számára látható View-kat.

A könyvtárak között találjuk a Media Framework nevű keretrendszert, ami a különböző multimédiás anyagok megjelenítéséért felel. Ide tartoznak a képek (png, bmp, jpg), a hangok (mp3, wav) illetve mozgóképek (h.264 alapú MPEG illetve az újabb rendszerekben a WebM van támogatva). Az Androidon mai napig nem támogatja a DivX/XviD alapon kódolt videók lejátszását ezen a szinten, ennek licence-elési akadályai vannak, ugyanakkor bizonyos készülékgyártók beépítik ennek a népszerű formátumnak a támogatását is.

A könyvtárak között találjuk az SQLite-ot, ami a kisebb teljesítményű rendszerek manapság egyik legelterjedtebb adatbázis kezelő rendszere. Ennek segítségével bármely alkalmazás saját adatbázist hozhat létre, ami nagyban felgyorsítja az adatelérést a felhasználó oldalán.

Az OpenGL a mai világ de facto grafikus szabványa, gyakorlatilag minden olyan eszköz, ami képes képek megjelenítésére, támogatja ezt a szabványt. Az Androidon is fontos ennek a támogatása, mivel a nagy konkurens iOS is támogatja, és az azon méltán népszerű játékok javarészt erre a szabványra építenek. Manapság, amikor az rendszer

lassan betölti a harmadik évét már szinte naponta jelennek meg rá komoly grafikát felvonultató játékok és programok, hiszen akár egy háromdimenziós navigációs alkalmazás is az OpenGL-t használja.

Fontos megemlíteni a WebKit-et, ami egy nyílt forrású weblap megjelenítő motor, asztali környezetben erre épül a Google Chrome vagy az Apple Safari. Lényeges tulajdonsága, hogy törekszik a webes szabványok követésére, így az Android teljes egészében kompatibilis a HTML5 által nyújtott új lehetőségekkel. Egy, a rendszeren futó alkalmazás használhat olyan View-t a grafikus felületén, amely egy WebKit motor által megjelenített tartalmat mutat a felhasználónak, lényegében a rendszer beépített böngészője is így épül fel.

Az Androidos alkalmazásokat futtató rendszer két részből áll, a Dalvik virtuális gépből, ami olyan összetett és egyben érdekes, hogy egy külön fejezetben foglalkozom vele illetve az ezek által használt könyvtárak szűk csoportjából, amiket a Dalvikon futó alkalmazások használhatnak ki.

A már Java-ban írt és a Dalvikon futó része a rendszernek az alkalmazási keretrendszer. Itt található az Activity Manager. Az Activity az alkalmazásnak egy futtatható része, ahogy egy asztali alkalmazásban is lehet több belépési pont (mondjuk több .exe), úgy egy Androidos apk-ban is több belépési pontot definiálhatunk, az ezek által elindított programrészt nevezzük Activity-nek, az Activity Manager ezeknek a futtatását vezérli.

A Window Manager a rendszer azon része, mely az alkalmazások látható részével foglalkozik, ez mondja meg az Activity Managerrel közösen, hogy melyik alkalmazás milyen életciklusban van éppen és ezek vezérlik közösen a ciklusok közötti váltást. Ugyanakkor a Window Manager felel azért, hogy amikor megnyomja a felhasználó a vissza gombot, akkor a legutóbb használt ablak jelenjen meg, ezzel akár több alkalmazáson átívelő vissza-lánc, vagyis ablakváltás is kezelhetővé válik.

A Content Provider az eszközön tárolt standard adatok elérését biztosítja, például a tárcsázó alkalmazás számára a telefonkönyvben lévő névjegyek listáját szolgáltatja.

A View System a grafikus felületek könyvtára, itt van az összes olyan standard osztály, amik a View-ból származnak. A View is egy összetett téma, erről is egy külön fejezetben értekezem.
A Notification Manager vezérli az Android nagy újdonságát, a Notification Bar-t, vagyis az értesítési sávot, ami a legtöbbször a kezelő felület felső részén helyezkedik el. Ezen a rendszerkomponensen keresztül hozhat létre, vagy éppen törölhet egy értesítést egy alkalmazás. Ezek az értesítések adnak lehetőséget arra, hogy egy a háttérben futó alkalmazás üzenhessen a felhasználónak.
A Package Manager az apk csomagok kezeléséért felel, ez adja az eszközkészletet az új csomagok telepítéséhez illetve a már feltelepítettek eltávolításához, valamint a telepítés utáni optimalizációt is ez végzi.
A Telephony Manager a nevéből nyilvánvaló funkciókat végzi, ezen keresztül használható a GSM/CDMA/3G/4G hálózat hang illetve sms továbbításra.
A Resource Manager segítségével erőforrásokat érhetünk el, például fájlokat vagy hálózati erőforrásokat érhetünk el.
A Location Manager a helymeghatározó szolgáltatásokat burkolja be. Egy átlagos Androidos rendszeren elérhető a GPS alapú kifejezetten pontos helymeghatározás valamint a mobiltelefonok átjátszó tornyainak és az ismert helyzetű WiFi hálózatok segítségével egy kevésbé pontos ám még fedett területen vagy magas épületek mellett is elég jó közelítéssel helyez el minket a rendszer a világban. Egy alkalmazás ami szeretne helymeghatározást használni egy Criteria objektumot kell átadjon ennek a rendszernek, amiben megmondhatjuk, hogy mennyi idő illetve mekkora megtett távolság után szeretnénk, ha a rendszer jelezné nekünk az aktuális hollétünket. Az Androidon egyébként sok szolgáltatás működik így, vagyis push alapon. Ez is az energiatakarékosságot erősíti.
Ha belegondolunk egy asztali alkalmazás implementációja jellemzően futtat egy ciklust, amiben minden iterációban megnézi, éppen milyen pozíciót mutat a gps szenzor. Azt könnyen beláthatjuk, hogy ez az átlag felhasználó számára felesleges, mert így akár másodpercenként több tíz vagy több száz olvasás is történhet (egyébként az Android is képes erre, de általában ez egy rosszul megtervezett

alkalmazásra utal), ami felesleges ha mindennapos sebességgel haladunk. Egy másik, picivel jobb megvalósítás az, ha a fenti alkalmazás minden egyes ciklusban olvassa a koordinátákat, majd egy bizonyos időre elalszik. Ekkor az a probléma lép fel, hiszen mi van akkor, ha közben igazán nagy sebességgel sikerült helyet változtatnunk, ekkor a rendszer tévesen azt feltételezi, hogy a korábbi sebességünkkel haladunk a korábbi irányunkba, így ez is egy egyértelműen rosszul tervezett alkalmazás. Az Androidos Location Manager úgy kerüli el a fenti hibákat, hogy a már említett Criteria objektum példányosításakor megadhatjuk, mekkora elmozdulásonként, illetve milyen időközönként jelezze nekünk az aktuális pozíciót. Az alkalmazásunkban kell definiálnunk metódusokat, amiket meghív a Location Manager ha a Criteria-ban megadott elmozdulás vagy idő megtörtént, ezekben a metódusokban dolgozhatjuk fel az új helyzetünket.

Még utolsóként néhány szó a GTalk Service nevű könyvtárról, ami a Google XMPP alapú csevegő szolgáltatásának az Androidos kliense. Ez már a legkorábbi verziókban is szerepelt, ebből is látszik, a Google igazán komolyan gondolja, hogy a többi szolgáltatását az Androiddal integrálja. Ugyanakkor ez a rendszer komponens segíthet eszközök közötti peer-to-peer kommunikációs csatorna kiépítésében, persze csak megfelelő hitelesítés és engedélyek birtokában.

Mivel az Android egy teljes értékű mobil operációs rendszer, ezért a rendszer szintű komponenseken és a middle-wareen kívül tartalmaz felhasználói szintű alkalmazásokat is. Ezekről is írok még pár szót.

Az első, amivel a felhasználó találkozik, az a Home nevű, ami nem más, mint a nyitóképernyőt illetve képernyőket kezelő alkalmazás. Ez úgynevezett widgeteket vagyis mini alkalmazásokat tartalmazhat illetve alkalmazások ikonjait, valamint mappákat, amikbe további ikonokat tehetünk.

Az Android tartalmaz egy alap funkciókkal rendelkező böngészőt, ami szintén a webkit-re épül és HTML5 kompatibilis.

Mivel a legtöbb Androidos készülék még mindig úgynevezett okostelefon, ezért készült a platformhoz egy tárcsázó illetve névjegyeket kezelő és SMS író alkalmazás is, ami a Google fiókban lévő névjegyekkel push módszert használva szinkronizál.

Az Androidban mint minden mást ezeket is lecserélhetjük, létezik főleg a hírekre fókuszáló “home” alkalmazás, létezik a régi számtárcsás telefonok hangulatát idéző tárcsázó illetve több fület kezelni tudó böngésző is.

2.1 A DalvikVM

Korábban már említettem, hogy az Androidra fejlesztett alkalmazások javarészt Java nyelven íródnak, azonban arról még nem volt szó, hogy az Androidos eszközökön nem fut JVM. Helyette egy a Google által kifejlesztett Dalvik VM nevű virtuális gép fut az eszközön.

A Dalvikot Dan Bornstein vezetésével fejlesztette ki a Google. Megemlíteném, hogy Dan nem kezdő ezen a területen, egyebek mellett a PHP-t akár 50%-al is felgyorsító Hip Hop VM is az ő nevéhez fűződik, ennek köszönheti a Facebook, hogy a webserverei lényegesen nagyobb terhelést bírnak, mintha sima php-t használnának. A fejlesztés folyamán végig szem előtt tartották, hogy az átlagos Androidos eszköz viszonylag lassú processzorral és kevés memóriával rendelkezik illetve, hogy ezt swap fájl vagy partíció sem egészíti ki, valamint azt, hogy nagyrészt akkumulátorról üzemel egy ilyen eszköz.

A fejlesztők szeme előtt egy olyan elméleti rendszer lebeget, melynek mindössze 64 megabyte RAM memóriája van és nincs hozzáférése semmilyen swap partícióhoz. Ennek tudatában sikerült az Androidot úgy felépíteni, hogy amikor a legalacsonyabb szintű rendszeralkalmazásokat betölti az Android, akkor még mindig 40 megabyte áll rendelkezésre, majd mikor a legmagasabb szintű alkalmazások is betöltődtek, akkor még mindig 20 megabyte áll rendelkezésre, amit az alkalmazások felhasználhatnak. Az architektúrának köszönhetően ez elégséges is.

A Dalvik virtuális gép egy izlandi halászfaluról kapta a nevét, ahonnan Bornstein ősei származnak.

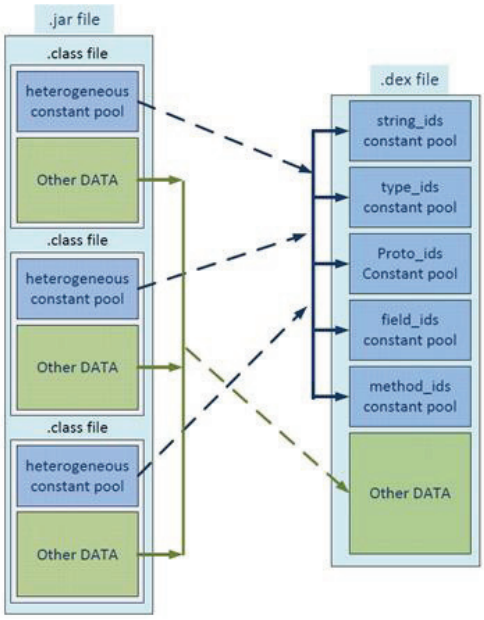
Szemben a Sun/Oracle Java virtuális géppel, amely verem-gép a Dalvik regiszter-gép, mert ezt az architektúrát jobbnak ítélték meg a fent leírt kisebb teljesítményű eszközök esetében. Ahhoz, hogy a Java nyelven írt alkalmazást futtatni tudjon a Dalvik, át kell alakítani olyan formátumba, amit a teljesen más architektúrájú virtuális

gép futtatni tud. Ekkor a dx nevű eszköz, ami szintén az Android SDK része, egy .jar fájlt kap bemenetként, és ebből egy .dex (Dalvik EXecutable) fájlt generál.

2.1.1 A .dex fájl felépítése

A .dex fájlt a .jarba csomagolt .class fájlokat feldolgozva hozza létre a dx.

Nézzük meg hogyan:



1. ábra: konvertálás .dex formátumba[6]

Míg egy .classban a lefordított osztályhoz tartozó konstansok egy heterogén pool-ban vannak elhelyezve, és az egy .jaron belül lévő .classokban valószínűleg elég sok ismétlődés található, addig a .dex az összes osztályt egyetlen fájlba helyezi át, ahol 5 különböző tárolót tart fent:

1. az osztályok kódjaiban szereplő string-ek tárolására
2. osztály prototípusok tárolására
3. mezők tárolására
4. metódusok tárolására
5. az osztályok definíciós részének tárolására

[6] http://imsciences.edu.pk/serg/wp-content/uploads/2010/10/1.1.png

Nézzük meg közelebbről egy .dex fájl felépítését. A dex fájlok formátuma nem a google által publikált anyagokból ismert, egy Michael Pavone nevű informatikusnak sikerült visszafejteni a fájl formátumát, később az ő honlapjáról lekerült az információ, és a http://www.dalvikvm.com/ oldalon jelent meg újra. Én is ebből az információból építkezve mutatom be a fájl szerkezetét.

Egy táblázatban mutatom be a .dex fájl fejlécét.

1. táblázat: a .dex fájl fejléce

offset	méret (byte)	leírás
0x0	8	'Varázs' érték: "dex\n009\0"
0x8	4	Checksum
0xC	20	SHA-1 aláírás
0x20	4	a fájl hossza byteban
0x24	4	a fejléc hossza byteban (jelenleg minden esetben 0x5C)
0x28	8	üres rész, talán későbbi használatra fenntartva
0x30	4	string literálok száma a string táblában
0x34	4	a string tábla abszolút helye a fájlon belül
0x38	4	funkciója nem ismert
0x3C	4	osztályok száma az osztály listában
0x40	4	az osztály lista abszolút helye a fájlon belül
0x44	4	a mezők száma a mező táblában
0x48	4	a mezőtábla abszolút helye a fájlon belül
0x4C	4	metódusok száma a metódus táblában
0x50	4	a metódus tábla abszolút helye a fájlon belül
0x54	4	osztály definíciók száma az osztálydefiníciós táblában
0x58	4	az osztálydefiníciós tábla abszolút helye a fájlon belül

Az összes nem string típusú adat little-endian formában van tárolva. A fájl checksum-jának és aláírásának generálása közben a checksum és az aláírás mezők 0 értékkel szerepelnek.

A fájl további részei a fejléc által hivatkozott string tábla, osztály lista, mező tábla, metódus tábla, osztály definíciós tábla, valamint egy mező lista, metódus lista, egy kód fejléc és egy lokális változó lista. A következőkben ezekről kicsit bővebben fogok beszélni.

A string táblában található a .class fájlokból kinyert szöveges literálok. Ezek közé tartoznak a szöveges konstansok, az osztály-, változó és egyéb elnevezések és sok miden más. A táblázat mindenegyes sora a következő formátumú:

2. táblázat: a string tábla formátuma

offset	méret (byte)	leírás
0x0	4	a szöveges literál abszolút helye a fájlban
0x4	4	a szöveg hossza

Bár a táblázat tartalmazza a szöveg hosszát is, de a C stílusú nyelvekhez hasonlóan a szöveg a ’\0’ karakterrel van lezárva

Az osztálylista a .dex fájlba bemozgatott vagy benne hivatkozott osztályok listája. Mindenegyes bejegyzés a következő formátumú:

3. táblázat: az osztálylista egy eleme

offset	méret	leírás
0x0	4	az osztály nevének megfelelő string index-e a string táblából

A mező tábla a java osztályokban definiált összes mezőt (field-et) tartalmazza a következő formátumban:

4. táblázat: a mezőlista egy eleme

offset	méret	leírás
0x0	4	a mezőt tartalmazó osztály indexe az osztály listából
0x4	4	a mező nevét tartalmazó string indexe a string táblából
0x8	4	a mező típusának leíróját tartalmazó string indexe a string táblából

A metódus tábla a java osztályokban található metódusokról tartalmaz információt a következő formátumban:

5. táblázat: a metódustábla sorai

offset	méret	leírás
0x0	4	a metódust tartalmazó osztály indexe az osztálylistában
0x4	4	a metódus nevének indexe a stringtáblából
0x8	4	a metódus típusleírásának megfelelő string indexe

Az osztályok definícióját leíró táblában miden olyan osztályhoz van információ, amit a dexbe átmozgattunk, vagy aminek egy mezőjére vagy metódusára hivatkozunk valamelyik átmozgatott osztályból. A formátuma a következő:

6. táblázat: az osztálytábla

offset	méret	leírás
0x0	4	az osztály indexe
0x4	4	elérési flagek (public/protected/private)
0x8	4	az ősosztály indexe
0Xc	4	az interfészek listájának elérhetősége a fájlban
0x10	4	a statikus mezők listájának elérhetősége a fájlban

0x14	4	a többi mező listájának elérhetősége a fájlban
0x18	4	a direkt metódusok elérhetősége a fájlban
0x1C	4	a virtuális metódusok elérhetősége a fájlban

A mező lista az osztályokban található mezők inicializációját írja le osztályonként egy 32 bites integer-rel kezdve, ami a mezők számát tartalmazza, majd az összes mezőt a következő formátumban.

7. táblázat: a mezőlista egy sora

offset	méret	leírás
0x0	8	a string indexe vagy az objektum konstans indexe vagy egy primitív literál

Ha egy mező nincs inicializálva, akkor a 0 primitív érték, illetve egy -1 értékű mutató áll a listában.

A metódus lista egy adott osztályhoz tartozó metódusok információit írja le. Egy 32 bites integerrel kezdődik, ami a metódusok számát jelzi, majd a metódusok adatait tartalmazza a következő formátumban:

8. táblázat: a metóduslista egy eleme

offset	méret	leírás
0x0	4	a metódus indexe
0x4	4	elérhetőség flagek (private, public, protected)
0x8	4	a metódusok által dobható kivételek listája
0xC	4	a metódus implementációjának fejlécének elérhetősége a fájlban

Azt, hogy egy metódus konkrétan mit csinál, azt a code header fájlblokk mondja meg. Ennek formátuma:

9. táblázat: a code header fájlblokk szerkezete

offset	méret	leírás
0x0	2	a metódus által használt regiszterek száma
0x2	2	a metódus paramétereinek száma (ide számítjuk a „this” mutatót is a nem statikus metódusoknál)
0x4	2	a kiment típusának mérete
0x6	2	kitöltés
0x8	4	a metódus implementációját tartalmazó fájl nevének string literálja
0xC	4	a metódus aktuális kódjának fájlon belüli helye, ami egy 32 bites integerre mutat, ami a dalvik VM által értelmezett 16 bites utasítások számát tartalmazza, majd a fájl ezt az integert követő részében találhatóak az aktuális VM utasítások, amiket a dalvik végrehajt
0x10	4	a metódus által dobható kivételek listájának helye a fájlban
0x14	4	hibakeresési célból tartalmaz a fájl egy listát, ami a forráskód sorszámaihoz rendeli a metódus .dex fájlba csomagolt kódjának részleteit, ennek a listának a fájlon belüli helyét tartalmazza ez a mező
0x1C	4	a metóduson belüli lokális változók listájának helyét mutatja

Már csak egy dolog hiányzik ahhoz, hogy egy java nyelvű alkalmazás teljes forráskódjának ilyen módú reprezentációját leírjuk, mégpedig a lokális változók leírása. Mint minden mást, ezt is egy listában tárolja a .dex fájl. A lista egy 32 bites integerrel kezdődik, ami a metódus lokális változóit tartalmazza, majd a változókat leíró rekordok következnek, ezek formátuma:

10. táblázat: egy változó leírása

offset	méret	leírás
0x0	4	a változó láthatóságának kezdete
0x4	4	a változó láthatóságának vége
0x8	4	a változó nevéhez tartozó string indexe a string táblában

0xC	4	a változó típusának leíró stringjének indexe a string táblában
0x10	4	a regiszter száma, amibe a változó értéke kerül

Ebben a listában szerepelni fognak a metódus paraméterei is épp úgy, mint a nem statikus metódusok esetében a „this” mutatót.

Egyik legnagyobb előnye a .dex fájlok használatának, hogy az alkalmazás mérete szignifikánsan csökken. Példaként felhoznám a Dan Bornstein 2008-as Google I/O-n tartott prezentációjában említett adatokat, a teljes prezentáció megtekinthető [5] oldalon:

Az ébresztő óra alkalmazás mérete:

- tömörítetlen .jar fájlként: 119.200 byte
- tömörített .jar fájlként: 61.658 byte
- tömörített .dex fájlként: 53.020 byte

Látható, hogy az alkalmazás mérete az eredeti 40%-ára csökkent. Általánosan elmondható, hogy hasonló méretcsökkenést jelent a dalvik virtuális gép számára értelmezhető dex fájlok használata jar csomaggal, vagy a tömörítetlen .class fájlokkal szemben.

A dalvik fejlesztői a memóriát 4 különböző típusra osztják fel két tengely mentén:

1. a lefoglalás módja alapján
   a. clean: mmap()-al foglalt memória: olyan, aminek a tartalma hozzáköthető egy fájlhoz, vagy egyszerűen létrehozható újra (pl.: csupa nulla)
   b. dirty: malloc()-al foglalt memória: amelyet egy program futási időben foglal le
2. a memóriaterület megosztottsága alapján
   a. private: olyan memória, amelyhez csak egy process fér hozzá
   b. shared: olyan memória, amelyhez bármely process hozzáférhet

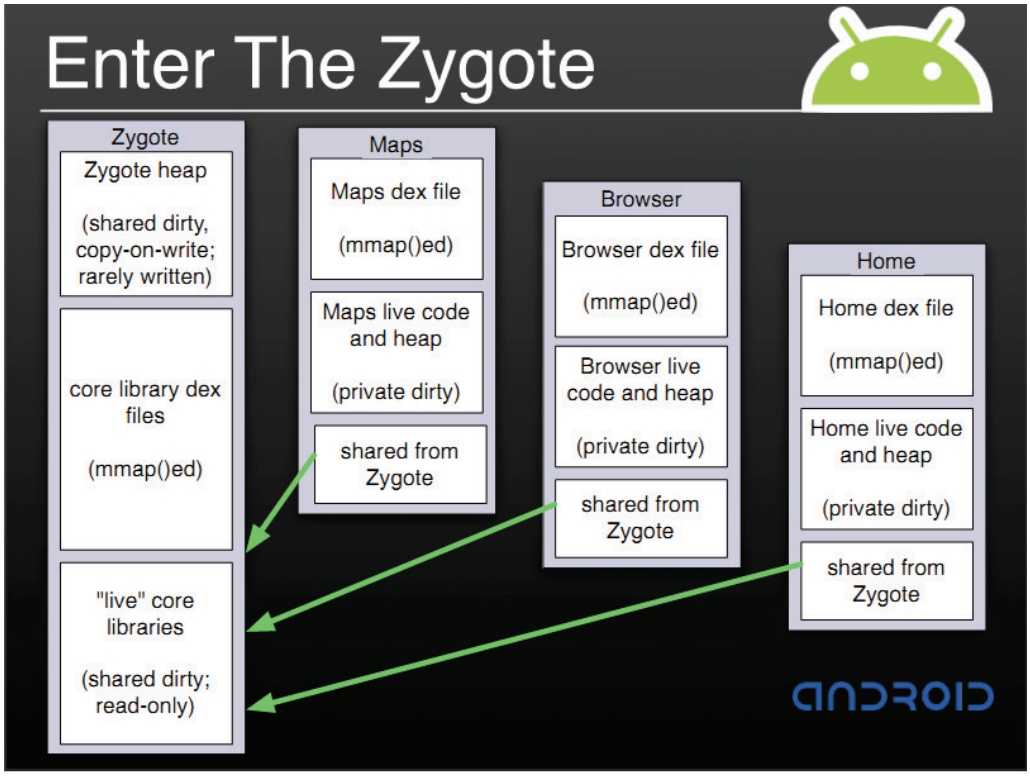
A clean memória nem jelent veszélyt a rendszerre, hiszen csak egy fájl memóriabeli másolata, a private memória, legyen az dirty vagy clean szintén nem okozhat gondot, legyen bármi is benne az nem elérhető más processek számára. A problémát a dirty

shared memória jelenti, ehhez bármely process hozzáférhet, és rosszindulatú módosítása katasztrofális következményekkel járhat a rendszer működésére nézve.
Az Android a következőképpen gazdálkodik a memóriával:

- a clean memóriába kerülnek az általános dex fájlok (a programkönyvtárak) illetve az alkalmazások dex fájljai, legyen ez shared vagy private, nem okozhat gondot a rendszerre nézve
- a private dirty memóriába az alkalmazás futási idejű adatai kerülnek (pl: ha példányosítunk egy osztályt futási időben, a létrehozott objektum ide kerül be, ezért tekinthető az android heap területének)
- a problémás shared dirty memória kezelésére jött létre a Zygote nevű process

A Zygote (zigóta) process a boot folyamat korai szakaszában elinduló VM folyamat, bizonyos gyakori osztályokat betölt és inicializál, hogy a későbbi folyamatok indulási idejét csökkentse. Amikor betöltődött, akkor csak figyel egy socket-et és ha egy alkalmazás el akar indulni, akkor a Zygote process egy a unix fork() rendszerhívásnak megfelelő tevékenységet hajt végre magán, majd a létrejött Zygote másolat process gyakorlatilag átváltozik az induló alkalmazássá, így jön létre az új alkalmazás process-e, és így kap memóriát is. Az így létrejövő process-ek öröklik a Zygote memória területének tartalmát, így férnek hozzá a rendszerkönyvtárakhoz.

Így néz ki ez a gyakorlatban:



2. ábra: a Zygote process felépítése[7]

Ezáltal a shared dirty memória igazából a Zygote heap-jének felel meg, ezáltal van manage-elve a memóriának ez a része is.

Az dalvik a JVM-hez hasonlóan ismeri és kihasználja a garbage collection technológiát, mivel minden process saját memória területtel rendelkezik, ezért ezeknek külön garbage collectoruk van, amik figyelembe veszik a memória megosztást, nem fordulhat elő, hogy felszabadítja az egyik process GC-je a másikkal megosztva használt memóriaterületet. Mivel a processeknek külön külön GC-je van, ami kis valószínűséggel fut egyszerre, ezért lényegesen kisebb terhelést jelent a szemétgyűjtés a rendszer egészére nézve, mint a JVMszerű masszív szemétgyűjtés, ami a teljes JVM-et lefedve és lelassítva egyszerre szabadítja fel az összes process memóriáját.

[7] http://codinglog.googlecode.com/files/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf

A 2.2-es Android nagy újdonsága volt a JIT, vagyis a Just In Time compiler. Ez röviden azt jelenti, hogy a virtuális gép által futtatott kódok bizonyos részeit a futtató platform processzorának nyelvi készletének megfelelőre fordítják.

2.2 Néhány szó a JIT-ről

A létezésének oka könnyen magyarázható, a virtuális gépek jellemzően lassúak, hiszen a virtuális gép valós időben interpretálja a byte-kódot, a processzor ezt az interpretálást is kiszolgálja, valamint a virtuális gépből érkező utasításokat is végrehajtja. Így látható, hogy a ez az architektúra minden utasítást kétszer dolgoz fel, először az interpretáció majd maga a futtatás fogyasztja az erőforrásokat.

Gondolhatnánk arra is, hogy egyszerűen hagyjuk ki ezt és fordítsuk gépi kódra az alkalmazásainkat ám ezzel sok a biztonságosság szempontjából fontos tulajdonság vész el.

A másik lehetőség a kettő között van, abban áll, hogy bizonyos kiválasztott részeket lefordítunk gépi kódra ezzel növelve a hatékonyságot.

Ezt már persze sokan sokféleképpen megcsinálták néha több, néha kevesebb sikerrel. Az egész dolog kulcsa az, hogy mikor és mely részeket fordítjuk át. A mikor nyilván azért fontos, ismét visszatérve az erőforrás szűkösségre, mert ha egy telefon akkumulátorról működve egyszer csak úgy dönt, hogy a rajta lévő 2-300 alkalmazást lefordítja gépi kódra, akkor nem igazán számíthat sok sikerre a rendszer gyártója. A fordítás idejének másik megközelítése az, hogy a futási ciklus melyik részén kezdünk fordítani, lehet ez telepítés közben, a program indításakor, a metódus hívásakor vagy akár az aktuális kódrészlet betöltésekor. A mit pedig azért érdekes, mert ha visszagondolunk az egész rendszer architektúrájára, akkor láthatjuk, hogy egyes kulcselemek már C/C++ nyelven íródtak és csak az igazán magas szintű részek készültek Java-ban és futnak interpretálva a Dalvik-on. Elképzelhető olyan architektúra, ami a teljes programot fordítja, vagy olyan, ami egy könyvtárat, de akár olyan is, ami egy metódust vagy egy részét vagy akár csak maréknyi utasítást is fordíthat.

A fentiek ismeretében adja magát a kérdés, hogy miért is kellett a Dalvik-hoz a JIT fordító?

A válasz két fontos részből áll. Először is fontos tudni, hogy 2009-2010-ben ez nagyon sokszor előforduló kérés volt a médiában és a Google fórumain, sokan mondták, hogy jó már ez az Android, de sokkal jobb lehetne JIT-el, hiszen ezekben az időkben jelent meg pl a dolgozat elején már említett HipHopVM PHP-ra, szóval eléggé aktuális volt a téma. A másik fontos ok az, hogy bár az alkalmazások lényeges hányada nem vagy keveset profitál a JIT létezéséből, bizonyos alkalmazások, amik kifejezetten sok számítást végeznek interpretált környezetben lényeges teljesítményjavulást várhattak ettől az új funkciótól.

Fontos megjegyezni, hogy még a JIT bejelentése előtt megszületett az Android NDK vagyis a Native Development Kit ami lehetővé teszi egy alkalmazás bizonyos részeinek vagy akár egészének C++-ban való megírását.

Miután körülírtam a lehetőségeket bemutatom, hogy milyen a konkrét implementáció, amit az Androidra építettek:

Először is kezdjük a mit-el. A rendszert fejlesztő mérnökök úgy döntöttek, hogy csak bizonyos kódrészleteket fognak lefordíttatni a JIT-el. Ám meghagyták a lehetőséget, hogy akár teljes metódusok is fordíthatóak legyenek, de ezzel nem él az Android, csupán a jövőre gondolván merült ez fel, hiszen ha a telefont bedugjuk tölteni, akkor már nem szűkös az akkumulátor energiakapacitása. Sarkalatos pont az egészben, hogy mi alapján választják ki azt a néhány kódsort, amit fordítanak.

A mikor kérdése már érdekesebb, de egyben meg is magyarázza, hogy mi alapján választ a JIT kódot. A mérnökök úgy döntöttek, hogy amikor egy adott utasítás sorozat elindul, akkor mindig megnövel egy hozzá tartozó számlálót. Amennyiben ez a számláló elérte a kritikus szintet az interpretálás még egyszer megtörténik, de már egy speciális “trace tracking” vagyis nyomkövető módban, aminek az a lényege, hogy a JIT fordító feljegyzi, hogy mit csinált a kód, majd ezt optimalizálja és végül az alkalmazáshoz exkluzívan tartozó lefordított kód tárolóba helyezi, majd a következő indításnál már ezt futtatja.

A kezdeti tesztekben meggyőző akár 3-6szoros teljesítmény többletet jelentett a JIT azokban az esetekben, ahol az alkalmazás lényeges mennyiségű interpretált kóddal rendelkezett.

3. Az Android építőkövei

Egy az Androidon futó alkalmazás teljesen el van zárva az összes többi alkalmazástól. Minden process külön memóriaterületen és külön felhasználóként fut a Linux kernelen. Ezáltal elkerülhető az, hogy egyik alkalmazás a másik memóriaterületét beszennyezze, vagy kompromittálja annak adatait, esetleg a másik alkalmazás valamely metódusát meghívja, vagy egy mezőjének tartalmát átírja.

Amikor a példaalkalmazás fut egy ssh-n elért terminálon kiadva a ps linux parancsot láthatjuk, hogy az alkalmazás a saját felhasználói nevével (app_241) fut, ezért a fájljai nem lesznek elérhetőek a többi alkalmazás számára:

```
localhost / # ps | grep bboti86

app_241   31319 187   97364  19452 ffffffff afd0c5bc S net.bboti86.android.mq
```

Természetesen az alkalmazások kommunikálhatnak egymással, egy jól meghatározott módszer segítségével. Intent-eknek nevezett objektumok jelentik a kulcsot, ezek nagyban hasonlítanak az eseményekre az asztali operációs rendszerekben.

3.1 Activity

Egy olyan alkalmazást, amelyiknek vizuális reprezentációja van (GUI) Activity-nek nevez a rendszer. Egy Activity-t egy java osztály reprezentál a kódban, aminek ősosztálya az android.app.Activity.

Az Activity legegyszerűbb forráskódjának kivonata:

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Mindössze azért másoltam ezt ide, mert szépen látszanak benne, azok a metódusok, amelyek az Activity életciklusát kezelik. Illetve van benne még egy érdekesség: a Bundle típusú objektum, amit az onCreate metódus kap paraméterül. Egy nem bonyolult alkalmazásban az onCreate semmi mást nem csinál a Bundle-el, csupán egy super(savedInstanceState) hívással átadja az ősének. Azonban egy bonyolultabb alkalmazás már kihasználhatja ezt. Az alkalmazások életciklusának tárgyalásakor kifejtem, hogy bizonyos esetekben a rendszer megölheti az alkalmazást. Ebben az esetben az onSaveInstanceState() illetve az onRestoreInstanceState() metódusok felüldefiniálásával elmenthetjük bizonyos változóink értékét illetve visszaállíthatjuk őket. Hasznos lehet ez például akkor, ha egy szövegmező szövegét tesszük a Bundle-be, így biztosítva, hogy ne vesszen el a szöveg, ha esetleg leállítja az Android a szolgáltatást. Az Activityről sok érdekes információ megtudható a [6] linket követve.

Az Activity adatai (mezői) és szolgáltatásai (metódusai) is teljesen be vannak zárva, az Activity-n kívülről nem érhető el. Bizonyos esetekben azonban hasznosnak tűnik az, ha egyes alkalmazások csak adatokat, másik csak szolgáltatást nyújtanak. Ezeket rendre a Content Provider, illetve a Service teszi lehetővé.

3.2 Content Provider

Kezdjük a Content Provider-rel. Ennek a reprezentációja kód szinten szintén egy osztály, aminek ősosztálya az android.content.ContentProvider. Ennek fontosabb metódusai az

1. onCreate() amivel a ContentProvider-t inicializáljuk
2. query(Uri, String[], String, String[], String) ami adatokat szolgáltat a hívófélnek
3. insert(Uri, ContentValues) ami új adatokat vesz fel a providerbe
4. update(Uri, ContentValues, String, String[]) ami a már létező adatokat változtatja meg
5. delete(Uri, String, String[]) ami adatokat töröl a providerből
6. getType(Uri) ami a provider által kezelt adatok MIME típusát adja vissza

Nem véletlen, hogy a metódusok nevei hasonlítanak az adatbázis kezelő rendszerekben használt terminológiával, a ContentProvider lényegében beburkol egy sqlite adatbázist, ami az alkalmazásunkhoz tartozik.

Egy URI példát mindenképp mutatnék, hogy nyilvánvalóbb legyen, hogy néz ki egy ilyen URI:

```
market://search?q=pub:”bboti86”
```

Ez a példa az általam publikált programokat keresteti a market alkalmazással. Az alkalmazások marketon történő megosztásáról lesz még szó a dolgozatban is, de egy részletes leírás található a [7] alatt.

Azt már bemutattam, hogy hogyan érhetünk el olyan adatokat, amelyek nem a mi alkalmazásunkhoz tartozik, hátravan az, hogy megnézzük, hogyan érhetjük el egy külső alkalmazás valamely szolgáltatását. Ehhez a kulcsot a Service jelenti.

3.3 Service

Egy Service-t is, mint egy ContentProvider-r vagy egy Activity-t egy osztály reprezentál a kódban, melynek ősosztálya az android.app.Service.

Fontos megjegyezni, hogy a service nem feltétlenül egy külön process a kernel szemszögéből, de akár lehet külön processben is futtatni, nem is külön szál, de lehetséges külön szálban is futtatni.

A service nem más, mint egy olyan utasítás sorozat, amit esetlegese az alkalmazás GUIjának eltűnése után is futtatni akarunk. Talán a két legjobb példa egy letöltés kezelő, ami a háttérben tölt le bizonyos fájlokat a hálózatról vagy egy zene lejátszó program, ami a háttérben is folytatja a lejátszást. A Service leírása megtalálható [8]-at követve.

A szakdolgozat végén lévő példa alkalmazás egy megvalósít egy Activity-t valamint az Android részét képező egyik Content Providert illetve egy Service-t is használ.

3.4 Broadcast Receiver

A Broadcast Receiver lényegében az eszközön futó alkalmazások közötti üzenetszórást teszi lehetővé. A lényege, hogy ha az egyik alkalmazásnak olyan információi vannak, amit más alkalmazások is hasznosítani tudnak, akkor egy üzenetet küld a rendszernek, ami az erre az üzenettípusra regisztrált alkalmazásokat felébreszti és ezek feldolgozhatják az üzenetet, vagyis az információt. Olyan helyzetekre kell gondolni, mint például ha a képernyő kikapcsol, vagy telefonhívás érkezik. Egy Androidos alkalmazásnak illik leállítani a tevékenységét, ha bejövő hívást érzékel.

3.5 Intent

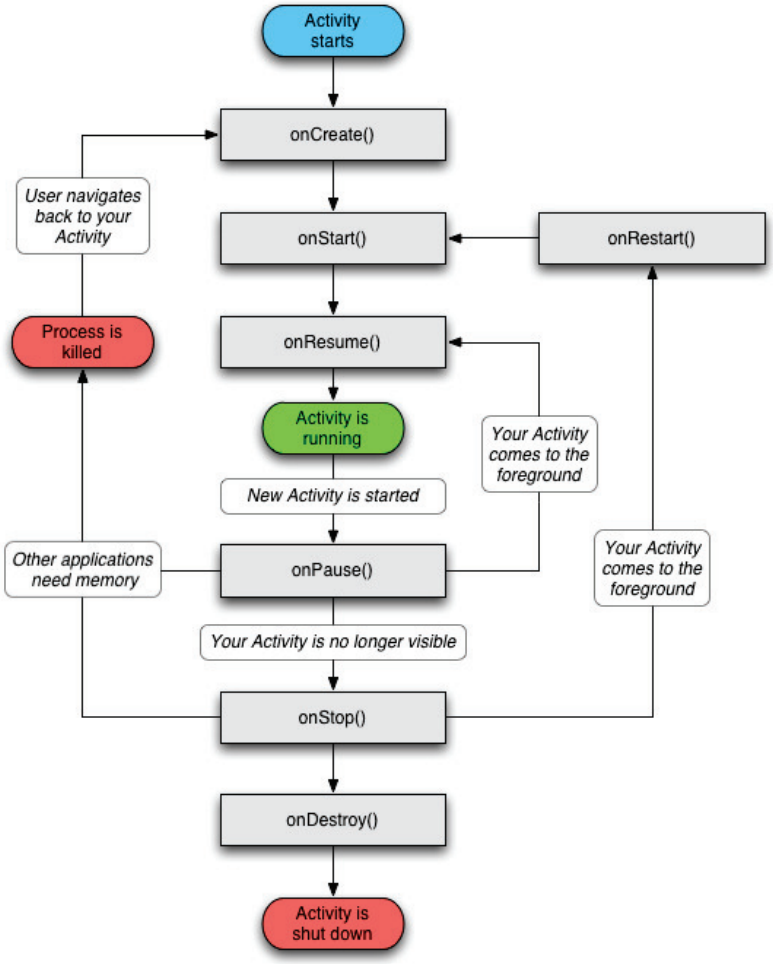
Az üzenet szórás és alapvetően az alkalmazások és azok egyes részei közötti kommunikációnak a standard Androidos eszköze az intent. Mint minden más az intent is egy osztály. A funkciójuk lényege, hogy bizonyos “acion”-öket ad át az egyik féltől a másiknak, ez az acion lehet az, hogy elinduljon egy Activity, de akár az is, hogy egy bizonyos erőforrást használjon. Az Intentbe belecsomagolhatunk extra információt, például egy URL-t amivel az Action-nek megfelelő tevékenységet fog a hívott Activity végezni. Azt, hogy egy adott alkalmazás milyen Intent-eket fogad el, azt az

androidmanifest.xml írja le. Az androidmanifest.xml tüzetes leírása a [9] mögött található.

4. Egy Android alkalmazás életciklusa

Míg egy Windows-on vagy Linux-on futó alkalmazás életciklusa kicsit sarkítva a következő: valamilyen felhasználói interakció vagy valamilyen esemény kiváltódása miatt elindul, fut egy darabig majd dolgát végezvén leáll. Természetesen léteznek szolgáltatások, amelyek a háttérben futnak tovább is. Amikor egy a fent említett platform kifogy a memóriából, akkor egy régebben használt, de még futó alkalmazás memóriaterületét kiírja lemezre a swap fájlba/partícióra és ezzel megoldja a szűkös memória okozta gondot. A probléma ott kezdődik, amikor egy alkalmazás Androidon fut, akkor szignifikánsan kisebb memória területtel gazdálkodhat, és swap terület sem érhető el a rendszerben. Az Android ezt úgy oldja meg, hogy bármikor leállíthatja bármely alkalmazás futását. Ezt a funkciót a rendszerben low memory killer-nek nevezik. Ugyanakkor ez a megoldás egy új problémával szembesít: ha egy rendszer bizonyos időközönként egyszerűen leállítja a rajta futó alkalmazásokat, akkor az alapvetően halálra van ítélve, hiszen senki sem szeretné, ha a felhasználói élményét ilyen drasztikusan rontanánk, gondoljunk arra, hogy egy böngészőnek attól még, hogy a háttérbe került tudnia kell, hogy milyen oldalt böngésztünk, és hogy az azon lévő esetleges form-ba milyen adatokat írtunk be. Mivel az Android egész gyorsan terjed, ezért egyértelmű, hogy erre is született megoldás. A rendszer fejlesztői úgy oldották

meg ezt, hogy lényegesen több lépést tettek egy alkalmazás életciklusába.



Íme:

3. ábra: az alkalmazás életciklusai[8]

Látszik, hogy az alkalmazásnak még mindig 3 fő állapota van: futás előtti, futás közbeni, futás utáni. Az újdonságot az jelenti, hogy sok al-állapotot vezettek be.

Minden egyes állapotnak megfelel egy metódus az Activity-ből származtatott osztályban. Az onCreate-nek például az onCreate() metódus. Amikor egy alkalmazás ebbe az állapotba kerül, akkor lefut a metódus, ezzel biztosítva, hogy az alkalmazás inicializálódhasson, illetve ha szüneteltetve lett, vagy leállítva, akkor elmenthesse az adatokat, amikkel éppen dolgozott.

[8] http://code.google.com/intl/hu-HU/android/images/activity_lifecycle.png

Amikor egy alkalmazás elindul, sorra átmegy a következő állapotokon: onCreate, onStart, onResume, amikor ezeken az állapotokon túl van, akkor mondhatjuk, hogy egy alkalmazás fut.
Ha egy alkalmazás a háttérbe kerül, de még mindig látszik, mert mondjuk egy pop-up ablakot dob fel, amivel felhasználói interakciót vár, akkor lefut az onPause(), amikor újra előtérbe kerül, akkor az onResume() metódus.
Amikor egy alkalmazást teljesen eltakar egy másik, akkor az onStop() metódussal megáll a működése. Mindaddig álló állapotban van, míg újra előtérbe nem kerül a felhasználói felülete, ekkor az onRestart(), onStart(), onResume() metódusokkal újra életre keltjük.
Ha egy másik alkalmazás memóriaigénye drasztikusan megnövekszik, akkor elképzelhető, hogy a mi alkalmazásunkat megöli az Android. Ilyenkor az onPause és az onStop állapotok után az alkalmazásunk leáll, de kész arra, hogy újrainduljon. Ilyenkor az onCreate(), onRestart() és onStart() metódusok futnak le.
Ha egy alkalmazás magától áll le, akkor az onPause(), onStop(), onDestroy() metódusok futnak le.
Általánosan elmondható, hogy amikor egy alkalmazás elindul, újraindul vagy visszatér alvó állapotból, akkor épít fel a program kapcsolatot egy „drága” erőforrással (pl: interneten elérhető fájl) illetve ekkor regisztrálja magát bizonyos event-ek fogadására. Ezzel ellenkezően, amikor az alkalmazás elalszik, leáll vagy végleg meghal (onDestroy) akkor bontja ezeket a kapcsolatokat, illevte akkor szünteti meg az esemény figyelést.

5. A View, a GUI és az XML resource

Az android grafikus felületének az ős osztálya a View, és bár ez az osztály létezik a Java kódban is és ott is használható ám a fő előfordulási helyei ennek, de főleg az utódainak azok az xml fájlok, amiket az alkalmazásunk mellé csatolunk.

Hogy jobban megértsük, hogy hogyan kezeli az Android a grafikus felületét és a többi erőforrását előbb ezzel kezdenék.

Az Android lehetővé teszi, hogy bizonyos dolgokat a java kódon kívül definiáljunk.

A resource-okkal kapcsolatos további információk érhetők el a [10]-ben.

Ezek a dolgok a következők:

1. animációk
2. szín információk
3. statikus képek
4. a GUI elemek layout-ja vagyis az elhelyezkedésük a képernyőhöz és egymáshoz viszonyítva
5. menük
6. sztringek
7. stílusok (nagyjából a HTML mellé tett CSS szerepét játssza)
8. tömbök

Ez lényegében azt jelenti, hogy például egy képet ha felhasználok az alkalmazásomban, akkor ahhoz létrejön fordítás közben egy resource XML bejegyzés és utána annak megfelelően hivatkozhatok rá.

Például:

```
public static final class drawable {

        public static final int icon=0x7f020000;

        public static final int logo=0x7f020001;

}
```

Ezeket persze nem nekünk kell kézzel létrehozni, hanem elvégzi helyettünk a fordító.

Innentől kezdve erre a logo nevű képre hivatkozhatok az alkalmazás forráskódjának mind az XML alapú részében, mind a Java kódban.
Mutatok egy példát az XML-re:

```
<Button android:layout_width="fill_parent" android:layout_height="fill_parent"
android:layout_weight="1" android:layout_marginTop="5dip"
android:drawableBottom="@drawable/logo" android:text="@string/letsPlay"
android:textSize="30dip" android:typeface="monospace" android:id="@+id/playButton"
/>
```

A félkövér részen látszik, hogy a drawable mappába helyezett logo nevű resource-t fogom használni az android:drawableBottom pedig azt mondja meg, hogy ezen a GUI elemen fog megjelenni a logó, mégpedig az alsó részén (bottom).
Nézzük meg, hogy mit látunk még itt:
Az XML tag a Button vagyis gomb, ez az Android egyik GUI eleme. Látszik, hogy mindenféle tulajdonságai vannak beállítva az XML segítségével, mint például a már említett drawableBottom. Megfigyelhető, hogy a dőltbetűs részen egy string típusú resource-ra hivatkozik az xml. Ez az Android szoftverek lokalizációjának kulcsa, ugyanis sem a java, sem az xml kódba nem kerül bene a konkrét string, hanem egy külön resource fájlba kerül, amit a rendszer az éppen aktuális beállításainak megfelelően olvas be és ez alapján írja ki például erre a gombra a szöveget. Azonban lehetőség van arra, hogy az xml fájlban adjunk meg konkrét stringet a gombnak, ezt az android:text=”a konkrét sztiring” atribútummal tehetjük meg. Természetesen ez még mindig csak statikus szöveg, de akár dinamikusan a Java kódból is adhatunk meg sztringet ha a button.setText(“dinamikus szöveg”); kódrészletet használjuk.
Még egy speciális xml attribútumra hívnám fel a figyelmet ez pedig az aláhúzott android:id. Ezt használja a rendszer arra, hogy a dinamikusan beállított tulajdonságoknál egyértelműen meg tudjuk különböztetni ezt az erőforrást az összes többitől.
Bár egy másik gombon, de a példaalkalmazásban állítok be dinamikusan attribútumot, mutatom is, hogyan:

először létrehozzuk a Button típusú változónkat

```
Button button;
```

majd a változóba beírjuk a konkrét gombunk id-ját

```
button = (Button) findViewById(R.id.playButton1);
```

ezután az objektum egyik metódusát hívva beállítjuk a gombon megjelenő szöveget

```
buttons[i].setText(zenek[i].getTitle());
```

Ebből a példakódból újabb érdekességet tudok megmutatni. A findViewById metódus egy int-et vár és egy Object-et ad vissza. A findViewById leírása: [11]. Az érdekesség ott van, hogy ezt az R osztályt nem én hoztam létre, nem én példányosítottam és úgy alapvetően nem is nagyon tudom befolyásolni a létezését. Egy valamit tudok vele csinálni, az XML-ekben az android:id attribútummal definiált id-k alapján tudok gyakorlatilag bármit elérni a java kódból. Ezt az R (vagyis resource) osztályt a fordító hozza létre, és tényleg csak annyi a célja, hogy az XMLekben definiált dolgokat elérhessük általa, tegyük hozzá, hogy ezt nagyon jól végzi, hiszen őáltala válik egyszerűvé az absztrakció, gondolok itt a többnyelvű programokra éppúgy, mint arra, hogy az alkalmazás támogassa a különböző képarányú vagy felbontású kijelzőket.
Megmutatom azért az előző gombot az R-ben:

```
public static final int playButton=0x7f05000c;
```

Most, hogy már bemutattam a resource kezelést az XMLek és az R osztály segítségével kicsit beszélhetek a GUI elemekről, a Viewkról.
Az összes látható elem őse a Java kódban a View. Az XMLben viszont nincs öröklődés, itt nem kell ilyesmivel foglalkozni, annyi az egész, hogy csak néhány View leszármazott lehet az XML gyökéreleme:

- ListView - Listaszerű nézetet generál, nagyon hatékony eszköz sok ismétlődő alakú adat megjelenítésére, mint például a telefonkönyv bejegyzései

- GridView - rácsos formában mutatja az adatokat, amiket átadtunk neki, jó példa rá egy galéria alkalmazás
- TableView - táblázatszerűen sorokba majd oszlopokba tagolhatjuk az összes többi View leszármazottat
- LinearLayout - ami csak egyszerűen egymás alá vagy mellé pakolja az összes elemét
- RelativeLayout - nagyon hatékony eszköz, a benne lévő elemek egymáshoz viszonyított helyzetét írhatjuk le
- TabLayout - tabokra vagyis fülekre oszthatjuk a benne lévő elemek megjelenését, így egyszerre csak az egy fülhöz tartozó elemek vannak szem előtt, erre nagyon jó példa egy böngésző
- AbsoluteLayout - bár van rá lehetőség, hogy pixelre pontosan megmondjuk, hogy mi hol legyen, de ez nem ajánlott, mert nem az összes androidos eszköznek lesz ugyanolyan méretű, felbontású, képarányú és pixelsűrűségű kijelzője

A List- és a GridView úgynevezett adapterek segítségével kapja meg azt, hogy milyen adatokat kell neki megjeleníteni jellemzően futási időben, az összes többinek jellemzően az előre definiált gyerekelemeinek megfelelő View-kat kell megjeleníteni.
Már ezek az akár gyökérelemként is használható eszközök is egymásba ágyazhatók, például betehetek egy vertikális LinearLayoutba két horizontálisat, így két egymás alatti sorban lesznek elhelyezve a grafikai elemek.
Vannak kifejezetten egy célra létrehozott View-k is:

- DatePicker, TimePicker - ahogy a nevűk is sugallja dátum illetve idő kiválasztására használhatóak
- MapView, WebView - Google Maps térképek és weboldalak megjelenítésére használhatjuk őket
- Spinner, Galery - szöveges illetve grafikus adatelemek közötti választást könnyítik meg

- Auto Complete View-k, például az AutoCompleteTextView, ami egy megadott adathalmazból választja ki a már beírt karaktereknek megfelelő adatelemet

A teljesen általános feladatokat pedig a következők fedik le (a lista nem teljes, csak betekintést kíván nyújtani, nem kíván teljes referenciaként szolgálni):

- Button - gomb, megnyomásakor egy event váltódik ki, amit az onClickListener lekezel
- TextView - statikus szöveg megjelenítésére
- EditText - szöveg beviteli mező
- CheckBox, RadioButton, ToggleButton - választási lehetőséget adnak a felhasználónak
- RatingBar - egy ötös skálán értékelhet a felhasználó, ezt használja például az android market az alkalmazások értékelésére
- és még nagyon sok egyéb úgynezett widget található az androidban...

A View-k leírása és a Widget csomag többi tagja a [12]-ben és a [13]-ban.

Ezzel úgy érzem, hogy ki is merítettem a grafikus felületek és az XML alapú erőforrások témáját, összefoglalva az a lényeg, hogy a meglévő View-kat és Widgeteket szabadon kombináljuk, így összetett ám mégis könnyen kezelhető GUIt kapunk.

Egy igazán haladó példa az, ha a ListView előre definiált saját View-kat jelenít meg, mondjuk az aktuális adatelemeknek megfelelően, mondjuk hőmérséklet értékeket úgy, hogy ha hideg van, akkor kék, ha meleg, akkor piros betűket használva.

6. Egy Android alkalmazás mappaszerkezete

A fejezet zárásául már csak annyit mutatnék, hogy a konkrét erőforrásainkat és a forráskódunkat hogyan helyezzük el, hogy a fordító meg is találja. Ehhez a konkrét példaprogram forráskódjának a könyvtárstruktúráját fogom alapul venni.

6.1 Fordítás előtt

(felhívnám a figyelmet, hogy ez már egy lefordított és becsomagolt alkalmazás):

11. táblázat: a fordítás előtti fájl hierarchia

Fájl/mappa név (a félkövérek mappák)	Leírás
./AndroidManifest.xml	az alkalmazás alapvető beállításait tartalmazza, a nevét, a benne lévő Activityket és még sok minden mást, egy kielégítő lista itt található: http://developer.android.com/guide/topics/manifest/manifest-intro.html
./defalt.properties	automatikusan generált fájl ami beállításokat tartalmazhat, legtöbb esetben csak a megcélzott Android verzióról
./profuard.cfg	a nemrég bevezetett obfuszkációs vagyis a kódot visszafejthetetlenné tevő utófeldolgozó beállításai
./bin	a lefordított osztályok találhatóak benne
./bin/classes.dex	a java kódból fordított dex vagyis Dalvik EXecutable fájl
./bin/music quiz.apk	a csomagolt ám csak debug kulccsal aláírt alkalmazás, ezt egy eszközre felmásolva már telepíthetjük is

./bin/resources.ap_	szintén egy apk vagyis egy zip fájl, benne az összes erőforrással, amit xml-ben definiáltunk, illetve egy az xmleket összefűző bináris fájllal, valamint az androidManifest.xml-el
./bin/net/bboti86/android/mq/*	a java fordító által generált .class fájlok
./gen/net/bboti86/android/mq/R.java	az Android fordító által generált R.java
./res	a resource-ok és az őket leíró xml-ek helye
./res/drawable/* (de lehetne akár **./res/drawable-hdpi**)	a grafikák, amiket az alkalmazás használ, ide értve az ikonját is A -ldpi, -mdpi, -hdpi mappavégződések alapján az Android kiválasztja, hogy az éppen használt képernyő felbontásához és pixelsűrűségéhez melyik grafikus fájl illik legjobban
./res/layout/*	a GUI-t leíró layout fájlok általánosan
./res/layout-land/*	a landscape módban használt GUI leírása
./res/values/strings.xml ./res/values-hu/strings.xml ./res/values-de/strings.xml	az alkalmazás által használt sztringek, fontos látni, hogy mind itt, mind a layout-oknál meg kell adni specifikációk nélküli “univerzális” erőforrást, itt a hu a magyar, a de a német nyelvű eszközökön töltődik be, ám egy mondjuk török Android a specifikált nyelv nélkülit töltené be
./net/bboti86/android/mq/*	az alkalmazás java nyelvű forrása

6.2 Fordítás után

A fordítás és csomagolás után egy .apk kiterjesztésű fájlt kapunk, ez maga az alkalmazás, amit publikálhatunk akár az Android marketen, de akár a saját honlapunkon keresztül is.

Ebben egy kicsit átrendeződnek a dolgok:

12. táblázat: a fordítás utáni fájlhierarchia

Fájl/mappa név (a félkövérek mappák)	Leírás
AndroidManifest.xml	ez nem változik
classes.dex	az alkalmazás dalvikos kódja (ez már tartalmazza a string-eket)
resources.arsc	a resource xml-ek összefűzve, kivéve a layout-ok
./META-INF/CERT.RSA ./META-INF/CERT.SF ./META-INF/MANIFEST.MF	az alkalmazás ezeknek a fájloknak a segítségével lesz aláírva
./res/drawable/* **./res/layout/*** **./res/layout-land/***	a grafikák és a layout.xml-ek

Látszik, hogy eltűnik a forrás, ami talán nem is annyira meglepő, ami viszont már érdekesebb, hogy a .class-ok is eltűnnek. Ezt tüzetesen megvizsgáltam a DalvikVM-et és a dex fájlformátumot leíró részben.

7. Az SDK egyéb eszközei és a market

Az Android SDK sok egyéb eszközt tartalmaz. Ezek főleg különálló alkalmazások formájában érhetőek el. Most bemutatok néhányat illetve a fejezet végén körüljárom a market-et is, ami lehetővé teszi, hogy az alkalmazásunk elérhető legyen az egész világ Android felhasználóinak.

7.1 Android Debug Bridge

Az Android Debug Bridge vagyis adb egy nagyon erőteljes eszköz. Arra szolgál, hogy a telefonon közvetlenül tudjunk alkalmazásokat tesztelni. Ehhez elég komoly eszközkészletet vonultat fel, kezdve azzal, hogy akár távoli eszközökhöz is csatlakozhatunk, egészen addig, hogy ezen keresztül képesek vagyunk pszeudó-véletlen GUI műveleteket szimulálni. A hosszadalmas leírás helyett inkább néhány példát írok le felsorolásszerűen.

- az elérhető illetve szimulált eszközök listázása

  ```
  adb devices
  ```

- alkalmazások telepítése

  ```
  adb install alkalmazás_név
  ```

- fájlok másolása készülékről illetve készülékre

  ```
  adb pull fájlnév helyi_fájlnév
  ```

  ```
  adb push helyi_fájlnév fájlnév
  ```

- az eszköz naplójának elérése

  ```
  adb logcat
  ```

- a távoli parancsértelmező elérése

  ```
  adb shell
  ```

- a Monkey, vagyis a teszt „majom” használata, ami véletlenszerűen nyomkodja az alkalmazásunkat

adb shell monkey –v –p alkalmazás_csomagnév interakciók_száma

ahol a –v kiírja, hogy mit csinál a „majom”, a –p pedig bezárja a „majmot” az alkalmazásunk csomagjába, hogy ne tudjon mondjuk telefonálni, vagy sms-t írni

7.2 Eclipse Android Developer Tools

Ahhoz, hogy Androidra fejlesszünk alkalmazást alapvetően két eszközre van szükségünk: egy egyszerű szövegszerkesztőre (legyen ez a VIM vagy a notepad) és egy parancssorra, ahol a szövegszerkesztővel elkészült fájlokat fordíthatjuk illetve az adb segítségével telepíthetjük és futtathatjuk.
Azonban ha nagyobb kényelemre van szükségünk, érdemes elgondolkodni valamilyen IDE vagyis Integrated Development Environment használatán. Szerencsénk van ugyanis az összes nagyobb Java fejlesztésre használt IDE-hez (IntelliJ, Eclipse, Netbeans) már létezik Androidos kiegészítés.
Én a szakdolgozat készítése közben az Eclipse-t és a beépülőjét a adt-t használtam, mert ez az egyetlen, amit a Google fejleszt és mert kényelmesnek találtam már első használatra is.
Az adt főbb funkciói közé tartozik, hogy képes virtuális Androidos eszközt létrehozni, intelligensen kezeli az XML-ekben tárolt erőforrásokat, a GUI layout-okat WYSIWYG szerkesztővel hozhatjuk létre, automatikusan obfuszkálja a kiadásra kerülő alkalmazást, illetve képes egy előre megadott kulccsal aláírni a .apk-t és ezt exportálni, hogy feltölthessük az Android Market-re.

7.3 Az Android emulátor

Természetesen nincs az a fejlesztő, aki minden létező képarányú, felbontású, pixelsűrűségű vagy szoftver verziójú eszközt képes megvenni, ezért létezik már a kezdeti időktől egy fontos eszköz az SDKban, ami nem más mint az emulátor. Ezzel gyakorlatilag megadhatunk bármilyen paramétert az emulátornak, ami meghatározza, hogy nézzen ki és működjön az emulált eszköz. Végrehajthatunk bármit és kipróbálhatunk mindent a telefonhívástól kezdve a bejövő sms-en át egészen a szimulált GPS koordinátákig. Az eszköz hatékonyságát az adja, hogy az emulátoron ugyanaz az Android rendszer fut, ami az eladásra kerülő eszközökön is.

7.4 Az Android Market

Miután aláírtuk és becsomagoltuk (ezzel kapcsolatban a [14] add további tájékoztatást) az alkalmazásunkat akár az adt-vel, akár manuálisan parancssorból nincs más dolgunk, mint megosztani a világgal.

Néhány évvel korábban még elég kevés eszköz állt egy magányos fejlesztő számára rendelkezésre, hogy terjessze termékét. A nagy kiadók és szoftverházak képesek voltak komoly és drága marketing kampányokkal ismertté tenni amit készítettek, ám

egy kis halnak nem sok esélye volt, hogy egy szűk körön túl bárki megismerje művét bármennyire is jó volt az.
A Linuxokba épített csomagkezelők hoztak először elfogadható megoldást erre, itt csomagnév alapján kereshettünk egy listában, amit a disztribúció létrehozói válogattak össze, minőségi és hasznos alkalmazásokat megosztva.
A mobil platformok jelenlegi helyzete erre építkezik ügyesen felismerve azt, hogy itt egy kis cég vagy akár egyetlen fejlesztő munkája is elég egy jó alkalmazás létrehozására. Az Apple AppStore volt az első szoftver bolt, ami magáról a készülékről elérhető volt, így az egyszerű és a gyors kereshetőséget hozta el. Gyorsan népszerű lett és ma már többszázezer fizetős illetve ingyenes alkalmazás érhető el rajta keresztül a felhasználóknak.
Az OHA már amikor az Android korai változatait elkezdte terjeszteni egy hasonló infrastruktúrát helyezett a rendszer mögé. Ide is bárki feltölthet bármit, ám az Apple-el ellentétben nem kérnek érte több száz dollárt csupán 29-et és nem szűrik nagyító alatt a feltöltendő alkalmazásokat, hanem a közösségre bízzák, hogy ami nem tetszik, vagy nem azt teszi, amit állít magáról, az gyorsan eltűnjön a süllyesztőben. Az az igazság, hogy bár Steve Jobs, az Apple első embere állítja, hogy az Androidos alkalmazások nagy része értéktelen szemét, de úgy néz ki, hogy működik a közösségi értékelés, az igazán komoly fejlesztők figyelnek a felhasználóik véleményére és folyamatosan fejlesztik az alkalmazásaikat.
A Market-el kapcsolatban kell megemlítenem a legnagyobb negatívumot az Androiddal kapcsolatban. Sajnos ma Magyarországról nem érhetőek el a fizetős alkalmazások és e miatt egy magyar fejlesztő csak akkor tölthet fel fizetős alkalmazást, ha külföldön nyit céget, ami már adóügyi vonzatokat vet fel. Ezt a problémát nem csak a Google vagy az OHA okozza, sajnos amíg a magyar mobil szolgáltatók nem erőltetik ezt, nem fogják fontosnak érezni.

8. A mellékelt példaprogram rövid leírása

8.1 az alkalmazás rövid leírása

Röviden leírom, hogyan néz ki a példa program. Amikor elindítjuk egy főmenü fogad minket, ahol lehetőség van új játék kezdésére, vagy a jelenlegi pontszámok és a súgó megjelenítésére.

A főmenü két gombból áll. Itt a TableLayout-ot használom a kívánt kinézet eléréséhez. A felső TableRow, ami a játékindító gombot tartalmazza, nagyobb súllyal szerepel, ezért mérete lényegesen nagyobb, mint az alsóé. A főmenühöz tartozik egy Activity az alkalmazásban, a MenuActivity, ebben van két belső osztály, mindkettőnek a View.OnClickListener az őse, ezek felelősek a gombnyomás esemény kezelésére. Az Intent osztály segítségével indítják a súgóhoz vagy a játékhoz tartozó Activityket. A Button-okhoz a megnyomás eseményét lekezelő metódust a setOnClickListener metódussal rendelem hozzá. A gombnyomás kezelésének egy egészen más módját láthatjuk a játék Activity-jében.

A súgó és a pontszámokat összefoglaló rész egy LinearLayout-ba helyezett két TextViewból áll, amiket már részletesen bemutattam az 5. fejezetben. Érdekesség, hogy a súgónak létezik külön landscape nézettre optimalizált változata. Amikor a telefont elfordítjuk, ezt a nézetet az Android magától tölti be, nincs az alkalmazásban olyan kód, ami ezt kezelné. A pontokat tartalmazó TextView tartalma megváltozik, amikor a View elkészül, a pontszámokat beállítja a kód, ezeket egy SharedProperty-ben tárolja az alkalmazás.

A játékhoz tartozó View is viszonylag egyszerű, 3 sorban hat gombot tartalmaz. A hozzá tartozó Activity azonban sok érdekességet rejt.

Az alkalmazások életciklusait bemutató részben már említett OnCreate és OnPause metódusokat használom arra, hogy a játékot vezényeljem.

Az OnCreate először is a SharedProperty-vel megvalósított perzisztens adattárolót használja, majd beállítja az Activity-hez tartozó Viewt. Ezután az itt meghívott initMusic() metódus a musicCursor nevű változót használva az Android egyik beépített ContentProvider-éből lekérdez 6 véletlen zeneszámot, ezután a View 6 gombjának a

szövegét változtatja meg dinamikusan, majd kiválaszt egy zenét a 6-ból és elkezdi játszani azt a MediaPlayer nevű Service-el.
A OnPause metódusban egyszerűen leállítjuk a zene lejátszást. Korábban már említettem, hogy minden Android alkalmazásnak illik leállni, ha bejön egy hívás az, hogy az OnPause leállítja a zenét ezt is lefedi, hiszen a bejövő hívás egy másik Activity-t fog elindítani, ami miatt ennek az OnPause metódusa lefut, és ezzel leáll a zene.
Ebben az Activityben még három dologra hívnám fel a figyelmet, az egyik az, hogy az Activity maga az OnClickListener, és így a gombok megnyomásának eseményét ő maga kezeli le.
A második érdekesség a Toast nevű osztály, ez egy egyszerű szöveges üzenetet dob fel a felhasználónak, ami a kijelző alján jelenik meg és amikor a kellő idő letelt eltűnik magától.
A harmadik érdekesség a kivételt elkapó ágban van ahol, ha nincs elegendő zene, vagy nem érhetőek el a fájlok egy PopUp-ot használunk. Ennek is van View-ja, a notenoughmusic.xml-ben leírva, ami egyszerűen tájékoztatja a felhasználót a problémáról, majd a kivételkezelő kilép az alkalmazásból. Ezt a View-t azonban, mivel nem az Activity-hez tartozó elsődleges View, amit a setContentView metódussal beállítottunk, fel kell „fújni” az android terminológiának megfelelően. Ez azt jelenti, hogy a View összes elemét példányosítja felparaméterezve a háttérben a ViewManager nevű rendszerkomponens. Erre szolgál a LayoutInflater nevű osztály, ami szintén egy Service az Android-ban.

8.2 Drawable, Layout, String

Egy fontos dolog van még, ami az Android erejét mutatja. Ha valaki végignézi a forráskódot, sehol nem lát olyan string literált, ami megjelenne a grafikus felhasználói felületen, de nem lát a logót tartalmazó fájlnévre még utalást sem.
Lát viszont annál több hivatkozást az R osztályra és annak belső osztályaira. Ez adja az Android igazi erejét szerintem, hogy csak elküldöm e-mailben azt a néhány mondatot, ami a GUIt alkotja egy tegyük fel törökül beszélő ismerősömnek és ha ő visszaküldi

lefordítva, akkor létrehozok egy values-tr mappát, benne egy strings.xml-t amibe beletéve a török mondatokat máris anyanyelvén fog szólni a program következő verziója ahhoz, akinek a telefonja török locale-ra van beállítva. Még egyszer megismétlem, ehhez nem kellett semmi fölösleges programozási feladatot végrehajtani.
Amennyiben a program egy más képarányú vagy méretű kijelzőn jelenne meg, ott sem lenne gond, mert sehol nem adtam meg konkrét méretet, minden dinamikus. Néha azonban szükség van méret értékek beállítására is, ekkor használhatunk egy absztrakt mértékegységet, ahogy én is teszem a főmenü nagy gombján: android:textSize="30dip". A dip vagyis a Device Inependent Pixel (eszköz független pixel) minden készüléken az aktuális kijelző mérethez, képarányhoz és pixelsűrűséghez van hozzáskálázva.
Szerintem a fent említett dolgok, a beépített ContentProvider-ek és Service-k, a dinamikusan átméreteződő GUI és a teljesen automatikus lokalizáció adják az Android fejlesztés legnagyobb előnyét.

8.3 AndroidManifest.xml

Az androidmanifest.xml egy talán a legfontosabb fájl egy Android projectben. Az én példaalkalmazásomban több célja van. Először is a manifest tagben megmondjuk a csomag nevét valamint az alkalmazás verziójának kódját és számát. Azért van két verzió is, mert a versionCode normál verzióként viselkedik, amit inkrementálni kell, ám a versionName egy szabadon választott string lehet, amiben akár az aktuális verzió újdonságait is leírhatjuk, persze mint minden string ez is lehet a values mappákban lévő strings.xml-ekben lévő stringek egyike.
Itt definiáljuk, hogy melyik verziójú SDK-t használjuk. Ebben a példában a 7-est, ami a 2.1-es eclair-t jelenti. Ugyanitt kell megadnunk explicit azt, hogy milyen jogosultsággal látjuk el az alkalmazást. Ezeket a jogokat listázza a telepítő amikor telepítünk, illetve ha az alkalmazás megszegi a jogosultságait, például használná a GPS-t vagy telefonálna anélkül, hogy ezt itt definiáljuk a DalvikVM leállítja a futását.

A példaalkalmazás a MOUNT_UNMOUNT_FILESYSTEMS jogosultságot használja, ez szükséges a médiafájlok lejátszásához.
Az application tag-ben magát az alkalmazást kell definiáljuk, itt adjuk meg a nevét, az ikonját és a leírását. Persze ezek az adatok is jöhetnek a strings.xml-ből.
Van az application tagnek 3 leszármazottja, a három Activity, amiből az egész alkalmazás felépül. Jól látszik, hogy az intent-filter ágak különbözőek, a MenuActivity-hez tartozó MAIN típusú, ez jelenti azt, hogy ez az elsődleges belépési pontja az alkalmazásnak, a kategóriája pedig LAUNCHER, vagyis megjelenik a programindítóban, míg a másik kettő VIEW és DEFAULT action-t és category-t kap, mert ezek nem látszanak különálló alkalmazásként és csak megjelennek, nem kapnak speciális paramétert.

Összefoglalás

A bevezetésben említettem, hogy manapság szinte mindenkinek van mobiltelefonja, és az okostelefon sem luxus vagy méretes, ám buta eszköz. Számomra ezt az bizonyítja leginkább, hogy a közvetlen környezetemben találtam majdnem tíz embert, aki szívesen kipróbálta az általam készített programot, a saját Androidos eszközén.

A fejlesztés gyorsan és viszonylag problémamentesen zajlott. Csupán egy nagy negatívumot tudnék megemlíteni, amikor elkezdtem a szakdolgozatom írását egy autópályadíj fizető alkalmazást szerettem volna készíteni, ami majdnem el is készült, amikor megjelent a marketen egy ilyen alkalmazás, ami bár nem tudta azt, amit az enyém, mégis az első volt. Ez a negatívum, hogy naponta több száz új név jelenik meg a piacon felhasználóként valójában nagy előny, hiszen folyamatosan frissül a kínálat, ami most nincs, holnapra már 3 külön verzióban lesz elérhető. Fejlesztőként azonban teljes embert kíván, aki napi 8-16 órában képes ezzel foglalkozni.

Ebből is látható, hogy most éli az Androidot célzó alkalmazás fejlesztés az abszolút hőskorát, alig egy év alatt megtriplázódott az elérhető alkalmazások száma és a növekedés nem lassul mind a mai napig.

A fejlesztői platformot illetve az operációs rendszert és a felhasználói élményt bemutatni egy egész könyvsorozat sem lenne elég, de remélem, hogy ez a rövid ízelítő elég ahhoz, hogy felkeltse a figyelmet a mobiltelefonokat célzó fejlesztésre és ezen belül is az Androidra.

Irodalomjegyzék

[1] http://en.wikipedia.org/wiki/Andy_Rubin

[2] http://en.wikipedia.org/wiki/Danger,_Inc

[3] http://en.wikipedia.org/wiki/Open_Handset_Alliance

[4] http://en.wikipedia.org/wiki/Android_version_history

[5] http://www.youtube.com/watch?v=ptjedOZEXPM

[6] http://developer.android.com/guide/topics/fundamentals.html

[7] http://developer.android.com/guide/publishing/publishing.html

[8] http://developer.android.com/reference/android/app/Service.html

[9] http://developer.android.com/guide/topics/manifest/manifest-intro.html

[10] http://developer.android.com/guide/topics/resources/available-resources.html

[11] http://developer.android.com/reference/android/app/Activity.html#findViewById%28int%29

[12] http://developer.android.com/resources/tutorials/views/index.html

[13] http://developer.android.com/reference/android/widget/package-summary.html

[14] http://developer.android.com/guide/developing/building/index.html#detailed-build

Egyéb felhasznált irodalom

http://www-igm.univ-mlv.fr/~dr/XPOSE2008/android/fonct.html

http://www.youtube.com/watch?v=G-36noTCaiA

http://www.youtube.com/watch?v=Ls0tM-c4Vfo

http://www.slideshare.net/matthewmccullough/introduction-to-android-by-demian-neidetcher

http://www.fhnw.ch/technik/imvs/publikationen/vortraege/jazoon09_android.pdf

http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

http://blog.vlad1.com/2009/11/19/android-hacking-part-1-of-probably-many/

http://www.dalvikvm.com/

http://en.wikipedia.org/wiki/Dalvik_(software)

http://www.youtube.com/watch?v=RF62c4HgbU4&feature=channel

http://imsciences.edu.pk/serg/wp-content/uploads/2010/10/1.1.png

http://en.ophonesdn.com/uploads/Image/4(17).jpg

http://www.devx.com/wireless/Article/41133/1763/page/2

http://thinkandroid.wordpress.com/2010/01/13/writing-your-own-contentprovider/

http://stackoverflow.com/questions/151777/how-do-i-save-an-android-applications-state

Nagy hatással voltak rám Mark Murphy könyvei, melyek bár hagyományos értelemben nem jelentek meg, de mégis minden Android fejlesztőnek érdeme elolvasni őket: http://commonsware.com/books

Függelék:

F1) Android fejlesztés Linuxon

Egy 10.10-es Ubuntu Linux disztribúció felkészítése arra, hogy az Android platformra fejleszthessünk alkalmazást

1)Java

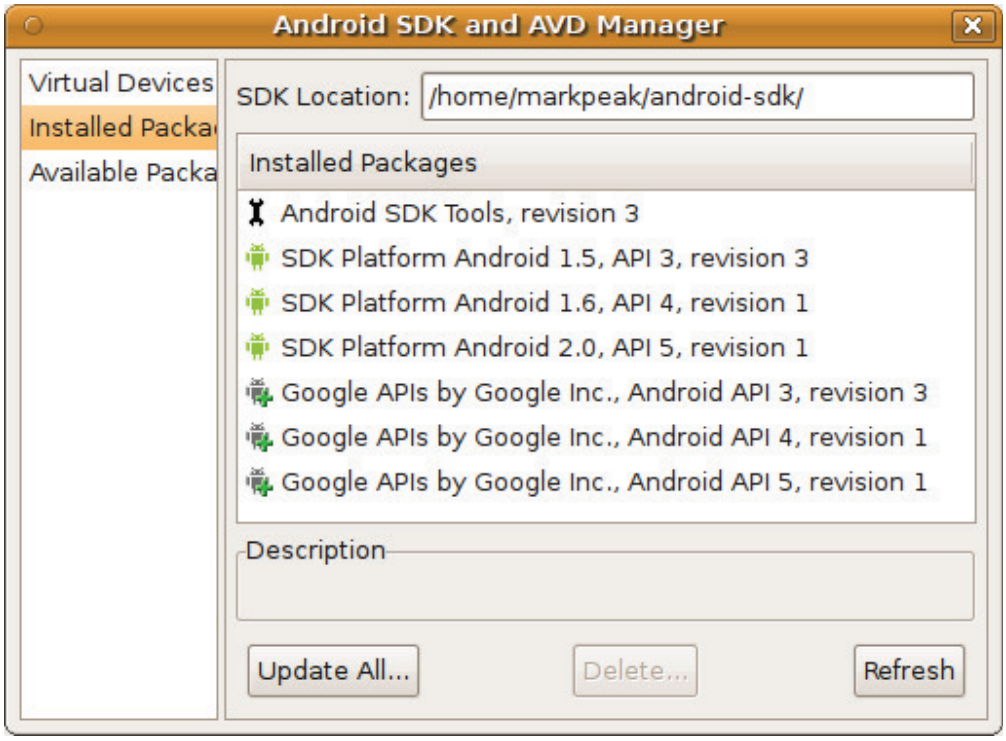
Kezdjük mindjárt az elején, ahhoz hogy minden működjön szükség lesz java-ra. Mégpedig a SUN JDK-jára. Az Ubuntu alapból az OpenJDK-val települ, ezt lehet lecserélni a Sun JDK-jára. Ez egy elég összetett dolog, ezért most nem részletezném, de ha semmiképp nem sikerül, akkor az OpenJDK is jó lesz.

2)Következzen az Android SDK.

A developer.android.com-ról minden gond nélkül le is tudjuk tölteni az aktuális verziót, ami megfelel az operációs rendszerünknek (jelenleg a 10-es verzió, amit a Honeycomb hozott el nekünk). Ez nem más, mint egy sima .zip, amit egy kényelmes mappába ki is csomagolhatunk.

3)Android SDK and AVD manager

A mappában, amit kicsomagoltunk találunk egy tools mappát, ebben pedig egy Android nevű scriptet, ez indítja el az SDK and AVD managert.



a kép innen: http://www.flickr.com/photos/isriya/4084862375/

Itt az available packages menüpontban találjuk az elérhető csomagokat. Válasszunk egy/több SDK verziót és azokhoz tegyük fel az "SDK Platform"-ot, a "Documentation for Android SDK"-t és a "Samples for SDK"-t. Itt meg is jegyezném, hogy a telefonunkon futó Android SDK-ját érdemes feltenni, mert a telefonon tesztelés sokkal gyorsabb, mint az emulátoros. Illetve érdemes az aktuálisan legelterjedtebb Android verziókhoz (jelenleg 2.1, 2.2, 2.3). Tegyük még fel a gépre a kiválasztott verziókhoz tartozó Google API-t is. (a market licensing package sem árthat meg)

4)Eclipse

Ez lesz a legegyszerűbb:

a következő parancsot kiadva feltelepül a legfrissebb Eclipse verzió:

```
sudo apt-get eclipse
```

5)Eclipse ADT

Eclipse-et elindítva a help menüben találjuk az Install new software... menüpontot, a feljövő ablakban rákattintunk az add-ra, beszúrjuk ezt:

```
https://dl-ssl.google.com/android/eclipse/
```

Majd követjük a varázsló lépéseit és a végén újraindítjuk az Eclipse-t, hogy betöltse a plug-int.

Újraindítás után a Window - preferences - android - menüben megadhatjuk az SDK elérési útvonalát.

6)Telefon használata debuggolásra

Windowson ez nem lenne egy nagy kaland, linux alatt viszont már kicsit bonyolultabb. De rövidre tudjuk zárni, ha a

```
/etc/udev/rules.d
```

mappában létrehozzuk az

51-android.rules

fájlt a következő tartalommal:

```
SUBSYSTEM=="usb", SYSFS{idVendor}=="0bb4", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="22b8", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="18d1", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="18d1",
ATTRS{idProduct}=="0c01", MODE="0666", OWNER="[me]"
SUBSYSTEM=="usb", SYSFS{idVendor}=="19d2", SYSFS{idProduct}=="1354",
MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="04e8", SYSFS{idProduct}=="681c",
MODE="0666"
```

Ez lényegében csak annyit csinál, hogy bizonyos vendorID-khez hozzáférést add. E nélkül elég bonyolult lenne az adb használata.

Ezek után dugjuk be a telefont és a menü - settings - applications - development - USB debuging engedélyezi, hogy az előbb már említett adb elérje a telefont.

Próbáljuk ki a dolgot úgy, hogy a második pontban létrehozott mappában kiadjuk a

```
./adb devices
```

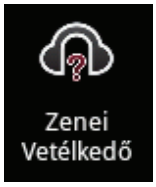
parancsot, aminek hatására kb ezt írja ki a parancssor:

```
List of devices attached
HT97JL90XXXX        device
```

(nyilván a négy X helyén a telefon egyedi azonosítójának a vége látható)

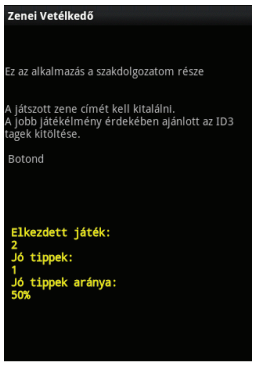
Ezek után már minden a rendelkezésünkre áll, hogy alkalmazást fejleszthessünk Androidra, annyi a dolgunk, hogy az Eclipse-ben létrehozzunk egy új Android projectet, ez egy olyan template alapján jön létre, amit már egyből futtathatunk is és a Hello World Androidos megfelelőjével találjuk szembe magunkat.

F2) Képek a példaalkalmazásból



Az alkalmazás ikonja magyar locale-al

Az alkalmazás főmenüje magyar locale-al



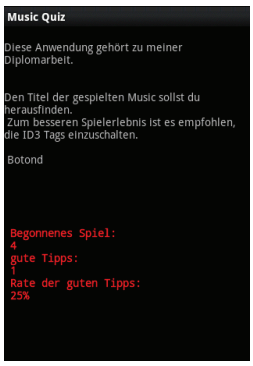
A súgó magyar locale-al

A játék nézete magyar locale-al

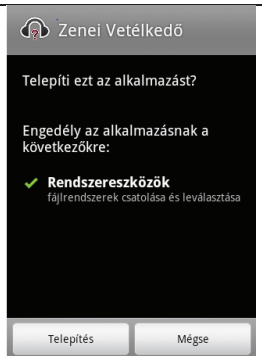
A rossz választ jelző „toast” magyarul

Ugyanez a toast német locale-on

Ismét a főmenü elfordított kijelzőn



A német súgó



Telepítéskor látszik a jogosultság