JYRKI PALKONEN

TEKSTI-TV-SOVELLUKSEN MUOKKAAMINEN MIDP:STÄ ANDROIDILLE

Diplomityö

Tarkastaja: prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
12. tammikuuta 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
PALKONEN, JYRKI: Teksti-TV-sovelluksen muokkaaminen MIDP:stä Androidille
Diplomityö, 44 sivua, 2 liitesivua
Huhtikuu 2012
Pääaine: Hajautetut järjestelmät
Tarkastaja: professori Tommi Mikkonen
Avainsanat: Android, MIDP, siirrettävyys, mobiili

Vuosien varrella ohjelmistokehittäjille kertyy paljon vanhoja sovelluksia ja ohjelmistokomponentteja, joille voisi olla käyttöä uusissa projekteissa. Korkean tason kielet lupaavat helppoa siirrettävyyttä ympäristöstä toiseen, jopa ilman uudelleenkääntämistä. Vanhan koodin ottaminen uusiokäyttöön ei kuitenkaan välttämättä ole kovinkaan helppoa, varsinkin jos tätä ei ole aikanaan mietitty etukäteen ja toteutettu sen mukaisesti. Joissain tapauksissa voi olla helpompi kirjoittaa toteutus kokonaan uudestaan kuin lähteä selvittämään, mitä vanha koodi tekee ja mitkä kaikki osat kuuluvat minimissään mukaan.

Tässä diplomityössä tutustutaan kahteen eri mobiili-Java-ympäristöön, MIDP:hen ja Androidiin, ja tutkitaan siirrettävyyttä näiden välillä. Työssä tarkastellaan ympäristöjen eroja yleisesti ja rajapintatasolla ohjelmoijan näkökulmasta. Konkreettisena työnä siirretään Teksti-TV-sovellus MIDP:stä Androidille. Lisäksi siirtotyön määrää yleensä arvioidaan lyhyesti kokemuksiin perustuen.

Diplomityössä käy selväksi, että vaikka molemmat ympäristöt ovat nimellisesti Java-ympäristöjä, ne ovat monelta osin hyvin erilaisia, varsinkin kun tarkastellaan toteutuksen yksityiskohtia. Siksi siirtäminen ei onnistu mekaanisesti. Teksti-TV-sovellus saatiin siirrettyä, mutta se vaati myös paljon uudelleenkirjoittamista. Käytännössä vain ne osat ohjelmaa, jotka eivät käytä mitään ulkopuolisia rajapintoja, saatiin siirrettyä ilman isoja muutoksia.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
PALKONEN, JYRKI: Porting Teksti-TV application from MIDP to Android
Master of Science Thesis, 44 pages, 2 Appendix pages
April 2012
Major: Distributed systems
Examiner: Professor Tommi Mikkonen
Keywords: Android, MIDP, portability, mobile

Over the years a software engineer collects lots of old applications and software components that could be used in new projects. High level programming languages promise easy software portability from one environment to another even without recompiling. However, reusing old code is not necessarily easy, especially if the implementation was not done with reuse in mind. In some cases it might be easier to start from scratch instead of trying to figure out what the old code does and which parts of it are necessary.

This thesis introduces two different mobile Java environments, MIDP and Android, and examines portability between them. Thesis discusses the differences in general and on the API level from the programmer's point of view. As a practical implementation Teksti-TV application is ported from MIDP to Android. Based on the gained experiences, a generic estimate on portability between the platforms is made.

Thesis makes it clear that even though both MIDP and Android platforms use Java, they are very different in many parts, especially when considering the details of the implementation. Therefore, porting between platforms cannot be done mechanically. Teksti-TV application was eventually ported, but this required writing a lot of new code, so that only the core parts that do not use any outside interfaces were ported without major modifications.

ALKUSANAT

"A jug fills drop by drop."
-Buddha

Kiitokset vaimolleni Jeerathayalle ja pojilleni Jimille ja Jarille.
Kiitos myös ohjaajilleni Tommi Mikkoselle ja Mikael Rinnetmäelle.

Hervannassa 15.3.2012
Jyrki Palkonen

SISÄLLYS

1 Johdanto 1
2 Siirrettävyys 3
 2.1 Ohjelmien siirtäminen 3
 2.2 Uudelleenkäyttö 5
 2.3 Siirtoon liittyviä ongelmia 5
 2.4 Siirtämisen ongelmat Javassa 8
 2.5 Siirrettävyys mobiiliympäristössä 8
3 Mobiili Java 10
 3.1 Java Micro Edition ja MIDP 10
 3.2 Android 11
  3.2.1 Sovellukset 12
  3.2.2 Sovelluskehys 13
  3.2.3 Kirjastot 14
  3.2.4 Ajonaikainen ympäristö 14
  3.2.5 Linux-ydin 14
4 Erot ohjelmoijan kannalta 16
 4.1 Sovelluksen elinkaari 16
 4.2 Sovellusten rakentaminen 18
 4.3 Sovelluksen julkaiseminen 21
 4.4 Rajapinnat 22
  4.4.1 Käyttöliittymä 22
  4.4.2 Multimedia 23
  4.4.3 Yhdistyvyys 24
  4.4.4 Siirräntä 24
  4.4.5 Työkalut 25
 4.5 Testaus 26
5 Teksti-TV-sovelluksen siirtäminen 27
 5.1 Olemassaoleva Teksti-TV-sovellus 27
 5.2 Vaatimukset 28
 5.3 Siirtämisen vaiheet 28
 5.4 Miten siirto meni 29
 5.5 Millainen MIDP oli 30
  5.5.1 Jad-tiedosto 30
  5.5.2 Elinkaari 30
  5.5.3 Käyttöliittymä 31
  5.5.4 Tiedostojen lukeminen 33
  5.5.5 HTTP-yhteydet 34
  5.5.6 Säikeet 34
 5.6 Millainen Android on 35
  5.6.1 AndroidManifest.xml 35

5.6.2 Elinkaari 36
5.6.3 Käyttöliittymä 36
5.6.4 Tiedostojen lukeminen 39
5.6.5 HTTP-yhteydet 40
5.6.6 Säikeet 41
6 Johtopäätökset 43
Lähteet 45
Liitteet 49

1 JOHDANTO

Ohjelmistojen siirtämisellä voidaan saavuttaa taloudellisia säästöjä, jos voidaan säästää aikaa ohjelmiston suunnittelussa, toteutuksessa ja testaamisessa käyttämällä jo olemassa olevan ohjelmiston lähdekoodia hyväksi. Erityisesti korkean tason ohjelmointikielissä lupaavat ohjelmoijalle helppoa siirrettävyyttä eri järjestelmien välillä. Yksi näistä kielistä on Sun Microsystemsin vuonna 1996 julkaisema Java.

Java on laitteistoriippumaton ohjelmointikieli ja ajoympäristö. Ajoympäristöstä on kolme versiota: Standard Edition (Java SE) [1], Enterprise Edition (Java EE) [2] ja Micro Edition (Java ME) [3]. Java SE on tarkoitettu työasema-sovellusten ajamiseen, Java EE on tarkoitettu palvelinsovellusten ajamiseen ja Java ME resursseiltaan pienissä laitteissa, kuten puhelimissa ja digibokseissa, ajettaviin sovelluksiin. Java lähdekoodit käännetään tavukoodiksi, jota sitten ajetaan Java virtuaalikoneessa. Java ME:ssä käytetään Java virtuaalikoneen asemesta rajoitetumpaa KVM (K Virtual Machine) [4] virtuaalikonetta, josta puuttuu joitain ominaisuuksia, kuten reflektio, eli kyky yhdistää luokka tai metodi nimen perusteella oikeaan kohteeseen. Java ME on edelleen jaettu tarkemmin määriteltyihin profiileihin, joista tässä käsitellään MIDP 2.0 –profiilia (Mobile Information Device Profile) [5], joka on tarkoitettu sulautetuille laitteille, kuten puhelimille.

Android on Open Handset Alliancen [6], jota Google [7] johtaa, käyttöjärjestelmä, joka on suunnattu mobiililaitteille, kuten puhelimille ja taulutietokoneille. Android sovellukset tehdään Javalla, mutta Androidissa ei ole mitään Java-profiilia kokonaan tuettuna, vaan sen käyttämä Dalvik-virtuaalikone on yhdistelmä Java SE:sta, avoimen lähdekoodin kirjastoista ja Androidin omista rajapinnoista. Olemassa olevia Java ME-sovelluksia ei siis voi ajaa suoraan Android-järjestelmässä, vaan niitä pitää muokata käyttämään Androidin tarjoamia rajapintoja.

Tässä diplomityössä tutkitaan ohjelmien siirrettävyyttä yleisesti ja sitten tarkemmin kahden eri mobiili-Java ympäristön välillä ja siirtoon liittyviä käytännön ongelmia ohjelmoijan näkökulmasta. Lisäksi esitellään molemmat ympäristöt ja käydään läpi niiden tarjoamia rajapintoja. Konkreettisena esimerkkinä esitellään Teksti-TV-sovelluksen [8] siirtäminen MIDP:stä Android-ympäristöön.

Luvussa kaksi tarkastellaan siirrettävyyttä yleisesti ja erityisesti mobiili-Java-ympäristössä. Luvun kolme tarkoituksena on käydä läpi molemmat käsiteltävät mobiiliympäristöt. Luvussa neljä selvitellään ympäristöjen eroja ohjelmoijan näkökulmasta. Luvussa viisi esitellään käsiteltävänä oleva Teksti-TV sovellus, minkä jälkeen käydään läpi siirron vaiheet ja miten siirto meni ja millaisilta molemmat ympäristöt näyttivät

tämän sovelluksen näkökulmasta. Lopuksi luvussa kuusi tehdään johtopäätökset siirrettävyydestä kahden eri Java-ympäristön välillä.

2 SIIRRETTÄVYYS

Tässä luvussa käsitellään ohjelmien siirrettävyyttä yleisesti. Aluksi tarkastellaan siirtämisen historiaa ohjelmointikielien kehityksen näkökulmasta. Toiseksi esitellään aihetta sivuavaa ohjelmien uudelleenkäyttöä, kolmantena käsitellään siirtämiseen liittyviä ongelmia, jonka jälkeen tarkastellaan Javaa erityisesti ja lopuksi käydään läpi Javan siirrettävyys mobiiliympäristössä.

Ohjelmistoyksikkö on siirrettävä, jos uuteen ympäristöön siirron ja muokkauksen kustannukset ovat pienemmät kuin uudelleenkehityksen [9].

Ohjelmistoyksiköllä tarkoitetaan sovellusohjelmaa, systeemiohjelmaa tai ohjelmistokomponenttia. Ympäristöllä tarkoitetaan alustaa, jossa ohjelmistoa ajetaan. Siihen kuuluu prosessori ja käyttöjärjestelmä, ja yleensä siirräntälaitteita, kirjastoja ja tietoverkkoja.

Parhaimmillaan siirrettävä ohjelma:

- on ajettavissa eri toimittajien erilaisilla alustoilla
- toimii heterogeenisessä verkkolaskennassa samoin kuin perinteisessä moniprosessorijärjestelmässä
- on kehitettävissä hajautetusti kätevimmällä alustalla
- sallii käyttäjien jakaa lähdekoodia, sitä muuttamatta
- tarjoaa ylöspäin yhteensopivuutta, kun rinnakkaiset järjestelmät muuttuvat [10].

2.1 Ohjelmien siirtäminen

Tietokoneiden kehitys on ollut nopeaa ja eri laitteistoja ja käyttöjärjestelmiä on ollut (ja on yhä) paljon. Tästä seuraa, että jos ohjelmiston siirtäminen on helppoa, siirtämisellä säästetään paljon aikaa, vaivaa ja kustannuksia. 1950-luvulla kehitettiin ensimmäiset modernit ohjelmointikielet: Fortran, Lisp ja Cobol. Nämä kielet eivät olleet sidottuja tiettyyn laitteistoarkkitehtuuriin, kuten aikaisemmat assembler-kielet, jolloin samasta lähdekoodista voitiin periaatteessa kääntää versio aina uudelle alustalle, mikäli alustalla oli kääntäjä käytetylle kielelle.

Nykyään ohjelmointikieliä ja niiden murteita on satoja ja olemassa olevaa lähdekoodia todella paljon. Vaikka monet kielet lupaavat täydellistä siirrettävyyttä esimerkiksi ”write once — run anywhere”, joka mainitaan monessa paikkaa Oraclen sivustolla Javaan viitaten [11], lupauksia ei aina lunasteta käytännössä. Tämä voi johtua siitä, että ajoym-

päristöt ovat niin erilaisia, ettei siirto ole mahdollista, erilaisista resursseista kuten muistin määrästä, suorittimen tehosta ja näytön koosta johtuen.

Laskentatehon kasvaessa myös ohjelmointikielten ominaisuudet ovat kasvaneet ja monimutkaistuneet, taulukossa 2.1 on kuvattu ohjelmointikielten kehitysasteita. Ylemmän abstraktiotason toteutus vaatii enemmän tehoa laitteistolta, joko kääntämisen tai ajon aikana, joten on loogista, että mitä vähemmän resursseja on käytössä, sitä matalammalla abstraktiotasolla yleensä toimitaan.

Taulukko 2.1: Ohjelmistokielten abstraktiotasot ja esimerkkikieliä.

Abstraktiotaso	**Esimerkkikieliä**
1. Suora laitteiston ohjaaminen	Konekieli (00110001) Assembler
2. Käännettävät kielet	C ALGOL Fortran C++ Cobol Eiffel
3. Tulkattavat ja virtuaalikoneessa ajettavat kielet	Lähdekoodina suoritettavat: Python JavaScript
	Esikäännetyt: ActionScript Java

Ensimmäisellä abstraktiotasolla ohjelmat syötettiin aluksi suoraan binäärikoodina prosessorin ymmärtämässä muodossa. Tämä on hyvin virheherkkää, koska kaikki täytyy olla täsmälleen oikein, eikä virhe paljastu muuten kuin ajon aikana, eikä välttämättä silloinkaan, jos esimerkiksi konekäsky on muuttunut toiseksi lailliseksi konekäskyksi. Symbolinen konekieli eli assembler tarjoaa mahdollisuuden tarkistaa syntaksin laillisuus ennen ajoa ja sen kirjoittaminen on ihmiselle paljon helpompaa kuin binäärikoodin. Edelleen kuitenkin pääsääntöisesti yksi assembler-lause vastaa täsmälleen yhtä konekäskyä, joten ohjelmoija spesifioi täsmälleen, mitä käskyjä ja missä järjestyksessä prosessorin tulee suorittaa.

Toisella abstraktiotasolla ohjelmoija kirjoittaa koodia, jonka kielen kääntäjä kääntää konekieliseksi ajettavaksi binäärikoodiksi. Tämä mahdollistaa abstraktiotason noston, sillä kielen määrittelyssä voidaan määritellä rakenteita, jotka ovat työläitä käsin toteuttaa aina itse uudestaan. Ohjelmoija voi käyttää kielen tarjoamia palveluita ja jättää to-

teutuksen yksityiskohdat kääntäjän vastuulle. Nyt yhdestä kielen lauseesta voi syntyä nolla tai enemmän konekäskyä.

Kolmannella abstraktiotasolla lähdekoodia ajetaan lause kerrallaan tulkin avulla tai se käännetään johonkin tavukoodimuotoon, jota ajetaan virtuaalikoneessa. Tulkkia voi myös pitää virtuaalikoneena, jolloin lähdekoodi itse on tavukoodia. Isoimpana erona on, että tulkattavissa kielissä lähdekoodille ei usein ajeta syntaksitarkastusta ennen suorituksen aloittamista. Toinen iso ero toiseen abstraktiotasoon verrattuna on, että nämä kielet voivat tarjota myös graafisen käyttöliittymän abstraktion, jolloin ohjelmoijan ei tarvitse tietää, missä ympäristössä, missä käyttöjärjestelmässä ohjelmistoa tullaan ajamaan, vaan riittää että käytettävissä on kielen virtuaalikone.

2.2 Uudelleenkäyttö

Siirrettävyyteen liittyy läheisesti ohjelman osien ja komponenttien uudelleenkäyttö. Tällöin ohjelmaa ei siirretä kokonaan, vaan siitä otetaan joitain pienempiä kokonaisuuksia ja käytetään niitä toisaalla. Nämä kokonaisuudet voivat olla muutakin kuin lähdekoodia; esimerkiksi testaussuunnitelmasta tai vaatimusmäärittelystä voidaan ottaa soveltuvat osat.

Krueger [12] listaa erilaisia uudelleenkäytön kohteita, joista tähän työhön liittyy eniten uudelleenkäyttö suunnittelussa ja koodin pengonta (Reuse in Design and Code Scavenging). Siinä ohjelmoija huomaa, että uudessa sovelluksessa on samantapainen osio kuin jossain aiemmin toteutetussa sovelluksessa, jolloin vanhasta sovelluksesta voi penkoa käyttöön joitain paloja. Näin ohjelmoija voi uudelleenkäyttää koodia, sekä mahdollisesti vanhan sovelluksen suunnitteluratkaisuja liittyen koodin käyttöön. Uudelleenkäytettävän koodin toiminta täytyy tuntea hyvin, jotta sitä voi soveltaa mahdollisesti aivan erilaisessa uudessa ympäristössä. Parhaassa tapauksessa suuriakin määriä vanhaa koodia voidaan siirtää pienin muutoksin uuteen ympäristöön tai vastaavasti huonoimmassa tapauksessa ohjelmoija käyttää enemmän aikaa koodin toiminnan selvittämiseen, virheiden jäljittämiseen ja eri osien etsimiseen, kuin mitä menisi saman asian toteuttamiseen tyhjästä alkaen.

2.3 Siirtoon liittyviä ongelmia

Ideaalitapauksessa siirtäminen onnistuu binäärien tai virtuaalikoneen tavukoodin kopioimisella tai lähdekoodin kääntämisellä kohdelaitteistossa. Vaikeimmillaan lähteen ja kohteen välillä ei ole fyysistä tiedonsiirtotapaa, jolla lähdekoodia voidaan siirtää järjestelmästä toiseen. Kääntämisen epäonnistumiseen uudessa ympäristössä voi olla useita syitä; käyttöjärjestelmä, käyttöliittymä, erot kääntäjissä, tiedostojärjestelmä, ynnä muita.

Eri käyttöjärjestelmissä ja saman käyttöjärjestelmän eri versioissa ja configuraatioissa asiat toimivat eri lailla, esimerkiksi muistin suojaus ja säikeiden kontrollointi voivat erota paljon kahden eri käyttöjärjestelmän välillä. Aikaisemmin 80- ja 90-luvuilla saattoi olla mahdollista lukea muistia mistä hyvänsä ja jopa toteuttaa itseään muuttavaa koodia. Tämä luonnollisesti olettaa paljon alla olevasta järjestelmästä ja laitteistosta, joten oletuksena siirtotyö vaatii paljon vaivaa. Nykyään tällaiset temput ovat harvoin tarpeen. Sen sijaan säikeitä ja rinnakkaisuutta käytetään koko ajan enemmän. Säikeiden toteutuksia on myös erilaisia: säikeitä (thread), prosesseja (process), kevyitä säikeitä (fiber). Näiden eroina on ainakin: miten ne käynnistetään, mistä suoritus jatkuu ja mitä kaikkea dataa ne näkevät tai perivät käynnistäjä-säikeeltä.

Graafinen käyttöliittymä on vaikea toteuttaa siirrettävästi, koska eri käyttöjärjestelmät ja ikkunointijärjestelmät eivät yleensä ole yhteensopivia. Esimerkiksi C++ -kielisessä ohjelmassa Microsoft Windows-käyttöjärjestelmään tarvitaan include-tiedostoja, joita ei ole olemassa, kun samaa ohjelmaa yritetään siirtää Linuxille. Ratkaisuna usein on käyttää jotain kirjastoa, kuten wxWidgets [13], joka tarjoaa rajapinnan alustariippumattomalle käyttöliittymälle. Tällöin ongelmaksi tulee riippuvuus käytettyyn kirjastoon, jota ei välttämättä löydy kaikille halutuille alustoille. Toisaalta voi ajatella, että käyttöliittymän siirtäminen ei välttämättä ole edes kovin järkevää, koska eri ympäristöissä on erilaiset tavat toimia, joten siirretty käyttöliittymä voi tuntua todella luonnottomalta toisessa ympäristössä.

Eri valmistajien kääntäjät ja saman kääntäjän eri versiot generoivat koodia yleensä eri tavalla ja noudattavat kielen standardeja eri tavoilla. Siksi siirrettävyyden kannalta ei kannata käyttää kielten uusimpia ominaisuuksia tai jonkin tietyn kääntäjän tarjoamia ominaisuuksia. Käännettäessä kannattaa kytkeä kääntäjän kaikki varoitukset päälle, jotta riskialttiit osat paljastuvat mahdollisimman varhaisessa vaiheessa. Lisäksi kehityksen aikana kannattaa säännöllisesti kääntää eri ympäristöissä vikojen paljastamiseksi.

Tiedostojärjestelmät ja mihin kohtaan tiedostojärjestelmää käyttäjät saavat lukea ja kirjoittaa poikkeavat huomattavasti. Esimerkiksi Microsoft Windowsin C:\-polku ei tarkoita Linuxissa mitään erityistä, vaan kaikki merkit tulkitaan tiedostonnimeksi. Näin myös ohjelmien asetus ja konfiguraatiotiedostojen paikat vaihtelevat eri järjestelmissä, sekä yleisesti käytetty tapa näiden esittämiseen; Windowsin rekisteri tai Linuxin pisteellä alkavat tiedostojen nimet.

Turvallisuus liittyy käyttöjärjestelmän sallimiin toimintoihin ja toisaalta ohjelmointikielen sallimiin toimintoihin. Käyttöjärjestelmät eroavat sen suhteen, mitä käyttäjä ja käyttäjän käynnistämä ohjelma saavat tehdä ja millä oikeuksilla. Ohjelmalla ei välttämättä ole oikeuksia kuin omaan käynnistyshakemistonsa lukemiseen. Toisaalta, jos ohjelma on käynnistetty ylläpitäjän oikeuksilla, ”mitä tahansa” voi olla mahdollista tehdä. Esimerkkinä mainittakoon vaikkapa jokin haittaohjelma: jos käyttöjärjestelmä sallii

kaikkien tiedostojärjestelmän hakemistojen vapaan selaamisen, on paljon helpompi saastuttaa muita ohjelmia, kuin jos käyttöjärjestelmä sallii vain kotihakemiston vapaan käyttämisen. Ohjelmointikielissä turvallisuuseroja tuo esimerkiksi indeksien tarkistus, kun osoitetaan jotain taulukon alkiota. Jos tarkistuksia ei ole, voi syntyä tilanteita, joissa sopivasti taulukon ohi kirjoittamalla ja lukemalla päästään suorittamaan jotain ohjelmakoodia, jota ei alkuperäisessä ohjelmassa ole ollut, vaan joka on ujutettu sisään vaikka jonkin ohjelmistovirheen avulla. Erityisesti jos siirrettävä ohjelma on kirjoitettu kielellä, joka sisältää indeksitarkistukset, ja kohdekieli ei sisällä, voi tulla myöhemmin ongelmia, koska asiaan ei ole välttämättä kiinnitetty huomiota, vaan on luotettu alkuperäisen kielen turvallisuusmekanismeihin.

Rajapinnat ja kirjastot ovat tyypillisesti käyttöjärjestelmissä erilaisia ja siirto vaatii vähintään otsikkotiedostojen muutoksen. Usein joudutaan muutakin koodia muuttamaan, varsinkin jos tarkoitus on dynaamisesti ladata jotain kirjastoja, jolloin yleensä joudutaan tekemään latausrutiini jokaiselle eri käyttöjärjestelmälle, koska latausmekanismit ovat erilaisia ja kirjastojen nimeämiskäytännöt (Windowsin .dll- ja Linuxin .so -kirjastot esimerkkinä).

Lopulta myös fyysinen laitteisto on erilaista, prosessoreja on erilaisia, muistia on eri määriä, erilaisia oheislaitteita on paljon. Lisäksi laitteisto voi tarjota laitteistokiihdytystä, jolloin suuri osa laskennasta siirretään pois prosessorilta. Pahimmassa tapauksessa ohjelma muuttuu täysin käyttökelvottomaksi, jos se luottaa kiihdytykseen, jota ei sitten olekaan. Toisaalta, jos ohjelma on toteutettu hyvin, sen toiminta paranee ”automaattisesti”, jos käytetty alusta tarjoaa laitteistokiihdytystä, esimerkiksi äänen tai videon käsittelyssä. 3D-grafiikassa tästä on hyvänä esimerkkinä OpenGL [14], joka takaa, että ohjelma toimii aina, vaikka laitteisto ei tukisi jotain ominaisuutta, tällöin vain kyseinen ominaisuus toteutetaan ohjelmallisesti, oli se sitten kuinka hidasta tahansa.

Yleinen ratkaisu kaikkeen on nostaa abstraktiotasoa ja muodostaa rajapintoja, joiden kautta asioita tehdään, ja joiden toteutus on erilainen eri alustoilla. Tähän tarkoitukseen on olemassa jo erilaisia kirjastoja, kuten esimerkiksi edellä mainittu wxWindows, mutta riippuvuudet kirjastoista heikentävät siirrettävyyttä.

Nykyaikainen ratkaisu on käyttää erilaisia virtuaalikoneita, jolloin lähdekoodi käännetään ensin tavukoodiksi tai lähdekoodia ajetaan ja tulkataan suoraan. Näin abstrahointi voidaan jättää kielen tulkin tai virtuaalikoneen ratkaistavaksi. Javan ratkaisu on kääntää lähdekoodi tavukoodiksi, jota sitten virtuaalikone tulkkaa ja ajaa; etuna on tehonlisäys, koska koodia ei tarvitse jäsentää ajon aikana ja syntaksivirheet paljastuvat jo käännösvaiheessa. Lisäksi tavukoodi voidaan siirtää järjestelmästä toiseen ja se toimii aina samalla tavalla.

2.4 Siirtämisen ongelmat Javassa

Javan ideana on, että sovellus kirjoitetaan ja käännetään kerran, ja sen jälkeen sitä voi ajaa missä vain. Käytännössä asia ei toimi ihan näin helposti, vaan ongelmia aiheuttavat muun muassa eri Java-versiot, erilaiset virtuaalikoneet, alustan fyysiset ominaisuudet ja alustalla kulloinkin käytettävissä olevat kirjastot.

Tällä hetkellä Javan (standard edition) uusin versio on Java SE 7 [15]. Java ajoajan ympäristöön sisältyy virtuaalikone, jolla tavukoodiksi käännetyt ohjelmat ajetaan, ja virtuaalikone osaa ajaa vanhempia versioita vastaan käännettyjä ohjelmia. Versioiden vaihtuessa tavukoodin formaatti on myös vaihtunut ja näin vanhemmat virtuaalikoneet eivät osaa ajaa uudemmassa muodossa olevia ohjelmia, vaikka niissä ei käytettäisi uudempia kielen ominaisuuksia. Vastaavasti Oraclen javac-kääntäjä ei osaa kääntää uudempia ohjelmia siten, että ei-tuetut ominaisuudet jätettäisiin pois. Esimerkiksi Java SE 1.4 -versiossa [16] tuli tuki assertioille, mutta kääntäjä ei osaa kääntää Java SE 1.3 -yhteensopivaan [17] muotoon vain jättämällä assertiot huomioimatta.

Virtuaalikoneita on erilaisia, erityisesti mobiililaitteissa, joten eroja voi löytyä eri valmistajien toteutuksissa. Isoimmat erot löytyvät kuitenkin siitä, mille rajapinnoille (JSR, Java Specification Request) alustalta löytyy toteutus. Näissä standardirajapinnoissa on määritelty esimerkiksi, miten OpenGL:ää [18] tai SVG-grafiikkaa [19] käytetään Javassa tai miten jotain oheislaitetta ohjataan. Jos kohdealustalta ei löydy tarvittavaa rajapintaa, niin asialle ei useimmiten voi tehdä mitään, vaan siirtämisestä pitää luopua.

Vaikka Java-virtuaalikone abstrahoi alla olevan laitteiston ja käyttöjärjestelmän pois, käytännössä muistin määrä ja prosessorin nopeus voi aiheuttaa sen, että siirretty ohjelma ei ole käyttökelpoinen.

2.5 Siirrettävyys mobiiliympäristössä

Mobiileja laitteita on monia erilaisia. Muistin määrä vaihtelee, ruudun koko, prosessorin teho ja käskykanta, ja muut ominaisuudet, kuten GPS [20] ja Bluetooth [21]. Käännettävä kieli on huonosti siirrettävä ratkaisu, koska ohjelma pitäisi aina kääntää joka laitteelle erikseen tai vähintään joka eri valmistajan laitteille erikseen. Luonteva ratkaisu onkin käyttää virtuaalikonetta ja Java tarjoaa siihen yhden ratkaisun; kerran tavukoodiksi käännetty ohjelma voidaan ajaa missä tahansa laitteessa, jossa on virtuaalikone.

Javan mobiiliratkaisu MIDP 1.0 [22] julkaistiin vuonna 2000. Se oli suunnattu laitteille, joilla oli minimissään 96x54 pikselin kaksivärinen näyttö, 128 kilotavua pysyväismuistia MIDP komponenteille, 8 kilotavua pysyväismuistia sovellusten kirjoittamalle datalle ja 32 kilotavua suorasaantimuistia Javan ajo-ajalle, sekä langaton verkkoyhteys, joka voi toimia puuskittain. MIDP 2.0 [5] julkaistiin vuonna 2002 ja siinä on minimivaati-

muksia nostettu pysyväismuistin osalta 256 kilotavuun ja ajo-ajan muisti 128 kilotavuun. Lisäksi vaaditaan että laite osaa soittaa ääniä (tones). MIDP 2.0 -laitteiden tulee myös olla taaksepäin yhteensopivia 1.0 -määritelmän kanssa, joten vanhemmat toteutukset toimivat uudessa.

Toinen laajasti mobiililaitteissa käytetty Java-ympäristö Android 1.0 julkaistiin 2008. Siinä ei ole virallisia minimivaatimuksia laitteistolle, mutta pääsuunnittelija Andy Rubin on sanonut, että minimivaatimukset ovat 32 megatavua suorasaantimuistia, 32 megatavua pysyväismuistia ja vähintään 200 MHz prosessori [23]. Ominaisuudet ovat moninkertaiset MIDP:iin verrattuna, ja vaikka Androidiin kehitetään sovelluksia Javalla, tavukoodi ei ole yhteensopivaa eri virtuaalikoneen vuoksi. Lisäksi Androidin rajapinnat ovat erilaiset kuin MIDP:ssä, kuten seuraavissa luvuissa nähdään, joten siirtäminen Android-Javan ja MIDP-Javan välillä ei onnistu.

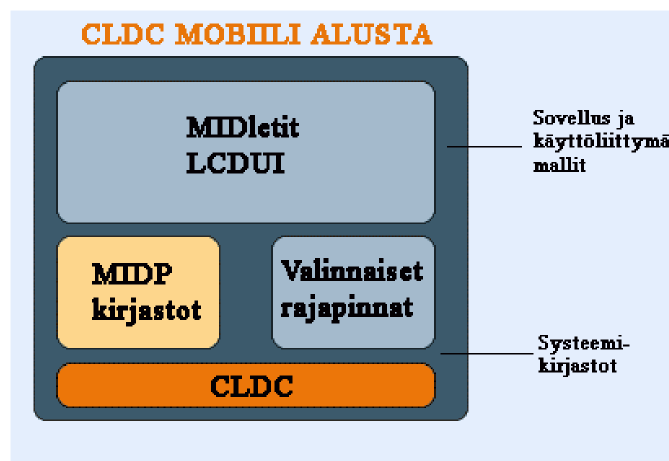
Lisäksi Research In Motionin BlackBerry-käyttöjärjestelmässä on myös Java-virtuaalikone, joka pystyy ajamaan MIDP 2.0 -sovelluksia [24]. Lisäksi Java-toteutus sisältää paljon muita rajapintoja, joita ei löydy Androidista tai MIDP:stä, joten siirrettävyys BlackBerryssä toimii vain MIDP:stä BlackBerryyn. Tässä työssä ei käsitellä BlackBerryä.

3 MOBIILI JAVA

Tässä luvussa käydään läpi käsiteltävät mobiiliympäristöt; MIDP ja Android. Molemmista esitellään niiden yleisarkkitehtuuri ja esitellään arkkitehtuuriin kuuluvat osat.

3.1 Java Micro Edition ja MIDP

Java ME suunniteltiin alun perin resursseiltaan pienien laitteiden käyttöön. Näissä laitteissa on mahdollisesti vähän muistia, pieniresoluutioinen näyttö tai vähän laskentatehoa tai yleensä kaikki nämä yhdessä. Ajan myötä Java ME on jakaantunut kahteen eri konfiguraatioon; pienten laitteiden Connected Limited Device Configuration (CLDC) ja vähän tehokkaampien laitteiden Connected Device Configuration (CDC). Näistä ensimmäinen löytyy esimerkiksi puhelimista ja jälkimmäistä DVB-MHP-digitv-vastaanottimista ja Blu-ray-soittimista. Konfiguraatioiden lisäksi Java ME alusta määrittelee joukon profiileja, jotka määrittelevät joukon korkean tason rajapintoja. Esimerkiksi CLDC yhdistettynä Mobile Information Device Profile (MIDP) profiiliin muodostaa Java sovellusympäristön kännyköille ja muille laitteille, joilla on samanlaiset ominaisuudet.



Kuva 3.1. *CLDC konfiguraatio [25].*

CLDC-kirjastot sisältävät pienen joukon peruskirjastoja [26] ja tarvittavat virtuaalikoneen ominaisuudet. Tämän päälle tulee MIDP-profiilista MIDP-kirjastot, kuten kuvassa 3.1, jotka sisältävät sovellus- (MIDlet) ja käyttöliittymäkirjastot (LCDUI). Näiden lisäksi laitteessa voi olla valinnaisia kirjastoja (JSR) ja päätelaitteen valmistajan omia yksityisiä kirjastoja.

Kaikki MIDP-sovellukset perivät Midlet-luokan ja siten kaikilla sovelluksilla on samanlainen elinkaari, eli keskeytynyt, käynnistynyt ja tuhoutunut. LCDUI (Limited Capability Device User Interface, suppeakapasiteettisen laitteen käyttöliittymä) tarjoaa joukon peruskäyttöliittymäkomponentteja, joilla voi tehdä käyttöliittymiä, mutta joiden ulkoasuun ja asetteluun ei voi juurikaan vaikuttaa.

Valinnaiset kirjastot liittyvät johonkin ominaisuuteen tai toiminnallisuuteen, jota päätelaite tukee. Esimerkkinä voidaan mainita 3D-rajapinnat, jotka löytyvät JSR-184-kirjastosta [27]. Näiden lisäksi valmistajalla voi olla omia rajapintoja päätelaitteen ohjaamiseen; niiden käyttö estää tai ainakin vaikeuttaa sovelluksen siirtämistä toisen valmistajan päätelaitteeseen tai jopa saman valmistajan eri mallin päätelaitteeseen [28].

Sovellukset pakataan Java Development Kitin mukana tulevalla Jar-ohjelmalla .jar-paketiksi, joka on käytännössä zip-tiedosto. Sen lisäksi tarvitaan .jad-tiedosto, joka on tekstimuotoinen kuvaus sovelluksesta [29]. Jad-tiedoston avulla voidaan päätelaitteelle kuvata, mitä rajapintoja sovellus vaatii, jolloin päätelaite voi estää sovelluksen asentamisen, mikäli se ei tue joitain sovelluksen vaatimia ominaisuuksia. Ennen paketoimista sovellukseen kuuluvat luokat ajetaan erillisen todentimen (class file verifier) läpi, minkä tehtävänä on tarkistaa, että käännetyt tavukoodit eivät sisällä ei-sallittuja käskyjä, koodi ei ole ei-sallitussa järjestyksessä, eikä sisällä viittauksia epäkelpoihin muistiosoituksiin tai muistialueiseen, jotka ovat olioille varatun alueen ulkopuolella. Todennin myös muokkaa hyppykäskyjä ja lisää tiettyjä attribuutteja päätelaitteessa olevaa todenninta varten [26]. Päätelaitteessa oleva todennin käy vielä tavukoodin läpi käsky käskyltä tehden muuan muuassa tyyppimuunnoksia, jos oikean tyypin pystyy päättelemään operandista. Lisäksi paikallisille muuttujille varataan tila tässä vaiheessa [26].

3.2 Android

Android on ohjelmistopino mobiililaitteille, mikä sisältää Linux 2.6 ytimeen perustuvan käyttöjärjestelmän, välikerroksen ja tärkeimpiä sovelluksia. Android on suurimmaksi osaksi avointa lähdekoodia [30], joten laitevalmistaja voi halutessaan tehdä tarvittavia muutoksia tai korjauksia käyttämäänsä Android-versioon. Kuvassa 3.2 on kuvattu Androidin kerrosarkkitehtuuri, jossa ylemmältä tasolta kutsutaan aina alemman tason palveluita.

Kuva 3.2. *Androidin arkkitehtuuri [31].*

3.2.1 Sovellukset

Androidin mukana tulee tärkeimpiä sovelluksia, kuten selain, sähköposti, SMS-sovellus ja muita. Kaikki sovellukset tehdään Java-ohjelmointikielellä. Javasta ei kuitenkaan ole otettu mukaan mitään tiettyä versiota (kuten Java SE 1.3), vaan eri versioista on otettu eri osia mukaan. Sovellukset pakataan Android-paketiksi (Android package) aapt-työkalulla ja paketin sisään tulee erityinen manifesti-tiedosto, joka vastaa MIDP:n jad-tiedostoa, eli siellä kerrotaan sovelluksen nimi ja annetaan muuta tietoa sovelluksesta.

Sovellukset voivat olla neljää eri tyyppiä: activity, service, broadcast receiver tai content provider. Yleisin sovellustyyppi on activity, sillä niille on käyttäjälle näkyvä käyttöliittymä.

Activity tarkoittaa sovellusta, joka tarjoaa visuaalisen käyttöliittymän käyttäjälle. Jokainen activity saa oman oletusikkunan, jonne se voi piirtää. Yleensä ikkuna on ruudun kokoinen, mutta se voi olla myös pienempi ja olla osittain toisten ikkunoiden päällä tai alla. Sovellus voi myös koostua useista activityistä, joista jokin on merkattu ensimmäiseksi, joka näytetään, kun sovellus käynnistetään.

Service on sovellus, joka on ajossa taustalla ja jolla ei ole mitään käyttöliittymää. Esimerkiksi jokin herätyskellotyyppinen sovellus, joka nukkuu taustalla, ja kun sopiva hetki tulee, soittaa herätysäänen tai käynnistää jonkin activityn.

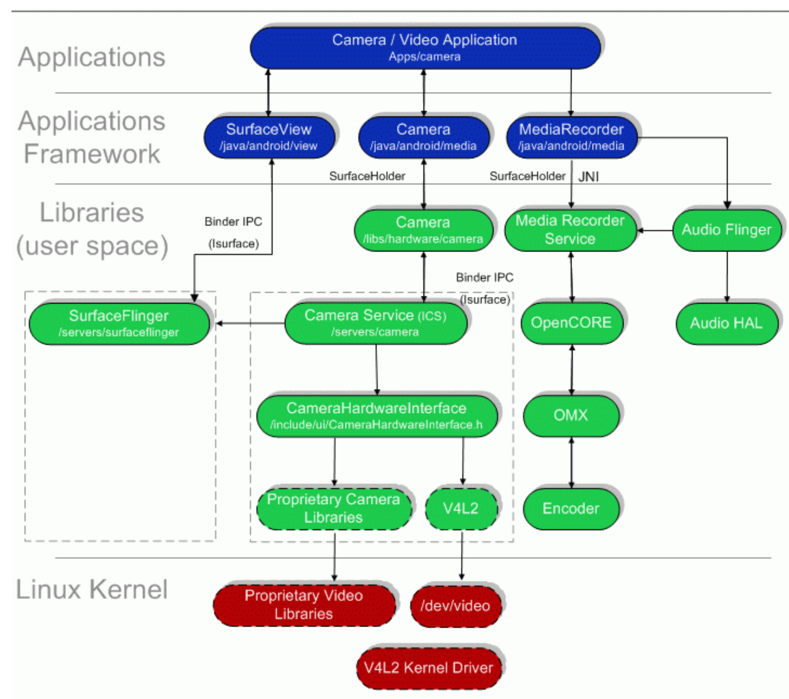
Broadcast receiver on käyttöliittymätön sovellus, joka vain kuuntelee ja reagoi yleislähetyksiin (broadcast), vaikka ilmoitukseen, että akun varaus on alhainen. Sovellukset voivat myös itse tehdä yleislähetyksiä, esimerkiksi ilmoittaa, että jokin laskentatehtävä on valmis.

Content provider-sovellus tarjoaa jotain sisältöä muille sovelluksille ja siten abstrahoi varsinaisen sisällön lähteen. Sisältö voi olla, ääntä, videota tai mitä tahansa, mitä halutaan välittää prosessilta toiselle.

3.2.2 Sovelluskehys

Sovellusarkkitehtuuri on suunniteltu yksinkertaistamaan komponenttien uudelleenkäyttöä ja mikä tahansa sovellus voi julkaista omat palvelunsa (capabilities) muiden sovellusten käyttöön. Samalla mekanismilla myös käyttäjä voi valita, mitä komponentteja hän haluaa käyttää, eli hän voi sitä kautta muokata sovellusten ulkonäköä tai toiminallisuutta.

Kehittäjillä on täysi pääsy sovelluskehyksen rajapintoihin. Androidin mukana tulevat sovellukset käyttävät samoja rajapintoja kuin kaikki muutkin sovellukset. Ne siis eivät käytä mitään valmistajan dokumentoimattomia rajapintoja.



Kuva 3.3. *Esimerkkikutsuhierarkia kamera-sovelluksesta [32].*

Kuvassa 3.3 näkyy, kuinka sovelluksesta tulee kutsu sovelluskehykseen (alempi sininen taso), josta se menee Javan natiivi-rajapinnan (JNI, Java Native Interface) läpi kirjastotasolle (vihreällä) ja sieltä edelleen Linuxin ytimeen (punaisella).

3.2.3 Kirjastot

Androidissa on joukko C/C++ -kirjastoja, jotka näkyvät kehittäjille sovelluskehyksen kautta, niitä on muun muassa:

- Media-kirjastot; sisältävät soitto- ja nauhoitustukea monille yleisille ääni- ja video-formaateille.
- 3D-kirjastot, joiden kautta sovellukset voivat käyttää OpenGL ES 1.0 -rajapintoja. Jos päätelaite tukee rautakiihdytettyä grafiikkaa, tämän rajapinta käyttää laitteistoa 3D-grafiikan laskemiseen, muussa tapauksessa laskenta tehdään ohjelmallisesti.
- SQLite, tietokantamoottori, jota kaikki sovellukset voivat käyttää.
- LibWebCore, www-selainmoottori, jota Androidin oma selain käyttää ja jota voidaan käyttää sulautettavana selainnäkymänä sovelluksissa.

3.2.4 Ajonaikainen ympäristö

Androidissa on oma Dalvik-virtuaalikone sovelluksia varten. Jokainen sovellus ajetaan omana prosessinaan omassa Dalvik-instanssissaan. Dalvik on suunniteltu ajamaan monia virtuaalikoneita tehokkaasti. Sovellukset käännetään Javasta poiketen .dex-muotoon, joka on optimoitu käyttämään mahdollisimman vähän muistia. Optimointi tapahtuu kehitysympäristön mukana tulevalla dex-työkalulla, joka kääntää normaalit Java-tavukooditiedostot .dex-muotoon. Dalvik-virtuaalikone on rekisterikone eikä pinokone, kuten Java. Rekisterikoneen etuina on muun muassa se, että se vähentää muistiinviittauksia, koska dataa voidaan pitää rekistereissä ja virtuaalikoneen konekäskyjä tarvitaan vähemmän, koska ei tarvitse käsitellä pinoa jatkuvasti; muuttujien laittaminen ja poistaminen pinosta generoi helposti paljon konekäskyjä. [33]

Androidin ajonaikaiset kirjastot ovat aina olemassa siinä rajapinnassa (API level), mikä on käytössä. MIDP:sta poiketen kirjastot eivät siis ole valinnaisia. Laitteistosta saattaa kuitenkin puuttua jotain ominaisuuksia, joita rajapinnat vaativat. Tämä on ratkaistu siten, että sovelluksissa mukana olevaan manifesti-tiedostoon kirjataan <uses-feature>-elementtiin sovelluksen vaatimat ominaisuudet, jolloin laite ei asenna sovellusta, jos se vaatii jotakin puuttuvaa ominaisuutta.

3.2.5 Linux-ydin

Android käyttää Linux 2.6 -ydintä pohjanaan. Ydin toimii myös rajapintana laitteiston ja muun ohjelmistopinon välissä. Linux ytimestä tulee: turvallisuusmalli, muistin ja prosessien hallinta, verkkopino ja ajurimalli. Sovelluskehityksessä Linux näkyy lähinnä

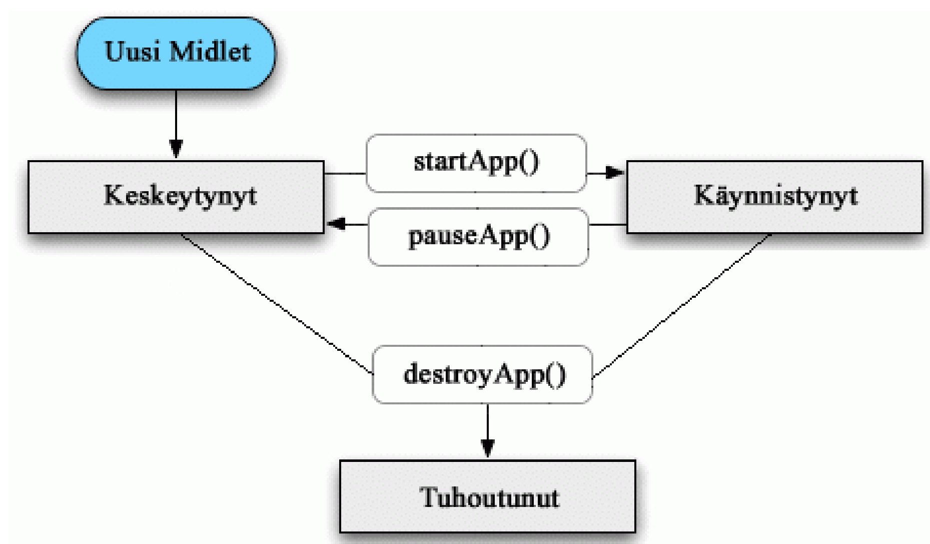
tiedostojärjestelmän osalta ja siinäkään suurimmalle osalle sovelluksia se ei ole mitenkään olennaista.

4 EROT OHJELMOIJAN KANNALTA

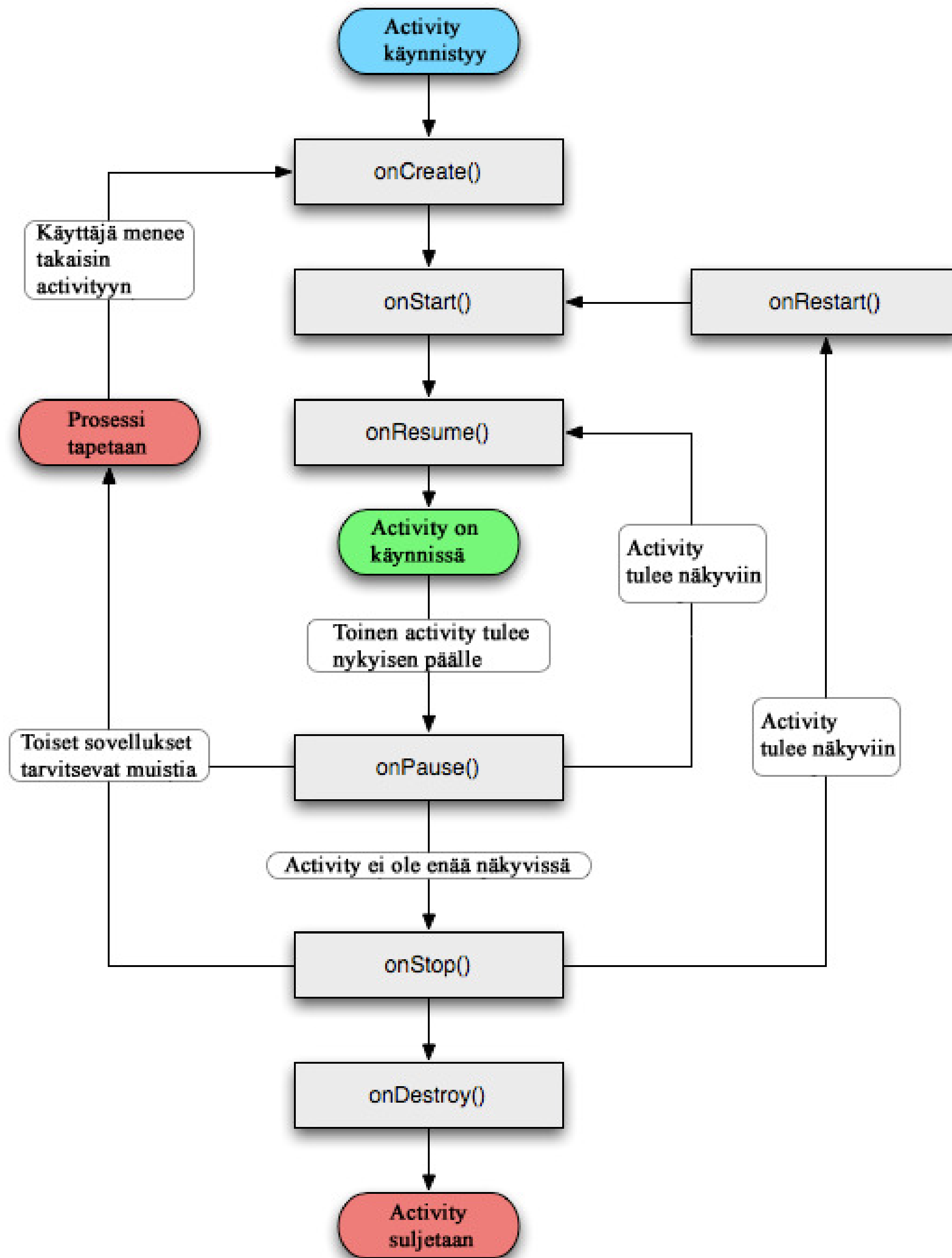
Tässä luvussa käydään läpi MIDP:n ja Androidin eroja ohjelmoijan ja ohjelmistokehityksen kannalta. Aluksi katsotaan sovelluksen elinkaarta ja sen jälkeen katsotaan, miten automaattista testausta voi tehdä molemmissa järjestelmissä. Kolmanneksi käydään läpi tärkeimpiä rajapintoja ja lopuksi tarkastellaan työkaluja ja sovellusten julkaisemista.

4.1 Sovelluksen elinkaari

MIDP:ssä sovellus (Midlet) luodaan keskeytyneeseen (paused) tilaan kuvan 4.1 mukaisesti, mistä järjestelmä sitten kutsuu startApp()-metodia ja sovellus siirtyy käynnistyneeseen (started) tilaan. Tässä tilassa sovellus voi varata ja pitää resursseja ja toteuttaa varsinaista toiminnallisuuttaan. Keskeytyneessä tilassa sovelluksen tulisi vapauttaa kaikki mahdolliset resurssit muiden sovellusten ja päätelaitteen käyttöön. Elinkaaren aikana sovellus voi siirtyä keskeytyneestä käynnistyneeseen ja toisinpäin rajoittamattoman määrän kertoja. Kun sovellus päätyy tuhoutuneeseen (destroyed)-tilaan, se ei enää pääse suoritukseen. Mikäli sovellusta halutaan suorittaa vielä, on se käynnistettävä luomalla uusi instanssi ja suorittamalla sitä.



Kuva 4.1. *Midletin elinkaari [26].*



Kuva 4.2. *Android activityn elinkaari [31].*

Androidin activityn elinkaari kuvassa 4.2 on jonkin verran monimutkaisempi kuin MIDP:n midletin; activity saa muun muassa tietää, kun jokin toinen activity tulee ruutuun joko osittain tai kokonaan nykyisen activityn päälle. Keskeytynyt tila (onPause()-kutsu) on tässäkin tarkoitettu resurssien vapauttamiseksi ja jos sovelluksessa on esimerkiksi tallentamatonta dataa, se tulisi kirjoittaa talteen. Toisaalta onPause():sta pitäisi palata nopeasti, sillä seuraava activity ei pääse ajoon ennen kuin onPause()-metodikutsu

palaa. Activity voi joutua pysähdyksiin (onStop()-kutsu), kun se ei ole enää ollenkaan näkyvissä tai kyseistä activitya ollaan sulkemassa.

4.2 Sovellusten rakentaminen

Sovelluksen saamiseksi ajettavaksi ei riitä pelkkä kääntäminen, vaan prosessiin liittyy esi- ja jälkikäsittelyä. Ohjelmointiympäristöä käytettäessä ohjelmoijan ei välttämättä tarvitse tietää siitä mitään, mutta tietyt asiat täytyy tehdä joka tapauksessa ennen kuin sovellus on valmis ajettavaksi. Molemmissa ympäristöissä sovelluksesta tehdään paketti, joka voidaan sitten siirtää päätelaitteeseen asennettavaksi.

MIDP

MIDP sovellus käännetään normaalisti lähdekoodista, mutta käännettyjä .class-tiedostoja täytyy vielä käsitellä. Kuvassa 4.3 on kuvattu koko prosessi.

Kuva 4.3. *MIDP-sovelluksen rakennusprosessi*

Kääntämisen jälkeen tavukoodit täytyy ajaa preverify-ohjelman läpi. Preverifyn tehtävänä on todentaa, että syötteena annettu tavukoodi menee myös päätelaitteen todentajasta läpi, eli koodissa ei saa olla viittauksia käskyihin tai rajapintoihin, joita päätelaite ei tue. Lisäksi preverifyn tehtävänä on poistaa aliohjelma kutsut ja korvata kutsu aliohjelmassa olevalla koodilla. Kolmantena tehtävänä on lisätä .class-tiedostoihin erityinen StackMap-attribuutti, joka auttaa päätelaitteen todentajaa toimimaan tehokkaammin ajoaikana [36].

Todentamisen jälkeen preverifyn muokkaamat tiedostot laitetaan JAR-pakettiin jar-ohjelmalla.

Manifesti kirjoitetaan erilliseen .jad-tiedostoon, joka sisältää tiedon, missä jar-tiedosto sijaitsee, näin sovellus voidaan asentaa esimerkiksi verkon yli. Manifestin voi laittaa

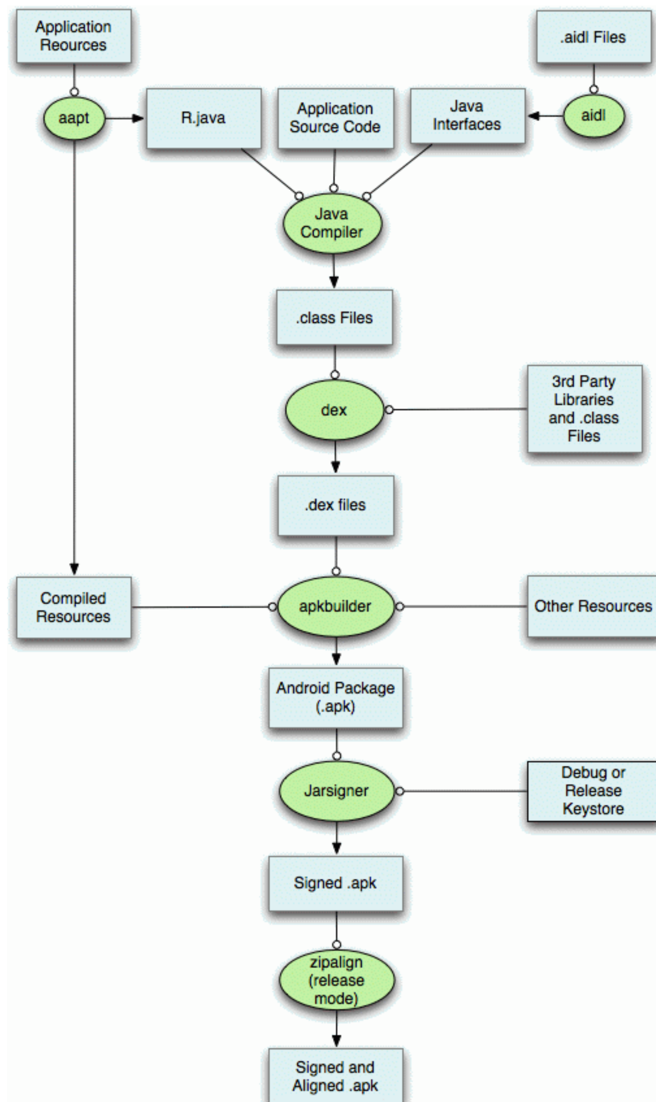
myös jar-paketin sisälle, jolloin se löytyy nimellä META-INF/MANIFEST.MF, muuten se on aivan sama kuin .jad, mutta jar-paketin sisällä ei tarvitse enää uudestaan määritellä, mistä paketti löytyy. Jad-tiedostossa arvo-avain parina täytyy olla MIDlet-Jar-URL, joka kertoo paketin osoitteen ja MIDlet-Jar-Size, joka kertoo kuinka iso paketti. Jar-paketti täytyy hylätä, jos sen koko on eri kuin jad-tiedostossa ilmoitettu. Muut pakolliset arvo-avain parit löytyvät MIDP:n spesifikaatiosta [5].

Jos sovellus allekirjoitetaan, tulee jad-tiedostoon rivi MIDlet-Jar-RSA-SHA1: <base64 koodattu tiiviste>. Päätelaitteen tulee tunnistaa allekirjoittaja tai muuten sovellusta ei saa asentaa. MIDP 2.0 -spesifikaatio ei määrittele mitään juurisertifikaattia, mikä aiheuttaa ongelmia sovellusten siirrettävyyteen laitteesta toiseen. Eri laitteissa voi olla eri juurisertifikaatit, eivätkä ne välttämättä tunne sovelluksen allekirjoittajaa, joten pahimmillaan sama sovellus joudutaan allekirjoittamaan ja jakelemaan erikseen jokaisen eri valmistajan päätelaitteisiin. Mikäli sovellusta ei allekirjoiteta, päätelaitteen tulee informoida käyttäjää asennuksen yhteydessä, että sovellus ei tule luotetusta lähteestä (trusted source), jolloin käyttäjän täytyy pystyä tekemään valistunut päätös haluaako hän antaa sovelluksen luvan asentua. Päätelaite ei saa sallia tällaisen ei-luotetun (untrusted) sovelluksen lukea henkilökohtaista tietoa laitteelta, kuten osoitekirjaa tai kalenteria [5].

Android

Androidissa koodia tuotetaan koneellisesti XML- ja AIDL-tiedostoista (Android Interface Definition Language, Android rajapintamäärittelykieli) [31]. Jos eri prosessit haluavat kommunikoida keskenään, jolloin käyttöjärjestelmän täytyy marshalloida (marshalling, datan konvertoiminen siirrettävään muotoon) kutsut prosessien välillä, tämä rajapinta prosessien välillä täytyy määritellä AIDL-kielellä. Tuotetun lähdekoodin lisäksi sovellukseen kuuluu ohjelmoijan kirjoittama lähdekoodi. XML-tiedostoista tuotetaan käyttöliittymän ja lokalisaation tarvitsemia lähdekooditiedostoja, joita tarvitaan ohjelmoitaessa resursseihin viittaamiseen ja resurssien hakemiseen.

Kuvassa 4.4 näkyy koko prosessi, jossa ovaaleilla on merkitty ajettava työkalu, jonka tuloste päätyy sitten seuraavan vaiheen syötteeksi. Prosessin lopputuloksena on apk-paketti, joka on sitten valmis siirrettäväksi päätelaitteeseen ja ajettavaksi.



Kuva 4.4. *Android-sovelluksen rakennusprosessi [31]*

Rakennusprosessi etenee seuraavasti:

- Aapt (Android Asset Packaging Tool) käsittelee sovelluksen XML-tiedostot ja muodostaa niistä R.java-tiedoston, jonka kautta sovellus käsittelee sovelluksen mukana jaeltavia resursseja, kuten kuvia ja kielitiedostoja.
- Aidl konvertoi mahdollisesti mukana olevat .aidl-rajapinnat tavallisiksi Javan rajapinnoiksi.
- Java kääntäjä kääntää generoidut ja ohjelmoijan tuottamat lähdekoodit .class-tiedostoiksi.
- Dex konvertoi .class-tiedostot Dalvik-virtuaalikoneen käyttämäksi .dex-tavukoodiksi, samoin konvertoidaan mahdollisesti mukana olevat kolmannen osapuolen kirjastot.
- Apkbuilder paketoi .dex-tiedostot ja sovelluksen resurssit .apk-paketiksi.

- Jarsigner allekirjoittaa sovelluspaketin. Allekirjoitus voidaan suorittaa debug-avaimella, joka on tarkoitettu kehitys- ja testausvaiheeseen, jotta sovellusta voidaan ajaa laitteessa tai sitten julkaisu-avaimella, joka voi olla itse allekirjoitettu (self-signed). Avaimen voi tehdä Javan mukana tulevalla Keytool:lla. Se ei ole Android-spesifinen sovellus, kuten ei myöskään jarsigner.
- Jos sovellus on allekirjoitettu julkaisu-avaimella, se täytyy ajaa vielä zipalign-ohjelman läpi, minkä tarkoituksena on vähentää sovelluksen muistin käyttöä, kun sovellusta ajetaan. Tämä saavutetaan siten, että pakkaamattomat resurssit, kuten kuvat, siirretään neljällä jaollisiin osoitteisiin, jotta niihin voidaan viitata mmap()-funktiolla, jolloin sovelluksen ajon aikana muistia kuluu vähemmän [31].

Lopullinen APK-paketti sisältää seuraavat tiedostot ja hakemistot:

- META-INF/MANIFEST.MF-tiedosto, joka on normaali Javan manifesti-tiedosto ja sisältä metadataa sovelluksesta avain-arvo pareina.
- META-INF/CERT.SF sisältää listan paketissa olevista tiedostoista ja näiden SHA1-DIGEST tiivisteen (hash). Näin paketin sisällön muuttaminen hankaloituu, koska SHAn (Secure Hash Algorithm) tuottama tiiviste muuttuu ja päätelaitteen tulee hylätä paketti, kun tiivisteet ovat erit. Ideana on vaikeuttaa pakettien muokkaamista esimerkiksi injektoimalla haittaohjelmia.
- META-INF/CERT.RSA on sovelluksen sertifikaatti, jonka avulla voi varmistaa allekirjoittajan.
- res/ -hakemisto, sovelluksen käyttämät resurssit, kuten XML-tiedostot, kuvat ja kielitiedostot.
- AndroidManifest.xml on sovelluksen XML-muotoinen manifesti, jossa kuvaillaan sovelluksen ominaisuuksia.
- classes.dex eli käännetty ohjelma Dalvik-tavukoodina.
- resources.arsc sisältää aapt:n kääntämät resurssit binäärimuodossa, esimerkiksi res-hakemiston XML-tiedostot.

4.3 Sovelluksen julkaiseminen

Testatessa sovelluksen voi siirtää esimerkiksi USB-kaapelin välityksellä päätelaitteeseen, mutta tämä ei luonnollisestikaan sovi loppukäyttäjille. Lisäksi julkaisemiseen liittyy kaupallisia intressejä; sovelluksia pitää pystyä myymään turvallisesti ja helposti käyttäjille, jolloin sovelluskehittäjilläkin on enemmän intressejä kehittää kyseiseen ympäristöön sovelluksia.

MIDP

MIDP-sovelluksia ajetaan lähinnä puhelimissa. Nämä markkinat ovat fragmentoituneet, koska markkinoilla on monia valmistajia ja valmistajilla monia eri malleja, eikä sama sovellus aina toimi edes saman valmistajan eri malleissa. Toimimattomuus johtuu erilaisista puhelimen ominaisuuksista, kuten näytön koosta, erityisesti pelien kohdalla, joten

sama sovellus joudutaan paketoimaan erikseen eri malleja varten. Fragmentoitumisesta johtuen ei ole myöskään mitään erityistä kauppapaikkaa, josta sovelluksia voisi ostaa, vaan eri toimijoilla on omat kauppapaikkansa, kuten Nokian Nokia Store [37]. Sovelluskehittäjän täytyy tehdä kaikkien kauppapaikkojen kanssa erikseen sopimus sovelluksen myymisestä. Sovelluksia voi myös jaella Internetin kautta ilmaiseksi tai maksullisesti esimerkiksi maksullisen SMS-palvelun kautta.

Android

Google tarjoaa Android Marketin [38], joka on keskitetty markkinapaikka ja jonne pääsee oletuksena kaikista Android-laitteista, kunhan vain Internet-yhteys on olemassa. Sovelluskehittäjän on helppo julkaista sovellus vain yhteen paikkaan, eikä siihen tarvita erillisiä sopimuksia, vaan Android Market ottaa prosenttiosuuden sovelluksen hinnasta ja lisäksi kehittäjä maksaa pientä kiinteää vuosimaksua saadakseen julkaista sinne. Android-sovelluksia voi ladata myös toisista kauppapaikoista tai verkon yli, mutta ylivoimaisesti suurin osa sovelluksista on Android Marketissa. Android Marketissa ei toisaalta voi julkaista ihan mitä tahansa sovelluksia, esimerkiksi aikuisviihde ja erilaiset uhkapelisovellukset ovat kiellettyjä [39].

4.4 Rajapinnat

Tähän lukuun valitut tarkasteltavat rajapinnat on valittu sen mukaan, mitä käytännön työssä on havaittu tärkeäksi tai usein käytetyksi. Valinta on siis osittain mielivaltainen, eikä rajapintoja ole käsitelty minkään tärkeysjärjestyksen mukaisesti. Tämän lisäksi käytännön MIDP-sovellukset sisältävät paljon valinnaisia rajapintoja eri tarkoituksia varten. Android on sen sijaan paljon homogeenisempi ympäristö.

4.4.1 Käyttöliittymä

Käyttöliittymää tehtäessä iso ero on värien käytössä; Androidin värit sisältävät läpinäkyvyyttä, kun taas MIDP:ssä ei läpinäkyvyyttä voi piirtää normaaleilla piirtometodeilla. MIDP:ssä voi kyllä käyttää kuvia, jotka sisältävät läpinäkyvyyttä. Molemmat järjestelmät tukevat kosketusnäyttöä ja perinteistä näppäimien käyttöä.

MIDP

Javax.microedition.lcdui-paketti tarjoaa komponentteja yksinkertaisten listojen ja lomakkeiden tekemiseen, esimerkiksi monivalinta- ja tekstikenttäkomponentit. Niiden ulkoasuun ei kuitenkaan voi vaikuttaa, eikä komponentteja voi asetella tarkasti haluamiinsa paikkoihin. Näiden toiminnallisuuksien toteuttamiseksi täytyy toteuttaa omat komponentit, joko perimällä lcdui-komponentteja tai tekemällä oma komponenttikirjasto. MIDP on alun perin kehitetty aikana, jolloin kosketusnäyttö ei ollut yleinen, joten se

ei ota kantaa, miten esimerkiksi ruudun vieritys tapahtuu, vaan se on valmistajakohtaista. Esimerkiksi Oraclen emulaattorissa ruutuun tulee vierityspalkki, jota voi vetää ylös ja alas. Näin esimerkiksi nykyään suosittu kineettinen vieritys täytyy toteuttaa itse, joka johtaa siihen, että käytännössä käyttöliittymäelementit täytyy toteuttaa itse. Samoin tuki sormella tehtäviä eleitä (gesture) varten täytyy toteuttaa itse.

Android

Androidissa on erilaisia näkymiä (View), joilla ruudulle voidaan asetella komponentteja. Näkymien latominen määritellään erillisissä XML-tiedostoissa, joissa kerrotaan, miten kyseinen näkymä ladotaan ruudulle. Samoin näkymään tulevien komponenttien värimaailman voi määrittää XML-tiedostossa, jolloin järjestelmä asettaa automaattisesti värit määrittelyn mukaisesti. Tämän jälkeen ohjelmakoodissa riittää vain asettaa XML:ssä määritelty tunniste komponentille. Sovelluskehittäjä voi määrätä värit haluamakseen myös ohjelmakoodissa, joten XML:n käyttö ei ole välttämätöntä. Androidissa on tuki kosketusnäytölle ja siihen sisältyy automaattisesti muun muassa kineettinen vieritys, eli kun näytöllä vetää sormea, vieritys saa jonkin vauhdin ja se hidastuu ja lopulta pysähtyy. Sormieleitä varten on oma rajapinta ja sovellus (Gesture Builder), joten eleiden käyttöönotto onnistuu yksinkertaisesti. [31]

4.4.2 Multimedia

MIDP:n Player ja Androidin MediaPlayer käyttävät lähes identtistä tilamallia ja metoditkin ovat lähes samoja; start(), stop() ja pause(), erona on alussa kutsuttava metodi: MIDP:ssä Player.prefetch() ja Androidissa MediaPlayer.prepare(). Tuetuissa formaateissa on suuria eroja, sillä MIDP ei vaadi kuin PNG-tuen, Android sen sijaan tukee lähes kaikkia yleisimpiä formaatteja. Android tarjoaa myös tuen multimedian tallentamiseen.

MIDP

Javax.microedition.media paketista löytyvä Player-luokka osaa soittaa multimediaa laitteen muistista tai verkon ylitse suoratoistoa RTSP-protokollan avulla. Eri videoformaattien tukemista ei vaadita spesifikaatiossa, vaan tuki on aina laitekohtaista. Kuvia voi ladata sovelluksen omaan käyttöön javax.microedition.lcdui.Image luokan kautta, tuettuja formaatteja on vähintään PNG. Valinnaisen JSR-272 DVB-H-paketin kautta MIDP tukee myös DVB-H-tv-katselua. Hankalinta MIDP:ssa on videoformaattien tuki, sillä tähän liittyen ei ole mitään vaatimusta tiedostomuodoista, joten pahimmassa tapauksessa yhdessä laitteessa testattu videon toisto ei välttämättä toimi toisessa laitteessa. Mielenkiintoisena yksityiskohtana on vaatimus, että MIDP-laitteella voi soittaa ohjelmallisesti monotonisia ääniä.

Android

Android.media.MediaPlayer-luokka hoitaa videoiden ja äänen toiston, muistista tai http- tai rtsp-osoitteesta. Tuettuja videoformaatteja ovat vähintään: H.263, H.264 AVC ja MPEG-4 SP ja yleisimmät audioformaatit, kuten AAC, mp3, MIDI sekä muutama muu. Kuvaformaateista tuettuja ovat JPEG, GIF, PNG ja BMP. Android tukee myös multimedian tallettamista luokan android.media.MediaRecorder kautta. Tuettuja formaatteja ovat AMR-NB (audio), JPEG (kuva) ja H.263 (video). [31]

4.4.3 Yhdistyvyys

Android tarjoaa Java SE:n java.net-paketista kattavat rajapinnat verkkoyhteyksien luomiseen ja hallinnointiin, MIDP:ssa java.net-pakettia ei ole mukana ja siinä TCP-yhteydet rajoittuvat http- ja https-yhteyksien ottamiseen. MIDP:llä voi tehdä UDP-palvelimen ja asiakkaan. Puheluja ja SMS-viestejä voi lähetellä molemmilla alustoilla.

MIDP

Sovellus voi yrittää aloittaa puhelun javax.microedition.midlet.Midlet-luokan platformRequest()-metodin kautta. Metodille annetaan puhelinnumero RFC2806:n [40] määrittämässä muodossa tel:<numero>. Tämä metodi saattaa tukea myös muitakin määreitä kuten fax: tai sms:, laitteesta riippuen. SMS-viestien lähetys onnistuu, jos laite tukee valinnaista JSR-205: Wireless Messaging API (WMA)-pakettia. Internet-yhteyksiä voi luoda javax.microedition.io.Connector-luokan kautta http- ja https-protokollilla, mutta puhtaita TCP-socket yhteyksiä ei voi luoda. Connector-luokan kautta voi kyllä luoda UDP-palvelimen ja kuunnella ja lähettää UDP-paketteja.

Android

Androidin android.content.Intent-luokan kautta sovellus voi aloittaa puhelun ja android.telephony.SmsManager-luokka tarjoaa metodin sendTextMessage() SMS-viestien lähettämiselle. Sockettien luomista varten Android tukee Java SE:iin kuuluvaa java.net.Socket-luokkaa, jolla Socketteja voi luoda ja käsitellä. Samasta paketista on myös ServerSocket-luokka, jolla voi kuunnella jotain porttia ja toteuttaa oman palvelimen tai protokollan tarvittaessa. Java SE:sta tulee myös java.net.URL-luokka, jolla voi avata http- ja https-yhteyksiä.

4.4.4 Siirräntä

Sovellukset tarvitsevat joskus pysyväismuistia tallentaakseen tietoa eri käyttökertojen välillä. MIDP:ssä onnistuu luku ja kirjoitus, mutta muu tiedostojen manipulointi vaatii

valinnaista pakettia. Androidissa on laaja tuki tiedostojen ja hakemistojen manipuloinnille.

MIDP

Tiedostojen manipulointi rajoittuu lukemiseen ja kirjoittamiseen javax.microedition.io.Connector- ja javax.microedition.io.Connection-luokkien kautta. Tiedostoja ei voi tuhota tai uudelleennimetä tätä kautta. Näin ollen jos sovellus luo paljon tiedostoja (eli kirjoittaa tiedostoon, jota ei ole vielä), se ei välttämättä pysty itse siivoamaan turhiksi käyneitä tiedostoja pois.

Valinnainen JSR-75 FileConnection-paketti [41] tarjoaa javax.microedition.io.file.FileConnection-luokan, jolla tiedostoja voi tuhota ja nimetä ja hakemistojen sisältöä voi katsella. Laitteesta riippuen tiedostojärjestelmässä voi olla tiedostoja ja hakemistoja, joihin ei pääse käsiksi sovelluksesta.

Android

Androidissa on Java SE:n java.io.File, jolla voi vapaasti manipuloida tiedostoja ja hakemistoja, eli luoda, poistaa, lukea, kirjoittaa. Tiedostot luodaan sovelluksen ID:llä, joten muut sovellukset eivät voi lukea tai kirjoittaa niihin, ellei omistajasovellus sitä salli. Alla oleva Linuxin tiedostojärjestelmä asettaa muut rajoitteet, minne sovellus voi luoda tiedostoja tai poistaa niitä.

4.4.5 Työkalut

Molempiin ympäristöihin löytyy kehitystyökaluja, jotka integroituvat kehitysympäristön osaksi, jolloin ohjelmoijan ei tarvitse välttämättä tietää, mitä kaikkia komentoja pitää ajaa, jotta sovelluksen saa paketoitua. Samoin sovelluksia voi ajaa emulaattorissa kummassakin ympäristössä, mikä nopeuttaa testausta.

MIDP

Oracle tarjoaa SDK:ta (Software Development Kit) [42], joka sisältää muun muassa emulaattorin ja profiloijan sovellusten ajamiseen ja profiloimiseen. SDK on toteutettu NetBeans-pluginina [43], joten se vaatii NetBeans kehitysympäristön asentamisen ensin. Toiselle suositulle kehitysympäristölle, Eclipselle [44], löytyy myös plugin, EclipseME [45], mutta tämä ei sisällä profiloijaa. Oraclen SDK-paketista emulaattoria voi kyllä ajaa myös ilman NetBeansia. Myös paljon muita työkaluja on olemassa [46]. Minimissään tarvitaan kääntäjä ja MIDP-kirjastot, joita vastaan käännetään, sekä preverify-työkalu, joka tarkistaa, että sovellus ei sisällä luvattomia kutsuja tai virtuaalikoneen käskyjä. Kolmanneksi tarvitaan vielä JAR-paketteja tekevä ohjelma sovellusten paketoimiseksi. Koska MIDP on vain määrittely, kuka vaan voi tehdä toteutuksen siihen, joten tässä ohjelmoija ei ole sidottu vain esimerkiksi Oraclen työkaluihin ja emulaattori-

toteutuksiakin löytyy esimerkiksi Nokialta [47]. Vaikka nämä emuloivat ensisijaisesti Nokian eri puhelinmalleja, niissä on myös MIDP-toteutus mukana.

Android

Androidin kehittäjäsivustolta [31] voi ladata SDK:n, joka sisältää työkalut ohjelmistokehitystä varten ja pluginin Eclipseen [44]. Samaan tapaan kuin MIDP:n yhteydessä SDK ei vaadi tietyn kehitysympäristön käyttämistä, vaan SDK-paketissa on mukana työkalut sovellusten kääntämistä ja paketointia varten. Muita hyödyllisiä työkaluja ovat muun muassa emulaattori ja debug-monitori ddms (Dalvik Debug Monitor Server), jolla voi seurata debug-viestejä ja kutsupinoja tai käyttää sitä debuggerina, jolloin suoritusta voi seurata lähdekoodirivi kerrallaan. Lisäksi työkalu toimii profiloijana.

4.5 Testaus

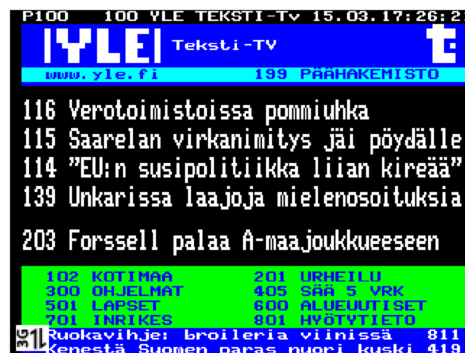
Java-maailmassa JUnit [34] on suosittu yksikkötestauskehys. JUnit käyttää testien ajamiseen reflektiota, jota ei ole tuettu MIDP:ssä. Reflektion avulla voidaan luoda luokista olioita vain tietämällä luokan nimi tai kutsua metodeita nimen mukaan, jolloin reflektioon perustuva testausjärjestelmä voi olla erillään itse testattavasta sovelluksesta. Jos reflektiota ei ole käytössä, täytyy sovelluksen sisältää myös kaikki testaukseen liittyvä toiminnallisuus. Androidissa java.lang.reflect-paketti on mukana ja Android tukee JUnit-testausta; JUnit tuleekin Androidin kehitysympäristön mukana. MIDP:een löytyy toisia ratkaisuja, mutta niillä on monia huonoja puolia toiminnallisuudessa ja suorituskyvyssä JUnitiin verrattuna [35].

5 TEKSTI-TV-SOVELLUKSEN SIIRTÄMINEN

Työn tarkoituksena oli Teksti-TV-sovelluksen [8] siirtäminen MIDP-ympäristöstä Android-ympäristöön. Tässä luvussa käydään läpi vaatimukset, jotka siirretyn sovelluksen täytyy toteuttaa ja sen jälkeen käydään läpi siirtämisen vaiheet. Kolmanneksi tarkastellaan, miten siirto onnistui ja lopuksi arvioidaan kumpaakin ympäristöä sovelluksen näkökulmasta ja käydään läpi rajapintoja, joita sovellus käyttää kummassakin ympäristössä.

5.1 Olemassaoleva Teksti-TV-sovellus

Teksti-TV-sovelluksella voi katsella teksti-tv sivuja, jotka ladataan verkon yli Sofia Digital Oy:n palvelimelta. Sovellus on alun perin tehty MIDP-ympäristöön. Käyttäjä saa käyttää ohjelmistoa 7 päivää ilmaiseksi, jonka jälkeen hänen täytyy ostaa lisää käyttöoikeuksia lähettämällä SMS-viesti palveluntarjoajan numeroon. Jos käyttöoikeuksia ei ole, käyttäjä voi lukea vain aloitussivua, eli useimmiten sivua 100. Sovelluksen käynnistyessä haetaan palvelimelta lista palveluntarjoajista, joista käyttäjä valitsee, minkä tarjoajan teksti-tv:tä hän haluaa katsella. Valittuaan teksti-tv:n käyttäjä pääsee lukemaan teksti-tv sivuja, joilla voi navigoida eteenpäin ja taaksepäin sekä alisivuille. Käyttäjä voi myös valita sivuilla olevia kolminumeroisia linkkejä, joilla pääsee suoraan linkissä lukevalle sivulle.



Kuva 5.1. *kuvaruutukaappaus Teksti-TV-sovelluksesta*

Kuvasta 5.1 näkyy, että käyttöliittymä on hyvin samanlainen kuin televisioiden teksti-tv:kin, vain alareunassa näkyvä 3G-ikoni ei kuulu sovellukseen, vaan puhelin piirtää sen merkiksi, että 3G-yhteys on avoinna. Palvelimelta saatavat, käyttäjälle näytettävät sivut tulevat ETSI EN 300 706 standardin [48] mukaisina tiedostoina, jotka jäsennetään ja näytetään käyttäjälle.

5.2 Vaatimukset

Sovelluksen Android-versioon piti siirtää kaikki sama toiminnallisuus kuin MIDP-versiossakin lukuun ottamatta linkkien navigoimista (nuoli-)näppäimistöllä, sekä mainoskuvan näyttämistä palveluntarjoajat-valikossa. Mainoskuva tulee seuraavaan versioon ja näppäimistönavigointia ei tarvita tällä hetkellä, sillä tiedossa ei ole laitteita, joissa on näppäimistö, mutta ei kosketusnäyttöä. Android-käyttöjärjestelmän määriteltiin olevan 2.0 tai uudempi.

5.3 Siirtämisen vaiheet

MIDP-versiosta oli tunnistettavissa ainakin kolme erilaista kokonaisuutta, jotka voisi siirtää erikseen: sivun jäsennys ja piirto, käyttöliittymä ja verkkoyhteyksien hallinta. Lisäksi uutena ominaisuutena toteutettiin, verkkoliikenteen vähentämiseksi, välimuisti sivujen tallentamista varten.

Työ aloitettiin sivujen jäsennyksestä ja piirrosta. Nämä olivat jo valmiiksi erotettu koodissa omiksi osikseen. Tässä kohdassa ainoa ei-siirrettävä osuus oli resurssien lataaminen, eli käytännössä kirjasin-tiedostojen luku. MIDP:ssä tiedosto luetaan jar-paketista Class-luokan getResourceAsStream(String)-metodilla ja Androidissa apk-paketista AssetManagerin open(String)-metodilla. Androidissa on käytössä myös Class.getResourceAsStream(String)-metodi, mutta se ei toimi samoin kuin MIDP:ssä, vaan se käyttää juurihakemistona classpathia, eli hakemistoa, josta etsitään käännettyjä luokkia. Piirron testaamisessa tehtiin normaali Java-sovellus, joka vain luki ja piirsi teksti-tv-sivun, jolloin testaaminen oli paljon nopeampaa, kuin tehdä jokin testisovellus, joka pitäisi sitten ladata joko Android-emulaattoriin ja puhelimeen. Myöhemmässä vaiheessa paljastui, että koska Android-laitteiden näytöissä voi olla isompi resoluutio kuin missä MIDP-sovellusta on ajettu, teksti-tv-sivu ei täytä koko ruutua niin kuin olisi tarkoitus. MIDP-sovelluksessa oli kyllä tuki piirtää jokainen pikseli isompana, 2x2 pikselin kokoisena, mutta toteutuksessa oli virhe, jonka vuoksi se ei toiminut oikein. Siirtämisen aikana tämä virhe korjattiin myös MIDP-versiosta.

Seuraavaksi siirrettävänä oli käyttöliittymä, mutta tästä selvisi heti, ettei olemassa olevaa koodia voi, eikä kannata käyttää, koska erot järjestelmien välillä olivat liian suuret. Siksi käyttöliittymä toteutettiin alusta alkaen uudelleen, kuitenkin säilyttäen logiikka samanlaisena. MIDP-versio oli myös alun perin toteutettu ennen kuin kosketusnäytöt olivat käytössä, joten se perustui hyvin pitkälti näppäimien lukemiseen, jonka päälle oli lisätty kosketusnäyttötuki. Android-versio aloitettiin suoraan kosketustuesta ja näppäimistötuki tehtiin myöhemmin. MIDP:ssä oli myös tarpeen piirron yhteydessä tutkia, miten päin näyttö on käännetty, mutta Androidissa Activity käynnistetään uudestaan,

kun näyttöä käännetään. Jos AndroidManifest.xml:ssä on määritelty, että Activity on kiinnostunut näytön kääntymisestä, järjestelmä kutsuu Activityn onConfigurationChanged()-metodia näytön kääntyessä, jolloin Activity voi tehdä tarvittavat toimenpiteet. Teksti-TV-sovelluksen tapauksessa lasketaan uudet dimensiot näkymälle ja vaihdetaan virtuaalinäppäimistön asettelu paremmin uuteen asentoon sopivaksi.

Lopuksi vielä piti siirtää verkkotoiminnallisuus, jota tarvitaan asiakkaan tunnistamiseen sekä sisällön hakemiseen palvelimelta. Tässäkin paljastui nopeasti, että koodia ei kannata siirtää, koska Android tarjoaa tarvittavat rajapinnat ja palvelut kaikkeen, mitä sovellus tarvitsee. MIDP-versiossa oli itse toteutettua pipareiden (cookie) ja yhteyksien hallintaa, mitä ei Androidissa tarvita, koska siellä voi käyttää esimerkiksi DefaultHttpClient-luokkaa.

Välimuistin vaatimuksina oli tukea http-otsikkotiedoissa tulevaa max-age-arvoa, joka määrittelee, kuinka kauan ladattua sivua voi säilyttää välimuistissa. Vanhentuneita sivuja ei tarvitse poistaa aktiivisesti, vaan tilalle kirjoitetaan uusi sivu, kun se on haettu. Ominaisuus toteutettiin punamustan puun avulla siten, että avain muodostettiin sivun ja alisivun numeroista, jolloin se on yksikäsitteinen, ja hyötykuormana oli expires-tieto ja varsinainen sivun data. Ainoa ongelma oli, että punamustalle puulle ei ole toteutusta MIDP:ssä, joten se toteutettiin itse. Puun asemesta olisin voinut myös käyttää Hashtablea (hajautustaulu), mutta sitä ei haluttu käyttää, koska jossain nähdyssä virtuaalikonetoteutuksessa hajautusfunktio oli `avain % taulukon koko`, jolloin törmäyksiä voi tulla niin paljon, että punamustan puun käyttö kannattaa, vaikka kyseessä ei olekaan aikakriittinen kohta. Toisaalta puun toteutus oli jo olemassa, joten vaiva ei ollut kovin suuri.

Loppujen lopuksi siis välimuisti oli ainoa osa, joka on aidosti kokonaan yhteinen molempien versioiden kesken. Sivujen jäsennys on myös samanlainen, mutta toteutuksessa oli riippuvuuksia piirtoon, jossa taas kirjasinten lataaminen täytyy tehdä eri tavalla eri järjestelmissä, mutta muuten koodi on samaa.

Isoista eroavuuksista johtuen eri versioiden lähdekoodeja ei voinut säilyttää samassa versionhallinnan projektissa, vaan Android-versiolle oli luotava oma projektinsa ja MIDP sai jäädä vanhaan olemassa olevaan projektiin. Välimuistille luotiin lisäksi vielä oma projektinsa, koska kaikkea siinä olevaa koodia voitiin käyttää hyväksi kummassakin projektissa, joten ylläpidettävyyden kannalta on parempi pitää sitä omana projektinaan.

5.4 Miten siirto meni

Paras tapa olisi pitää molemmat versiot samassa versionhallinnassa, jolloin yhteinen koodi olisi helpoimmin ylläpidettävissä. MIDP-versio kuitenkin käyttää ehdollista kään-

tämistä Antennan [49] avulla eikä tätä ollut tarjolla Androidille. Lisäksi ehdollinen kääntäminen vaikeuttaa koodin lukemista huomattavasti, joten oma versiohallinnan projekti oli parempi ratkaisu.

Käyttöliittymässä oli paljon ehdollista kääntämistä. Lisäksi perimmäinen filosofia järjestelmien välillä on niin erilainen, että käyttöliittymä oli parempi tehdä Android-tyylisesti, jotta se tuntuu Android-ohjelmalta, erityisesti valikkojen osalta. Muuten käyttöliittymän logiikka toteutettiin samanlaiseksi kuin alkuperäisessä versiossa.

Lopulta lopputulos oli varsin onnistunut, käyttöliittymä sopii Android-tyyliin ja skaalautuu erikokoisille näytöille, myös tableteille. Toteutuksessa käytettiin Androidin ominaisuuksia, kuten ASyncTask:ia perinteisen itse tehdyn säikeistyksen asemesta. Parannettavaakin vielä löytyy; yksikkötestejä olisi voinut tehdä, jolloin MIDP:nkin käyttämää koodia voisi yksikkötestata ja luokkia olisi voinut refaktoroida pienemmiksi. Toiminnallisuus on kuitenkin kunnossa.

5.5 Millainen MIDP oli

MIDP:ssä kaikki tuntui subjektiivisesti hieman vanhanaikaiselta ja määrittelyn ikä ja historian painolasti näkyy monessa yksityiskohdassa. Koska määrittely on tehty nykypäivän standardien mukaan heikkotehoisille laitteille, niin jäi kuva, että myös rajapinnat olivat osittain ”heikkotehoisia”, koska esimerkiksi käyttöliittymää ei juurikaan voi personoida oman sovelluksen näköiseksi toteuttamatta kaikkia käyttöliittymäkomponentteja itse. Silti MIDP on käyttökelpoinen ympäristö ja sillä on mahdollista toteuttaa monimutkaisiakin asioita tarvittaessa.

5.5.1 Jad-tiedosto

Teksti-TV-sovellus on jaossa Internetissä, joten siinä täytyy olla asentamista varten jar-paketin lisäksi erillinen jad-tiedosto. Jad-tiedoston sisältö löytyy liitteestä A, joskin siitä on poistettu viittaus jar-pakettiin, sertifikaatit ja SHA-tiiviste. Tämän sovelluksen tapauksessa kannattaa huomata MIDlet-Permissions-Opt, joka kuvaa, että sovellus osaa käyttää esimerkiksi SMS-palveluita, jos päätelaitteesta löytyy sille tuki. Sen lisäksi siellä on Nokian puhelimelle tarkoitettuja ohjeita ”Nokia-”-alkuisina avaimina. Muut kuin Nokian laitteet voivat vain hylätä nämä määrittelyt ilman, että se vaikuttaisi sovelluksen toimintaan.

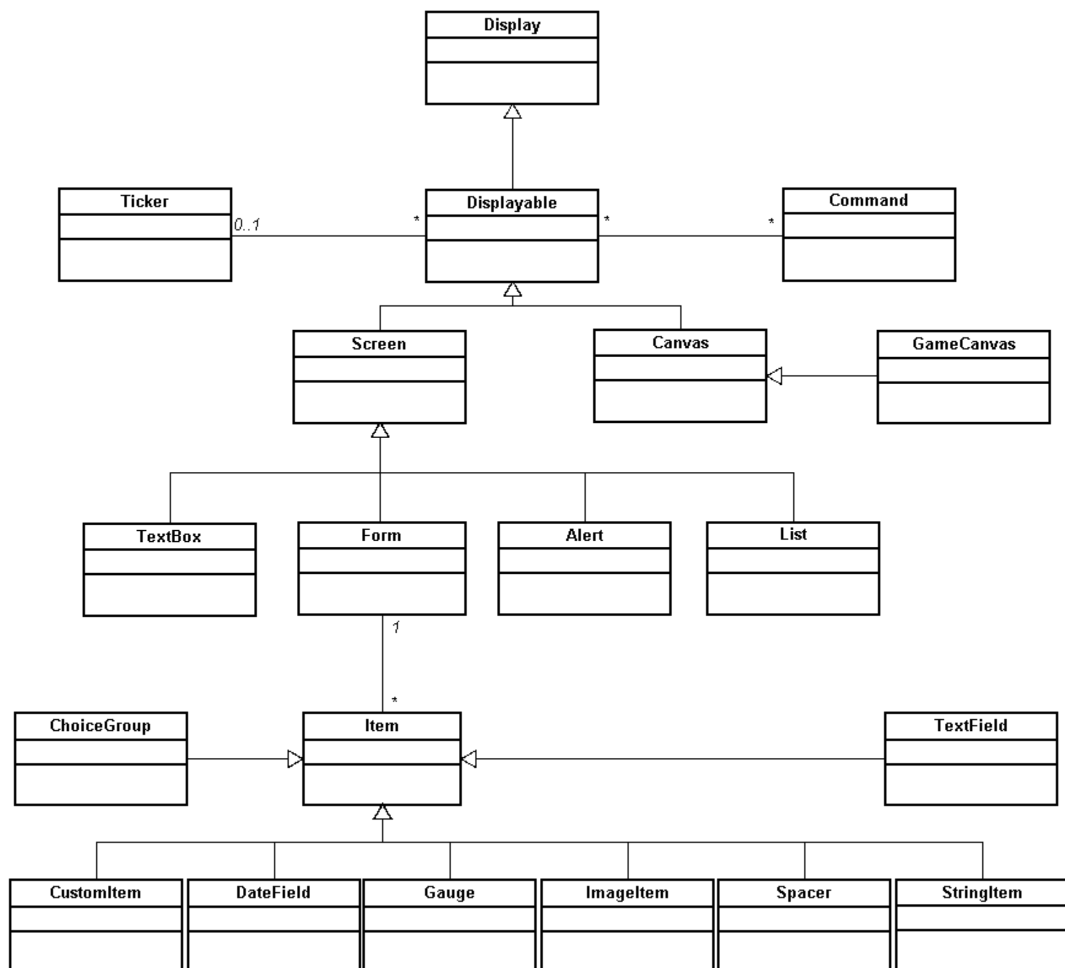
5.5.2 Elinkaari

MIDP-version sovelluksen elinkaari noudattaa edellä kuvattua elinkaarta, mutta sovellus ei vapauta pauseApp()-kutsussa mitään, vaan pitää kaikki resurssit varattuina koko ajan. Sen sijaan sovellus kuuntelee javax.microedition.lcdui.Canvas-luokan kautta hi-

deNotify()- ja showNotify()-tapahtumia, jotka tulevat, kun sovellus menee pois ruudusta ja palaa ruutuun. Jos sovellus ei ole näkyvissä ruudulla, se pysäyttää taustasäikeiden ajon ja täten toimii ikään kuin pause-tilassa. StartApp()-kutsussa luodaan resurssit ensimmäisellä kerralla ja muilla kerroilla käytetään aikaisemmin luotuja resursseja. DestroyApp()-metodissa vapautetaan resurssit sovelluksen kuollessa.

5.5.3 Käyttöliittymä

MIDP:n käyttöliittymäluokat sijaitsevat javax.microedition.lcdui-paketissa. Käyttöliittymmä koostuu loogisesti korkean tason ja matalan tason rajapinnoista. Kuvasta 5.2 näkee, kuinka korkean tason elementit perivät Screen-luokan ja matalan tason toteutus käyttää Canvas-luokkaa ja sen kautta Graphics-luokkaa suoraan ruudulle piirtämiseen.



Kuva 5.2. *MIDP javax.microedition.lcdui-paketin käyttöliittymäluokkien luokkakaavio.*

Korkean tason rajapinta on suunniteltu siten, että sovelluksien siirtäminen laitteista on mahdollisimman helppoa. Tästä johtuen rajapinta ei anna mahdollisuutta vaikuttaa käyttötuntumaan, vaan se tulee laitteen natiivitoteutukselta. Sovellus ei siis voi vaikuttaa esimerkiksi väreihin, eikä se ole tietoinen ruudun vierityksestä tai näppäimien

painalluksista. Korkean tason valmiita komponentti-toteutuksia löytyy listoihin, kuviin, päivämääriin, teksteihin ja mittaria (gauge) varten.

Canvas-luokkaa käyttämällä, käytännössä perimällä, saadaan täysi kontrolli, mitä ruutuun piirretään, sekä voidaan kuunnella näppäin-tapahtumia. Järjestelmä kutsuu Canvaksen metodeita keyPressed(), keyReleased() ja keyRepeated(), kun käyttäjä tekee jotain näppäimistöllä. Java SE:stä poiketen näiden parametrina ei tule KeyEvent-luokan oliota, vaan `int keyCode`. KeyPressed():iä kutsutaan, kun näppäintä painetaan ja keyReleased() syntyy vastaavasti, kun käyttäjä päästää näppäimen ylös. KeyRepeated()-kutsu tulee, jos näppäintä toistetaan, eli käytännössä pidetään pohjassa. Laitteisto ei kuitenkaan välttämättä tue tätä ominaisuutta. Tuen voi selvittää kutsumalla Canvaksen hasRepeatEvents()-metodia. Kosketusnäytöllisissä laitteissa kosketustapahtumat välitetään pointerPressed(), pointerReleased() ja pointerDragged()-metodien kautta. Metodeilla on parametreina Canvakseen suhteelliset x- ja y-koordinaatit, jotka ilmaisevat, mihin kosketus kohdistui. PointerPressed() ja pointerReleased() vastaavat keyPressed()- ja keyReleased()-metodeja. PointerDragged():iä kutsutaan, kun käyttäjä säilyttää kosketuksen näyttöön ja liikkuu siinä tai pysyy paikallaan. Tässäkin tilanteessa koordinaatit tulevat suhteessa Canvakseen, eikä esimerkiksi edelliseen sijaintiin. Laitteiston kosketustuen voi kysyä Canvakselta hasPointerEvents()-metodilla ja tuen raahaamiseen (pointerDragged) voi kysyä hasPointerMotionEvents()-metodilla.

Käyttöliittymän asettamiseen tarvittava Display-instanssi saadaan kutsumalla Display.getDisplay(Midlet)-metodilla. Parametrina on tyypillisesti `this`, joka viittaa parhaillaan suoritettavaan Midlettiin. Display:lle voidaan sen jälkeen asettaa abstraktin luokan Displayable toteuttava olio. Displayable on näytölle asetettava olio, jolla voi olla otsikko, tikkeri (ticker), nolla tai useampia komentoja ja kuuntelija komennoille. Toisena vaihtoehtona Display tarjoaa setCurrent(Alert, Displayable)-metodin, jolle voi antaa Alertin, joka huomauttaa käyttäjälle jostain ja mahdollisesti odottaa kuittausta. Alertin poistumisen jälkeen Display asettaa parametrina saadun Displayable-komponentin ruudulle.

Kuva 5.3. *Näppäimistö Sun Microsystemsin Java ME 3.0 SDK:n emulaattorista*

Käyttäjä antaa syötettä näppäimistön avulla tai kosketusnäytön kosketuksilla. Näppäimistön (kuva 5.3) yläosassa vasemmalla ja oikealla olevilla pehmonäppäimillä valitaan sovelluksen toimintoja, jotka laukaisevat komentoja (Command) niitä kuuntelevalle kuuntelijalle. Muiden näppäimien toiminta riippuu sillä hetkellä aktiivisena olevasta näyttöelementistä, esimerkiksi listassa nuolinäppäimet voivat vierittää listaa. Displayable tarjoaa metodin addCommand(), jolla lisätään Command-olioita kyseiseen ruutuun. Tämän lisäksi pitää asettaa kuuntelija setCommandListener()-metodilla. Käyttäjän valitessa jonkin komennon kuuntelija saa tiedon asiasta ja päättää, miten sovelluksen pitäisi reagoida siihen. Se, miltä komennot näyttävät käyttäjälle, riippuu laitteiston toteutuksesta, eikä ohjelmoijalla ole paljoa valtaa vaikuttaa asiaan. Komennoille annetaan tyyppi, joita on 7 erilaista: BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN ja STOP. Tyyppi vaikuttaa, mihin pehmonäppäimeen komento sidotaan. Lisäksi komennolle annetaan prioriteetti, joka on sovelluskohtainen, ja joka voi vaikuttaa missä järjestyksessä komennot listataan valikossa. Erikoistapauksena sovellus voisi olla laittamatta ruudulle yhtään komentoa, mutta tällöin käyttäjällä ei olisi keinoa päästä sovelluksesta pois tai vaihtaa sovellusta ilman sovelluksen tappamista muilla keinoin.

Teksti-TV-sovelluksen käyttöliittymässä yksi käyttötapaus on käyttöehtojen näyttäminen. Tämä tehdään MIDP:ssä luomalla Form ja lisäämällä siihen komentoja Form.addCommand()-metodilla, lisäksi tarvitaan vielä kuuntelija kuuntelemaan toimintojen laukaisemisia. Lopuksi Form laitetaan ruutuun Display.setCurrent(Displayable)-metodilla.

Varsinaisessa sivujen katselu-tilassa käytetään omaa Canvasta, jolloin sovellus voi reagoida näppäimiin ja kosketustapahtumiin. Näin sovellus voi toteuttaa oman navigointilogiikan ja määrittää ulkoasun vastaamaan tv:stä tuttua teksti-tv:tä.

5.5.4 Tiedostojen lukeminen

MIDP ei tue tiedostojen käsittelyä, mutta Class-luokan getResourceAsStream(String)-metodilla saa InputStreamin tiedostoon, joka on sovelluksen mukana jar-paketissa. Näin saatua InputStreamia voi lukea kutsumalla read()-metodia, joka lukee yhden tavun (byte) tai read(byte[] b, int offset, int length)-metodia, joka lukee taulukkoon b length merkkiä, kohtaan offset eli taulukko täytetään indekseillä b[offset] … b [offset + length – 1], jos syötteessä on vähemmän kuin length merkkiä, lukeminen keskeytetään ja lopulta metodi palauttaa montako merkkiä luettiin, tai -1, jos päästiin syötteen loppuun. InputStreamin voi kääriä InputStreamReaderin sisään, joka tarjoaa samanlaisen rajapinnan kuin InputStream, mutta InputStreamReaderin read() voi lukea useamman kuin yhden merkin kerralla muistiin ja tarjoaa näin parempaa suorituskykyä. Toisena vaihtoehtona on kääriä InputStream DataInputStreamin sisään, jolloin syötteestä voidaan lukea Javan primitiivejä: boolean, byte, char, double, float, int, long, short tai String. Lisäksi voidaan lukea unsigned byte- ja unsigned short-tyyppistä dataa, mitä

tyyppejä ei ole Javassa, joten metodit palauttavat kokonaisluvun. Stringit tulkitaan aina UTF-8 koodatuiksi.

Teksti-TV-sovelluksessa käytetään molempia tapoja. DataInputStreamia käytetään lukemaan kirjasin-tiedostot ja InputStreamReaderilla luetaan käyttöehdot tiedostosta. Erona on, että InputStreamReaderille voi asettaa jonkin merkkikoodauksen lukemista varten, vaikka kyseistä ominaisuutta ei tässä sovelluksessa käytetä.

5.5.5 HTTP-yhteydet

MIDP:stä löytyy tuki http- ja https-yhteyksille. Yhteydet avataan javax.microedition.io.Connector-luokan metodilla open(), joille voi antaa parametreina URLin, saantitavan (luku, kirjoitus, molemmat) ja lipun, haluaako kutsuja poikkeuksia aikavalvontakatkaisuista (timeout). Open() palauttaa Connection-tyyppisen olion, jonka kutsuja sitten konvertoi oikeaan tyyppiin, http:n tapauksessa HttpConnectioniksi tai HttpsConnectioniksi. Yhteys voi olla kolmessa eri tilassa: alustus, yhdistetty tai suljettu. Alustus-tilassa voidaan asettaa pyyntöön sopivat otsikkotiedot metodeilla setRequestMethod() ja setRequestProperty(). SetRequestMethod()-metodilla asetetaan pyynnön tyyppi: GET, POST tai HEAD. SetRequestProperty():llä asetetaan pyynnölle muut otsikkotiedot, esimerkiksi pipareita, jotka eivät toimi automaattisesti, vaan ohjelmoijan täytyy muistaa itse lukea ja asettaa niitä tarvittaessa. Yhteys siirtyy yhdistetty-tilaan, kun käyttäjä avaa datayhteyden tai yrittää lukea jotain tietoja, jotka tulevat vastauksena pyyntöön. Esimerkiksi getResponseCode() ja getLength() aiheuttavat yhteyden avaamisen ja tilasiirtymän. Datan lukemista varten kutsutaan HttpConnectionin openInputStream()- tai openDataInputStream()-metodeita ja kirjoittamista varten (lähinnä POST-pyyntö) metodeita openOutputStream() tai openDataOutputStream(), joita käytetään vastaavasti kuin tiedostoja edellä. Lopulta kutsumalla metodia close() yhteys suljetaan, mutta spesifikaatio ei määrittele, mitä tapahtuu, jos nyt kutsutaan olion metodeita. Tosin metodien tulee heittää IOException, jos yhteyden muodostamisessa palvelimeen tulee jokin virhe.

Teksti-TV-sovelluksessa käytetään HttpConnectionia hakemaan palveluntarjoajalista palvelimelta ja niihin liittyvät ikonit. Varsinaisen käytön aikana sovellus käyttää omaa HttpCookieConnection-luokkaa, joka käärii HttpConnection-luokan sisäänsä ja lisää siihen piparituen, mitä käytetään käyttäjän tunnistamiseen.

5.5.6 Säikeet

Säiemalli on ensimmäisen Java-version mukainen, eli luokat voivat periä java.lang.Thread-luokan, jolloin säie käynnistetään kutsumalla metodia start() ja uusi säie aloittaa suorituksensa run()-metodista. Toinen tapa on, että luokka toteuttaa java.lang.Runnable-rajapinnan, jolloin säie käynnistetään luomalla uusi säie Thread t =

new Thread(Runnable), jonka jälkeen t.start() käynnistää säikeen suorituksen metodista run().

Teksti-TV-sovelluksessa säikeitä käytetään animaatioon, kun käyttäjä vaihtaa sivua, sivujen jäsentämiseen ja lataamiseen verkosta, sekä kellon näyttämiseen ruudusta. Säikeiden ajo keskeytetään, jos sovellus ei ole näkyvillä ruudulla.

5.6 Millainen Android on

Android vaikuttaa modernilta järjestelmältä, joka tarjoaa ohjelmoijalle paljon asioita valmiina, jolloin hän voi keskittyä itse sovellukseen enemmän kuin kaikenlaisten pienten komponenttien toteuttamiseen. Modernilla tässä tarkoitetaan, että Android tukee hyvin tämän hetken vaatimuksia, kuten kosketusnäyttöä ja eleitä. Verkkoyhteyksiä on helppo muodostaa. Toisaalta aloituskynnys on hieman korkeampi, koska pelkän lähdekoodin kirjoittamisen lisäksi täytyy myös opetella käyttämään XML:ää. Kehitysympäristö on toimiva ja erityisesti JUnitin mukanaolo on hieno asia, vaikkei kehitysprosessi noudattaisikaan testivetoista kehitystä (TDD, test-driven development).

5.6.1 AndroidManifest.xml

AndroidManifest.xml-tiedostossa kuvataan sovelluksen Activityt ja määritellään niiden ominaisuuksia. Teksti-TV-sovelluksen kannalta olennaisia ominaisuuksia ovat Activityn määritteleminen koko ruudun vieväksi ja että Activityt ovat kiinnostuneita laitteen orientaation vaihtumisesta. Orientaatio ei varsinaisesti ole kiinnostavaa, muuten kuin, että näin estetään sovelluksen uudelleen käynnistäminen, jos käyttäjä kääntää päätelaitetta. Näin logiikasta tulee yksinkertaisempi, koska tilaa ei tarvitse tallettaa uudelleenkäynnistystä varten, vaan käyttäjän selailuhistoria ja nykyinen sivu säilyy tallessa ilman mitään erikoistoimenpiteitä.

Yleisiä koko sovellukseen, ei siis vain yhteen Activityyn, liittyviä asetuksia AndroidManifestissa, oli minimiversion määritys, sovelluksen oikeudet ja merkkaaminen että sovellus ei vaadi kosketusnäyttöä, tämä sen takia että se toimisi myös Androidia pyörittävässä televisiossa.

Androidista käytettiin versiota Android 2.0, jonka API (rajapinta) level on 5. Uudempiakin versioita olisi ollut tarjolla, mutta ne eivät tarjonneet mitään lisäarvoa. Tämän lisäksi sovellus tarvitsee seuraavat oikeudet:

- android.permission.READ_PHONE_STATE, IMEI-numeron lukemiseen, jotta palvelin voi luoda käyttäjälle id:n, joka on sidottu IMEI:hin.
- android.permission.INTERNET, internettiin pääsyä varten jotta sovellus voi hakea teksti tv sivuja ja palveluntarjoajalistan.

- android.permission.ACCESS_NETWORK_STATE, tämän avulla sovellus voi kuunnella internet-yhteyden tilaa ja avata verkkoyhteydet automaattisesti heti kun yhteys on auki.

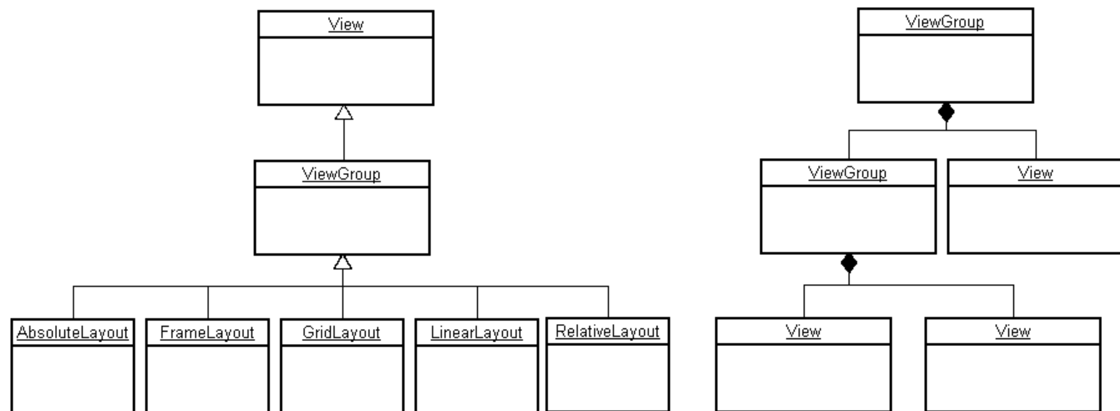
Liitteestä B löytyy koko AndroidManifest.xml, josta näkee, miten oikeudet ja Activityt määritellään.

5.6.2 Elinkaari

Teksti-TV-sovellus koostuu kahdesta Activitystä: ProviderActivitystä ja TeletextActivitystä. ProviderActivity näyttää palveluntarjoajalistan ja käyttäjän valittua, minkä tarjoajan teksti-tv:tä hän haluaa katsoa, ProviderActivity käynnistää TeletextAcivityn, joka on varsinainen teksti-tv-sovellus, eli siinä voi lukea ja selailla teksti-tv sivuja. Molemmat Activityt alustavat itsensä onCreate()-metodissa ja vapauttavat resurssit onDestroy()-metodissa. Lisäksi TeletextActivity pitää kirjaa, onko se pause-tilassa vai ei. Pause-tilassa ei päivitetä sivulla olevaa kelloa jatkuvasti, mutta muutoin kummankaan Activityn ei tarvitse välittää elinkaaren muista vaiheista. Toisen Activityn käynnistäminen tapahtuu luomalla android.content.Intent-luokan instanssi, jolle teksti-tv:n tapauksessa annettiin parametrina konteksti, eli this, ja käynnistettävä luokka TeletextActivity.class. Intentin putExtra()-metodeilla lisätään avain-arvo pareina parametrit, joista TeletextActivity tietää, mistä osoitteesta sivuja voi hakea ja mikä on aloitussivun numero. AddFlags()-metodilla määritellään, miten Activityt käyttäytyvät, eli voiko niitä olla monta päällekkäin ja tallentuuko Activity historia-pinoon [31]. Teksti-TV-sovelluksessa FLAG_ACTIVITY_CLEAR_TOP on sopiva lippu, koska haluttu toiminta on että ajossa on vain yksi TeletextActivity.

5.6.3 Käyttöliittymä

Android-sovelluksen käyttöliittymä rakentuu android.view.View- ja android-view.ViewGroup-olioista. View-olio huolehtii omalla alueellaan ruudulla omasta koostaan, asettelustaan, vierityksestään, tapahtumien käsittelystään, piirtämisestään ja kohdistetun elementin vaihdosta. ViewGroup kokoaa yhteen eri Viewejä ja asettelee ne ruutuun oman arkkitehtuurinsa mukaisesti. Kuvassa 5.4 on kuvattu osa perimishierarkiasta sekä koostesuhde.



Kuva 5.4. *Viewin ja ViewGroupin periytys- ja koostesuhteet.*

Viewistä on periytetty yli 60 aliluokkaa [31], joista löytyvät kaikki elementit, joita ruudulle yleensä laitetaan, kuten napit, tekstilaatikot ja valintalistat. Jos haluaa itse toteuttaa MIDP:n Canvasta vastaavan elementin, sen voi tehdä perimällä View-luokan ja toteuttamalla itse tarvitsemansa metodit.

View saa tiedon kuudesta erilaisesta tapahtumasta takaisinkutsu-mekanismin kautta, metodeita ovat:

- onClick(View v) kutsutaan, kun käyttäjä koskettaa komponenttia ruudulla tai navigoi siihen näppäimistöllä ja painaa ”ok” nappia tai painaa trackballia.
- onLongClick(View v) sama kuin onClick(View), mutta käyttäjä pitää valinnan pohjassa sekunnin ajan.
- onFocusChange(View v, boolean hasFocus) kutsutaan, kun käyttäjä navigoi komponenttiin tai pois komponentista näppäimistön tai trackballin avulla.
- onKey(View v, int KeyCode, KeyEvent event) kutsutaan, kun komponentti on valittuna ja käyttäjä painaa tai päästää napin ylös laitteistolla.
- onTouch(View v, MotionEvent event) kutsutaan, kun käyttäjä koskettaa ruutua, päästää irti ruudusta tai liikkuu ruudulla (sormella tai muuten).
- onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) kutsutaan, kun pitkä painallus (long click) aiheuttaa kontekstisidonnaisen menun rakentamisen ruudulle.

Näistä onLongClick():n, onKey():n ja onTouch():n täytyy palauttaa boolean-arvo, joka ilmaisee, onko tapahtuma käsitelty. Jos metodi palauttaa falsen, järjestelmä voi siten jatkaa tapahtuman käsittelyä ja mahdollisesti välittää tapahtuman muualle. Esimerkiksi jos onLongClick() palauttaa falsen, voidaan rakentaa menu ruudulle, jos sellainen oli määritelty.

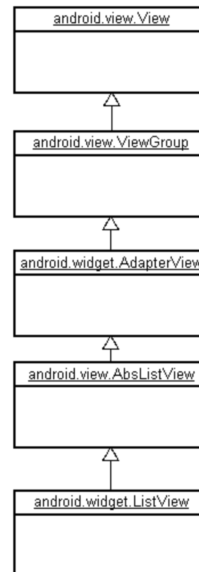
Itse toteuttaessa View-komponenttia voidaan toteuttaa seuraavat takaisinkutsumetodit:

- onKeyDown(int keyCode, KeyEvent event) kutsutaan, kun näppäimistön nappi painetaan alas.
- onKeyUp(int keyCode, KeyEvent event) kutsutaan, kun näppäimistön nappi vapautetaan.
- onTrackBallEvent(MotionEvent event) kutsutaan, kun trackballia liikutetaan.
- onTouchEvent(MotionEvent event) kutsutaan, kun käyttäjä koskettaa ruutua tai päästää ruudusta irti tai liikkuu ruudulla.
- onFocusChanged(boolean gainFocus, int direction, Rect previouslyFocusedRect) kutsutaan, kun View saa tai menettää kohdistuksen.

Näistä metodeista muiden, paitsi onFocusChanged():n, täytyy palauttaa boolean-arvo ilmaisemaan käsiteltiinkö tapahtuma vai ei.

Oman Viewin piirtäminen tapahtuu ylimäärittelemällä metodi onDraw(android.graphics.Canvas), jolloin View voi piirtää suoraan Canvakseen. Tässä oleva Canvas vastaa lähinnä MIDP:n Graphics-oliota, tosin rajapinta on erilainen ja on siis aivan eri asia kuin MIDP:ssä oleva javax.microedition.lcdui.Canvas.

Teksti-TV-sovelluksen ProviderActivityssä käytetään onItemClickListeneriä kuuntelemaan, minkä palveluntarjoajan teksti-tv sivut käyttäjä valitsee ohjelman käynnistyessä.



Kuva 5.5. *ListView-luokan isäluokat*

Palveluntarjoajalista näytetään android.widget.ListView näkymänä. Kuvasta 5.5 näkyy, että ListView on myös ViewGroup, jolloin se voi pitää toisia View-olioita sisällään ja näin ListViewin listassa näyttämät komponentit ovat myös kaikki View-luokan aliluokkia. Teksti-TV-sovelluksen tapauksessa komponentit ovat LinearLayoutin sisään ladottu ImageView ja TextView, joista ImageView näyttää kanavan logon ja TextView kana-

van nimen. OnItemClickListeneriä kutsutaan, kun käyttäjä valitsee jonkin komponentin listasta ja sovellus käynnistää toisen Activityn valittuun kanavaan liittyvillä parametreilla.

TeletextActivity käyttää omaa toteutusta Viewistä tv:stä tutun käyttöliittymän toteuttamiseksi ja sen lisäksi virtuaalinäppäimistöä varten käytetään KeyboardViewiä. Näiden saamiseksi yhtä aikaa ruutuun luodaan FrameLayout, johon asetetaan oma MyView ja KeyboardView, ja jolla on erilaiset ominaisuudet pysty- ja vaakatilaa varten; pystytilassa näppäimistö on ruudun alareunassa teksti-tv sivun alapuolella ja vaakatilassa näppäimistö on ruudun oikeassa laidassa, sen lisäksi näppäimien asettelu muuttuu hieman, jotta näppäimistö ei mene teksti-tv:n sisällön päälle. Tämä ei vaadi mitään toiminnallisuutta koodissa, vaan asettelu luetaan layout XML-tiedostoista, res/layout-land/ ja res/layout-port/ -hakemistoista automaattisesti, kun ruudun orientaatio muuttuu. Vastaavasti myös FrameLayout voidaan asettaa hakemaan halutun konfiguraation XML-tiedostoista, mutta sovelluksessa ei käytetä tätä tapaa, vaan orientaation vaihtuessa KeyboardView otetaan pois FrameLayoutista ja laitetaan uusi tilalle orientaatioon sopivien parametrien kanssa. Teksti-tv sivun piirtäminen tapahtuu onDraw(Canvas)-metodissa, jolloin ruudulle piirretään nykyinen sivu bittikarttana ja jos käyttäjä on ”raahaamassa” ruudulla tehdään translaatio ja piirretään kuva raahauksen verran sivuun horisontaalisesti tai vertikaalisesti.

5.6.4 Tiedostojen lukeminen

Sovellukset voivat lukea ja kirjoittaa tiedostoja laitteen sisäiseen tallennustilaan. Oletuksena tiedostot kuuluvat sovellukselle, eikä muut pääse niihin käsiksi. Nämä tiedostot tuhoutuvat, jos sovelluksen asennus poistetaan. Tiedostojen lukeminen tapahtuu Activityn perimän android.content.Context-luokan metodilla openFileInput(String name), joka palauttaa FileInputStreamin. Javan tapaan InputStreamia voi lukea suoraan, mutta yleensä se kääritään jonkin puskuroidun Readerin tai InputStreamin sisään. Näitä ovat esimerkiksi BufferedReader ja BufferedInputStream [31]. Puskurointi voi tarjota suorituskykyetua, koska ne voivat lukea syötettä etukäteen ja palauttaa datan puskurista. Tiedostoihin kirjoittaminen tapahtuu vastaavasti metodilla openFileOutput(String name, int mode), joka palauttaa FileOutputStreamin, jonne voi kirjoittaa suoraan tai kääriä Writerin tai toisen OutputStreamin sisään, esimerkiksi ObjectOutputStreamin olioiden tiedostoon sarjallistamista varten. Mode-parametrilla kerrotaan, missä moodissa tiedosto kirjoitetaan. Vaihtoehtoja ovat yksityinen (oletusarvo), tiedoston loppuun lisäys sekä tiedoston merkkaus kaikille muille sovelluksille lukemisen tai kirjoittamisen sallivaksi.

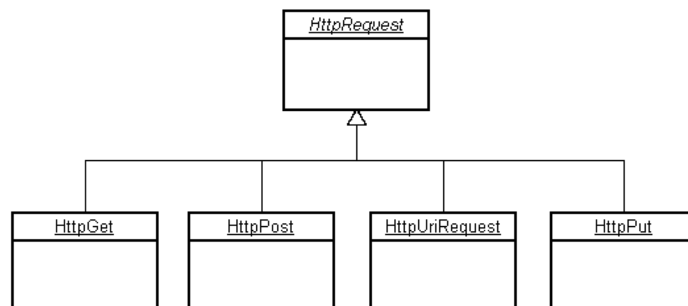
Teksti-TV-sovelluksessa tiedostoa käytetään vain käyttäjän palvelimelta saaman id:n tallentamiseen, jotta käyttäjä tunnistetaan myöhemmin ja palvelin voi tarkistaa, onko hänellä vielä käyttöoikeuksia. Käynnistyksessä id luetaan tiedostosta, ja jos sitä ei löydy oletetaan, että tämä on ensimmäinen kerta, kun sovellusta käytetään ja käytetään arvona

-1. Muuten id lähetetään palvelimelle pyyntöjen yhteydessä, jolloin palvelin voi tehdä tarkistukset. Kun palvelin lähettää sovelluksen oman yksikäsitteisen id:n, se kirjoitetaan yksityiseen tiedostoon tulevaisuutta varten.

5.6.5 HTTP-yhteydet

Resurssien hakemiseen verkon yli löytyy kaksi erilaista tapaa, ensimmäinen on käyttää java.net.URL-luokkaa ja toisena tapana on käyttää org.apache.http.client.HttpClient-rajapinnan toteuttavaa http-asiakas-luokkaa, joita on ainakin android.net.http.AndroidHttpClient ja org.apache.http.impl.client.DefaultHttpClient. Http-asiakkaan etuna on, että asetukset pysyvät eri kutsujen välillä voimassa ja asiakas osaa hoitaa piparit automaattisesti.

URL-luokka soveltuu yksittäisiin satunnaisiin hakuihin, mutta tarvittaessa sillekin voi asettaa itse pipareita tai muita http-otsikkotietoja. Yksinkertaisimillaan luodaan uusi URL-olio, jolla on parametrina haluttu osoite. Tämän jälkeen URL-oliolta pyydetään HttpURLConnection kutsumalla metodia openConnection() ja lopuksi InputStream saadaan HttpURLConnectionin getInputStream()-metodilla.



Kuva 5.6. *Osa HttpRequestin periytymishierarkiasta.*

HttpClient-instansseilla resurssin haku onnistuu execute()-metodeilla, joita on erimuotoisia riippuen annetaanko parametrina konteksti, vastauksenkäsittelijä tai erillinen isännän osoite erillisenä oliona. Tärkeimpänä parametrina tulee kuitenkin aina absoluuttisissa osoitteissa HttpUriRequest tai suhteellisissa osoitteissa HttpRequest, jotka sisältävät tiedot pyynnön tyypistä ja varsinaisen http-protokollan [50] mukaisen pyynnön, jonka voi kysyä getRequestLine()-metodilla. Protokollan eri pyyntöjä varten on eri toteutuksia, kuten HttpGet ja HttpPost kuvassa 5.6.

Teksti-TV-sovelluksessa käytetään DefaultHttpClienttiä, jolla haetaan palveluntarjoajalista, sekä varsinaiset teksti-tv sivut jäsentämistä varten. Pyyntö muodostetaan luomalla HttpGet instanssi, joka ottaa parametrina URL-instanssin, jonka parametrina haettava osoite on String-instanssina. HttpGet-olioon laitetaan User-Agent-otsake, joka sisältää

käyttäjän id:n, jonka jälkeen kutsutaan HttpClientin execute()-metodia parametreina HttpGet ja HttpContext-olio, joka sisältää CookieStoragen pipareita varten.

5.6.6 Säikeet

Androidissa on sama Thread-luokka ja Runnable-rajapinta säikeitä varten kuin Java SE:ssä ja MIDP:ssä. Androidin käyttöliittymäkomponentit paketeista android.widget ja android.view eivät ole säieturvallisia, joten Thread-instansseista ei voi suoraan manipuloida käyttöliittymän tilaa tai ulkoasua, vaan se täytyy tehdä UI-säikeestä. Thread-instanssista voi kutsua View-luokan metodeita post(Runnable) tai postDelayed(Runnable,long delayMillis), jotka aiheuttavat kutsun menemisen tapahtumajonoon ja käsittelyn aikanaan UI-säikeessä. Ongelmana tässä on, että metodit ottavat Runnablen eli uuden säikeen parametrina, joten käyttäjän täytyy vielä säikeen sisällä luoda uusi säie-instanssi tapahtumajonoa varten. Parempi ratkaisu on käyttää luokkaa android.os.AsyncTask<Params, Progress, Result>. AsyncTask ajaa laskennan säikeissä ja palauttaa tuloksen sovelluksen UI-säikeessä ilman, että sovelluksen tarvitsee tehdä mitään ylimääräistä tuloksen siirtämiseksi UI-säikeeseen. Sovellus vain toteuttaa oman AsyncTask luokan ja kutsuu sen instanssille execute()-metodia. AsyncTask toimii neljässä vaiheessa:

1. onPreExecute() kutsutaan UI-säikeessä, kun AsyncTask on käynnistetty execute()-metodilla. Tässä voidaan tehdä tarvittavat alustustoimenpiteet laskentaa varten.
2. doInBackGround(Params…) kutsutaan taustasäikeessä, kun onPreExecute():n suoritus on päättynyt. Params on luokan määrittelyssä määritelty tyyppi. Params tyyppiset parametrit tulevat parametrilistana, eli parametreja voi olla 0-n kappaletta. Tässä metodissa suoritetaan kaikki laskenta ja tarvittaessa voidaan kutsua publishProgress(Progress…)-metodia informoimaan käyttöliittymää laskennan edistymisestä. Laskennan valmistuttua palauttaa määritellyn Result-tyyppisen vastauksen.
3. onProgressUpdate(Progress…) kutsutaan jossain vaiheessa publishProgress()-kutsun jälkeen. Parametrit tulevat listana samoin kuin doInBackGround()-kutsussa, luokan määrittelyssä Progress-kohdassa määriteltynä tyyppinä.
4. onPostExecute(Result) kutsutaan UI-säikeessä sen jälkeen, kun doInBackground() palauttaa arvon, joka palautetaan tälle metodille parametrina.

AsyncTaskin voi keskeyttää kutsumalla cancel(boolean)-metodia. Jos parametrina annetaan true, säie voidaan keskeyttää, mutta muutoin suorituksen annetaan mennä loppuun asti. Jos cancel():ia on kutsuttu onPostExecute(Result):n asemesta kutsutaan onCancelled(Result)-metodia lopuksi. Mikäli metodit eivät tarvitse mitään parametreja voi tyyppinä käyttää Voidia, esimerkiksi seuraavasti: class MyTask extends AsyncTask<Void,Void,Void>

Teksti-TV-sovelluksessa käytetään AsyncTaskia lukemaan teksti-tv sivuja palvelimelta sovelluksen välimuistiin. Samalla, kun käyttäjä lukee sivua, kyseisen sivun alisivuja ja seuraavaa ja edellistä sivua haetaan verkon yli välimuistiin, mikäli niitä ei ole siellä vielä. AsyncTaskin metodissa doInBackground() tarkastetaan kohdistuiko pyyntö seuraavaksi näytettävään sivuun vai välimuistiin lataamiseen ja tämän perusteella aloitetaan joko lataamaan näytettävää sivua tai metadatan perusteella selvitettäviä muita sivuja. Lataamiseen jälkeen onPostExecute():ssa katsotaan, kumpi tapaus oli kyseessä. Mikäli ladattiin näytettävää sivua, päivitetään UI ja aloitetaan tämän sivun alisivujen ja seuraavan/edellisen sivun esilataus, muutoin jos vain välimuisti on päivittynyt, ei käyttöliittymää tarvitse päivittää. Erikoistapauksena jos palvelin palauttaa vastauskoodin 403, näytetään käyttäjälle palvelimelta saatu viesti, joka ilmaisee että käyttöoikeudet ovat loppuneet ja antaa ohjeet, miten niitä saa jatkettua.

6 JOHTOPÄÄTÖKSET

Työn tavoitteena oli tutkia siirrettävyyttä eri Java-ympäristöjen välillä. Javan alkuperäinen ajatus oli, että ohjelma kirjoitetaan kerran ja sitten ajetaan missä vain. Työssä toteutetun, olemassa olevan sovelluksen siirrossa oli ennakkoajatus, että siirtäminen olisi suoraviivaista rajapinnan korvaamista toisella vastaavalla rajapinnalla. Tämä ajatus ei toteutunut ja siihen löytyi varsin selvät syyt.

MIDP on osajoukko Java 1.1 -versiosta ja siihen on lisätty mobiililaitteissa tarvittavia rajapintoja. Android on lähtenyt Javan 1.5 -version osajoukosta ja siihen on lisätty mobiilirajapintoja ja toisaalta korvattu alkuperäisen Javan rajapintoja toisilla rajapinnoilla, sen lisäksi siihen on vielä tehty oma virtuaalikone, joka ei ole yhteensopiva Java-virtuaalikoneen kanssa. Androidia kehitettäessä on ollut mahdollista ottaa mallia MIDP:stä, koska se on ollut olemassa aikaisemmin, mutta ilmeisen tarkoituksella käyttöjärjestelmästä ei ole yritetty tehdä yhteensopivaa, vaan on yritetty tehdä parempi ja suorituskykyisempi ympäristö. Androidista tulee myös uusia versioita nopealla tahdilla, kun taas MIDP:n 3.0-versio on vuodelta 2009 ja vaikuttaa, että sen kehitys on lopetettu, eikä 3.0:aa tukevia laitteitakaan ole käytännössä markkinoilla.

Käytännön siirtotyön kohteena ollut Teksti-TV-sovellus siirtyi sinänsä aika helposti, koska yhteensopimattomat osat olivat kuitenkin melko pieniä ja lähdekooditasolla molemmat ympäristöt ovat Javaa, joten oli mahdollista siirtää ohjelman ydin, eli sivujen jäsennys, ilman muutoksia ympäristöstä toiseen. Ennakkoajatuksena ollut rajapintojen mekaaninen korvaus ei kuitenkaan toiminut ollenkaan. Käyttöliittymät ovat täysin erilaisia MIDP:ssä ja Androidissa, eikä selviä toiminnallisia vastaavuuksia löytynyt, joten toiminnallisuuden siirtäminen edellytti koko käyttöliittymän toteuttamista uudestaan. Http-yhteyksien käsittelyssä taas olisi ollut turhaa siirtää vanhaa toteutusta MIDP:stä, koska Android tarjoaa valmiina toimivan http-asiakkaan, jota voi käyttää sovelluksissa. Samoin tiedostojen käsittelyssä rajapinnat eroavat sen verran, että siirtämisen asemesta oli parempi tehdä uusi toteutus. Jos siirrettävyys olisi itseisarvo, voitaisiin ongelmalliset rajapinnat kääriä jonkin oman abstraktin rajapinnan taakse ja toteuttaa rajapinta kuhunkin ympäristöön sopivalla tavalla.

Jatkotutkimuksena voisi katsoa, miten siirtäminen onnistuu MIDP:stä tai Androidista BlackBerry-laitteille, jotka myös hyödyntävät Java-ympäristöä. Lisäksi mobiiliympäristöjä on muitakin, kuten Applen iOS, Microsoftin Windows Phone ja Samsungin Bada, joista mikään ei ole Java-ympäristö, mutta näidenkin välillä siirrettävyydestä olisi hyö-

tyä, jotta sama sovellus saataisiin mahdollisimman vähän vaivalla tuotettua mahdollisimman moneen eri ympäristöön. Tähän yksi ratkaisu voisi olla HTML5, jolloin sovellukset pyörivätkin päätelaitteen selaimessa. Näin alla oleva käyttöjärjestelmä ja ympäristö voisi olla mikä vain, riittää että löytyy selain, joka tukee HTML5:sta.

LÄHTEET

1. Java SE Overview - at a Glance. [Online] [Viitattu: 25. 2 2012.] http://www.oracle.com/technetwork/java/javase/overview/index.html.

2. Java EE at a Glance. [Online] [Viitattu: 25. 2 2012.] http://www.oracle.com/technetwork/java/javaee/overview/index.html.

3. Java ME Landing Page. [Online] [Viitattu: 25. 2 2012.] http://www.oracle.com/technetwork/java/javame/index.html.

4. *The K virtual machine (KVM).* [Online] [Viitattu: 25. 2 2012.] http://java.sun.com/products/cldc/wp/.

5. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 118. [Online] [Viitattu: 27. 2 2012.] http://www.jcp.org/en/jsr/detail?id=118.

6. *Open Handset Alliance.* [Online] [Viitattu: 25. 2 2012.] http://www.openhandsetalliance.com/.

7. Company. [Online] 25. 2 2012. http://www.google.com/about/company/.

8. *Teksti-TV sovellus - Sofiadigital.tv.* [Online] [Viitattu: 25. 2 2012.] http://sofiadigital.tv/info/teksti-tv.

9. Mooney, J.D. Bringing Portability to the Software Process. [Online] [Viitattu: 21. 2 2011.] http://www.cs.wvu.edu/~jdm/research/portability/reports/TR_97-1.pdf.

10. *Transportable programs for parallel and heterogeneous systems.* Pancake, C.M. 1994. System Sciences, 1994. Vol.II: Software Technology, Proceedings of the Twenty-Seventh Hawaii International Conference on. Vuosik. 2, ss. 581-585.

11. Input Method Framework Overview. [Online] [Viitattu: 6. 3 2011.] http://download.oracle.com/javase/7/docs/technotes/guides/imf/overview.html#Goals.

12. *Software Reuse.* Krueger, Charles W. 2, 1992, ACM Computing Surveys, Vuosik. 24, ss. 131-183.

13. wxWidgets. [Online] [Viitattu: 9. 3 2011.] http://www.wxwidgets.org/.

14. OpenGL - The Industry Standard for High Performance Graphics. [Online] [Viitattu: 30. 3 2011.] http://www.opengl.org/.

15. *Java Platform Standard Edition 7 Documentation.* [Online] [Viitattu: 25. 2 2012.] http://docs.oracle.com/javase/7/docs/index.html.

16. Java 2 Platform, Standard Edition (J2SE) 1.4.2. [Online] [Viitattu: 20. 2 2012.] http://www.oracle.com/technetwork/java/javase/index-jsp-138567.html.

17. Java(TM) 2 SDK Documentation. [Online] [Viitattu: 20. 2 2012.] http://docs.oracle.com/javase/1.3/docs/.

18. JSR 231: JavaTM Binding for the OpenGL® API. [Online] [Viitattu: 20. 3 2011.] http://www.jcp.org/en/jsr/detail?id=231.

19. JSR 226: Scalable 2D Vector Graphics API for J2METM. [Online] [Viitattu: 20. 3 2011.] http://jcp.org/en/jsr/detail?id=226.

20. Welcome to GPS.gov. [Online] [Viitattu: 30. 3 2011.] http://www.gps.gov/.

21. The Official Bluetooth® Technology Web Site. [Online] [Viitattu: 2011. 3 30.] http://www.bluetooth.com/Pages/Bluetooth-Home.aspx.

22. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 37. [Online] [Viitattu: 28. 2 2012.] http://jcp.org/en/jsr/detail?id=37.

23. Android Minimum Hardware Requirements. [Online] [Viitattu: 28. 2 2012.] http://www.talkandroid.com/android-forums/android-hardware/2-android-minimum-hardware-requirements.html#post56.

24. BlackBerry - Official BlackBerry - Tablets - Smartphones - Cell Phones - Mobile Phones - Apps at BlackBerry US. [Online] [Viitattu: 19. 2 2012.] http://us.blackberry.com/.

25. Java ME Technology. [Online] [Viitattu: 11. 8 2010.] http://www.oracle.com/technetwork/java/javame/tech/index-jsp-138655.html.

26. JSR-000139 Connected Limited Device Configuration 1.1 – Final Release . [Online] [Viitattu: 1. 8 2010.] http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html.

27. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 184. [Online] [Viitattu: 19. 2 2012.] http://jcp.org/en/jsr/detail?id=184.

28. *Comparative analysis of porting strategies in J2ME games.* Alves, V.;ym. 2005. Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on . ss. 123 - 132.

29. Learning Path: MIDlet Life Cycle. [Online] [Viitattu: 30. 7 2010.] http://developers.sun.com/mobility/learn/midp/lifecycle/.

30. Android Open Source. [Online] [Viitattu: 15. 8 2010.] http://source.android.com/.

31. *Android Developers.* [Online] [Viitattu: 15. 8 2010.] http://developer.android.com/index.html.

32. Camera | Android Open Source. [Online] [Viitattu: 28. 2 2012.] http://www.kandroid.org/online-pdk/guide/camera.html.

33. 2008 Google I/O Session Videos and Slides: Dalvik VM Internals. [Online] [Viitattu: 30. 7 2010.] http://sites.google.com/site/io/dalvik-vm-internals.

34. Welcome to JUnit.org! [Online] [Viitattu: 2. 8 2010.] http://junit.org.

35. *A Testing Method for Java ME Software.* Wang, Zhenglei & Du, Zhenjun & Chen, Rong. 25-27 Sept. 2009. Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on . ss. 58-62.

36. J2ME(TM) Connected Limited Device Configuration (CLDC) Specification 1.1 Final Release. [Online] [Viitattu: 26. 2 2012.] http://download.oracle.com/otndocs/jcp/7247-j2me_cldc-1.1-fr-spec-oth-JSpec/.

37. *Nokia Store: Download games, themes, wallpaper, ringtones and mobile apps on your Nokia phone.* [Online] [Viitattu: 27. 2 2012.] http://store.ovi.com/.

38. Apps on Android Market. [Online] [Viitattu: 27. 2 2012.] https://market.android.com/.

39. *Android.com.* [Online] [Viitattu: 27. 2 2012.] http://www.android.com/us/developer-content-policy.html.

40. Request for Comments: 2806. [Online] [Viitattu: 20. 7 2010.] http://www.ietf.org/rfc/rfc2806.txt.

41. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 75. [Online] [Viitattu: 25. 2 2012.] http://jcp.org/en/jsr/detail?id=75.

42. JAVA ME SDK DOWNLOAD. [Online] [Viitattu: 25. 2 2012.] http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/index.html.

43. Welcome to NetBeans. [Online] [Viitattu: 25. 2 2012.] http://netbeans.org/.

44. Eclipse. [Online] [Viitattu: 3. 8 2010.] http://www.eclipse.org/.

45. EclipseME Home Page. [Online] [Viitattu: 25. 2 2012.] http://eclipseme.org/.

46. MIDP Software & Tools. [Online] [Viitattu: 25. 2 2012.] http://developers.sun.com/mobility/midp/software/.

47. Symbian SDKs. [Online] [Viitattu: 26. 2 2012.] http://www.developer.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html.

48. Work Programme - Work Item Detailed Report. [Online] [Viitattu: 27. 2 2012.] http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=16166.

49. Antenna. [Online] [Viitattu: 19. 1 2012.] http://antenna.sourceforge.net/.

50. Hypertext Transfer Protocol -- HTTP/1.1. [Online] [Viitattu: 19. 2 2012.] http://www.ietf.org/rfc/rfc2616.txt.

LIITTEET

Liite A. Teksti-TV-sovelluksen jad-tiedosto

```
MIDlet-Jar-URL: [poistettu]
MIDlet-Jar-Size: 114248
MIDlet-Name: Teksti-TV
MIDlet-Vendor: Sofia Digital Ltd.
MIDlet-Version: 1.8.0
MIDlet-Permissions: javax.microedition.io.Connector.http
MIDlet-Permissions-Opt:
javax.wireless.messaging.sms.send,javax.microedition.midlet.MIDlet.pla
tformRequest.http,javax.microedition.midlet.MIDlet.platformRequest.sms
,javax.microedition.midlet.MIDlet.platformRequest.tel,javax.microediti
on.midlet.PlatformRequest.http,javax.microedition.midlet.PlatformReque
st.sms,javax.microedition.midlet.PlatformRequest.url,javax.microeditio
n.midlet.PlatformRequest.tel,javax.microedition.io.Connector.sms
Nokia-MIDlet-On-Screen-Keypad: no
Nokia-UI-Enhancement: PopUpTextBox
Nokia-MIDlet-S60-Selection-Key-Compatibility: true
Service-URL: http://sofiadigital.tv/txt/
MIDlet-Info-URL: http://sofiadigital.tv/
MIDlet-Certificate-1-1: [poistettu]
MIDlet-Certificate-1-2: [poistettu]
MIDlet-Certificate-1-3: [poistettu]
MIDlet-Jar-RSA-SHA1: [poistettu]
MIDlet-1: Teksti-TV, /teletext.png, fi.sfd.teletext.TeletextMidlet
Installer-ID: sofiadigital.tv
```

Liite B. Teksti-TV-sovelluksen AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.sofiadigital.teletext"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:label="@string/app_name"
android:debuggable="true" android:icon="@drawable/teletext" >
        <activity android:name=".ProviderActivity"
android:launchMode="standard"
android:configChanges="orientation|keyboard" >
          <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".TeletextActivity"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
                  android:label="@string/app_name"
android:launchMode="standard"
                  android:configChanges="orientation|keyboard">
             <intent-filter>
                <action android:name="android.intent.action.PICK" />
                <category
android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="5" />
    <uses-feature android:name="android.hardware.touchscreen"
android:required="false"/>
    <uses-permission
android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>

</manifest>
```