Kimmo Urtamo

Mobiiliohjelmointikielet ja niiden käyttöliittymämallit

Tietotekniikan kandidaatintutkielma

28. huhtikuuta 2015

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Kimmo Urtamo

Yhteystiedot: kimmo.j.urtamo@student.jyu.fi

Ohjaaja: Sanna Mönkölä

Työn nimi: Mobiiliohjelmointikielet ja niiden käyttöliittymämallit

Title in English: Mobile programming languages and their UI-patterns

Työ: Kandidaatintutkielma

Sivumäärä: 21+0

Tiivistelmä: Tässä tutkielmassa tutustutaan mobiilikieliin, älypuhelinsovellusten käyttöliittymämalleihin sekä mobiilisuunnittelussa käytettyihin prototyyppeihin. Mobiilikielistä esillä ovat Java, Objective-C ja Qt. Tutkielman aikana havaittiin, että ohjelmointikielissä työpöytäkielien ja mobiilikielien välillä ei ole suuria eroja. Suosittuja käyttöliittymämalleja mobiiliohjelmoinnissa ovat käynnistin ja lista. Käytettyjä prototyyppejä ovat paperiprototyyppi ja emulaattorin hyödyntäminen. Prototyyppejä myös yhdistellään, esimerkiksi siirtämällä kuvia käyttöliittymän osista puhelimeen selaamista varten.

Avainsanat: mobiili, ohjelmointi, ohjelmointikielet, objective-c, java, qt, käyttöliittymä, malli, prototyyppi

Abstract: The purpose of this thesis is to become acquainted with mobile languages, smart phone UI-patterns and the prototypes used in mobile design. Languages presented are Java, Objective-C and Qt. During the thesis it was observed that the differences between desktop languages and mobile languages are small. Also, popular UI-patterns in mobile programming are the launcher and the list. Prototypes used are the paper prototype and using an emulator. Prototypes can also combined, for example moving pictures of the user interface to a phone to browse.

Keywords: mobile, programming, programming languages, objective-c, java, qt, user interface, pattern, prototype

Termiluettelo

JVM	Java Virtual Machine, Java-kielen virtuaalikone, joka mahdollistaa koodin ajamisen käyttöjärjestelmäriippumattomasti.
Keko	Muistirakenne, jossa on juurisolmu ja tämän alla lapsia. Myös lapsisolmut voivat sisältää lapsia, muodostaen puun kaltaisen rakenteen.
Pino	Muistirakenne, jossa sijoitetaan objekteja päällekkäin. Kun pinosta poistetaan objekti, saadaan viimeksi sinne asetettu.
SDK	Software Development Kit, eli tarvittavat välineen ohjelmien kehittämiseen tietyllä alustalla.
API	Application Programming Interface, kokoelma rutiineja, protokollia ja välineitä sovelluskehitykseen.

Sisältö

1 Johdanto

Mobiililaitteet eroavat suuresti pöytätietokoneista. Näyttöala on pienempi, käyttöjärjestelmässä liikutaan kosketusnäytön avulla hiiren ja näppäimistön sijaan ja laskentatehoa on vähemmän. Lisäksi käyttötilanteet ovat yleensä erilaisia. Kun vietämme aikaa tietokoneen ääressä, syvennymme enemmän tekemiseemme. Puhelimen esiin kaivaessamme haluamme ehkä kuluttaa vain pienen hetken aikaa odottaessamme.

Koska tietokoneen ja puhelimen väliset käyttötilanteet ja ominaisuudet ovat niin erilaisia, niille täytyy myös suunnitella sovelluksia aivan eri ajatusmaailmalla. Suunnitteluratkaisut voivat muuttua näiden kahden välillä, pienet kosketusnäytöt voivat aiheuttaa haasteita sovellusten suunnittelijoille ja käyttäjät täytyy huomioida eri tavalla käyttöliittymäsuunnittelussa. Tässä tutkielmassa on tavoitteena selvittää, onko mobiilisovelluksille olemassa hyväksi havaittuja käyttöliittymämalleja ja mitä prototyyppejä alalla hyödynnetään. Samalla tutustutaan kahden suositun puhelimen, Apple iPhonen sekä Google Androidin ohjelmointikieliin Objective-C ja Java sekä käsitellään useilla eri alustoilla käytettävää Qt-ohjelmointikieltä. Tutkielman tavoitteena on, että sen lopussa tiedetään perusasiat käsitellyistä ohjelmointikielistä sekä tietää mahdollisista käyttöliittymämalleista ja alalla käytettävistä kehitysprototyypeistä.

Aihe valittiin, koska tarkoitukseni oli opetella Qt-kieli. Tähän mennessä ohjelmointikokemusta on vain työpöytäsovelluksista, joten mobiilikieliin ja puhelinsovellusten käyttöliittymiin tututustuminen antaa perustietoja mobiilikehityksestä ja mahdollistaa sen syvällisemmän tutkimisen ja oppimisen. Tutkielma noudattaa seuraavaa rakennetta: Aluksi käydään lyhyesti läpi kaksi ohjelmointikieltä, iPhonen Objective-C ja Androidin Java. Tämän jälkeen käsitellään muista erikseen Qt-ohjelmointikieltä ja tämän kevyempää versiota Qt Quickia. Lopuksi tutustutaan mobiilisovellusten käyttöliittymämalleihin sekä mobiilikehityksessä hyödynnettäviin prototyyppeihin.

2 Java ja Objective-C

Markkinoilla olevat puhelinmallit rakentuvat useille eri alustoille. Applella on oma iPhone-puhelimensa, Google tarjoaa Android-alustaansa puhelinvalmistajille ja Microsoft on keskittynyt omiin Windows Phone-puhelimiinsa. Lisäksi mukana on pienempiä yrittäjiä, kuten suomalainen Jolla, jonka puhelin perustuu sen omaan Sailfish-alustaansa. Tässä luvussa on kuitenkin keskitytty vain iPhoneen ja Androidiin.

Googlen Android-alustalla kehityskielenä on Java, mutta NDK:n (Native Development Kit) avulla myös C/C++ ja ARM-kehitys onnistuu. iPhone-kehitys tapahtuu suurimmilta osin Objective-C:llä, mutta myös sille pystyy luomaan sovelluksia C/C++:lla (Hammershoj, Sapuppo ja Tadayoni 2010). Seuraavissa alaluvuissa käsitellään kummastakin kielestä, Javasta ja Objective-C:stä, niiden perusteet.

Java-alaluku perustuu pääosin Java-ohjelmointia käsittelevään kirjaan (Parsons 2012) ja Objective-C-alaluku pääosin kirjaan, jossa Dovey ja Furrow (2012) ovat käsitelleet Objective-C-ohjelmointia.

2.1 Java

Javan alkuna pidetään yleisohjainta ”*7”, jonka tarkoituksena oli toimia kodin viihdelaitteiden ohjaimena. James Gosling kehitti sitä varten Oak-ohjelmointikielen, mutta laite ei saanut suosiota. Internetin noustua valtavirtaan Gosling muiden avustuksella alkoi kehittää HotJavaa, Oakiin perustuvaa Internet-selainta ja myöhemmin Netscape Corporation sisällytti Javan omaan Navigator-selaimeensa. Kieli oli käyttäjien keskuudessa suosittu ja sitä käytetään edelleen laajasti. Nykyään Java on jaettu kolmeen osaan kehittäjiä ajatellen: Standard Edition työpöytäsovelluksille, Enterprise Edition servereille ja Micro Edition puhelimille sekä muille pienille laitteille (Flynt ja Wells 2007).

Java on oliosuuntautunut kieli, eli sekä tila että käyttäytyminen on käärittyyhteen olioksi tarkoituksena helpottaa reaalimaailman mallintamista. Java-kielessä on paljon samoja ominaisuuksia kuin C++:ssa, mutta siitä on pyritty tekemään yksinkertaisempi ja vaarattomampi. Lisäksi Javasta on pyritty tekemään vakaa kieli. Siinä yksinkertaistetaan ohjelmoijan työtä

automatisoimalla tiettyjä asioita, kuten muistinhallintaa. Datatyyppien koot ovat tarkkaan määritelty, joka parantaa alustariippumattomuutta, kun esimerkiksi kokonaisluku ei ole yhdellä arkkitehtuurilla 8 bittiä ja toisella 16. Tiukka tyypitys myös vaikeuttaa tiedon tahallista korruptointia. Javassa ei ole myöskään C:stä tuttua osoitinta, jonka avulla ohjelmoija pääsisi suoraan käsiksi muistialueisiin. Manuaalinen muistin jakaminen voi johtaa ongelmiin, minkä takia se on Javasta poistettu. Myöskään muistin vapauttamista ei tarvitse tehdä käsin, sen hoitaa automaattinen roskienkeruu.

Java on alustariippumaton, mikä onnistuu JVM:n (Java Virtual Machine) avulla. Tämä toimii sekä kääntäjänä että tulkkina mahdollistaen koodin kirjoittamisen ja ajamisen kaikilla tuetuilla alustoilla. Tämä vaikeuttaa kuitenkin puhelimissa laiteresursseihin pääsyä ja mahdollisuus rautatason toimintojen hyödyntämiseen määräytyy käytössä olevan APIn mukaan (Hammershoj, Sapuppo ja Tadayoni 2010). Vaikka Android-ohjelmointia tehdään Javalla, Google kehitti siihen oman Dalvik-virtuaalikoneensa. Dalvik luo JVM:n tapaan ajotiedostoja, mutta niiden rakenne eroaa Javan jar-tiedostoista (Azimzadeh, Sameki ja Goudarzi 2012).

Yksinkertainen Java-ohjelma kirjoitetaan seuraavasti:

```
public class MyJavaProgram
{
  public static void main(String[] args)
  {
  System.out.println("My Java Program Running!");
  }
}
```

Ensimmäisellä rivillä määritellään käytettävä luokka. Tässä tapauksessa luokka näkyy kaikille muille luokille public-määritelmän vuoksi. Tämän jälkeen määritellään pääfunktio, joka on oltava jokaisessa luokassa. Lopuksi kutsutaan System.out-luokan tulostusmetodia ja tulostetaan haluttu merkkijono.

2.2 Objective-C

Objective-C-kielen loivat 1980-luvun alussa Brad Cox ja Tom Love ja sen tarkoituksena oli tuoda Smalltalk-kielen oliosuuntautuneen ohjelmoinnin käsite C-kielen puolelle. Vuosikymmenen loppupuolella kieli lisensoitiin NeXTin käyttöjärjestelmälle, johon myös kehitettiin Objective-C-kääntäjä ja ohjelmointirajapintoja. 2000-luvun alussa Apple otti kielen käyttöönsä Mac-tietokoneisiinsa ja lopulta vuonna 2008 iPhonen SDK:hon, joka aiheutti kielen yleistymisen.

Koska Objecive-C perustuu C-kieleen, siinä pätevät C:n perusteet, muun muassa:

- Se on staattisesti tyypitetty ja imperatiivinen kieli, jolla tarkoitetaan, että muuttujille pitää määrittää tyyppi ennen kuin niitä voidaan käyttää.
- Osoittimet toimivat C-kielen tapaan.
- Funktiot on määriteltävä koodissa ennen niiden käyttöä.

Lisäksi kielestä löytyy sille erityisiä ominaisuuksia. Ensimmäinen näistä ovat vahvat ja heikot olioviittaukset. Kielen roskienkeruun tekee ARC (Automatic Reference Counting), joka sijoittaa muistinhallintaa koodiin sitä käännettäessä. Se ei kuitenkaan ymmärrä katkaista viittaussilmukoita olioiden välillä. Koska muisti vapautetaan vasta, kun olioon ei viitata, vahvat viittaukset voivat aiheuttaa muistivuotoja. Heikot viittaukset poistavat tämän ongelman.

Toinen ominaisuus on nimeltään automaattivapautusvarasto (engl. *autorelease pool*). Siihen kerätään tulevaisuudessa poistettavat oliot. Varastoa käytetään lähinnä irroitettujen säikeiden poistossa ja vähentämään muistinkulutuspiikkejä. Esimerkiksi silmukassa voidaan vapauttaa muistia joka kierroksen jälkeen sen sijaan, että ajettaisiin ensin koko silmukka loppuun.

Viimeinen erikoisuuksista ovat lohkot (engl.*blocks*), jotka toimivat anonyymien funktioiden kehikkona. Vaikka ne on toteutettu C-tasolla ja tomivat C-koodissa, lohkot ovat Objective-C-olioita ja niille pätevät näiden ominaisuudet. Toisin kuin muut kielen oliot, lohkot luodaan pinomuistiin keon sijaan. Tämä tekee niiden luonnista kustannustehokasta, mutta jos ne halutaan säilyttää varmasti muistissa, kannattaa suorittaa kopiointi kekoon.

Objective-C tukee myös poikkeustenkäsittelyä, mutta sitä ei käytetä samaan tapaan kuin muissa kielissä. Poikkeukset ovat tarkoitettu todella poikkeaviin tai arvaamattomiin tilan-

teisiin, kuten kiintolevyn irtoamiseen kirjoittamisen aikana.

Hello World Objective-C-kielellä rakentuu seuraavasti (Knaster, Malik ja Dalrymple 2012):

```
#import <Foundation/Foundation.h>
int main (int argc, const char *argv[])
{
  NSLog (@"Hello, Objective-C!");
  return (0);
} // main
```

Ensimmäisellä rivillä haetaan otsikkotietoja Foundation-ympäristöstä. Tämän jälkeen alkaa pääfunktio, jossa tulostetaan haluttu teksti. @-merkki merkkijonon edessä tarkoittaa, että kyseessä on Objective-C:n oma merkkijono C-merkkijonon sijaan. Lopuksi ohjelma lopetetaan palauttamalla 0 onnistuneen suorituksen merkiksi (Knaster, Malik ja Dalrymple 2012).

3 Qt-kieli

Qt:n kehitti alunperin Trolltech-niminen yhtiö ja sen ensimmäinen julkinen versio ilmestyi vuonna 1995. Tarkoituksena oli kehittää kieli, jolla pystyisi kirjoittamaan C++- tai Java-ohjelman, joka toimisi eri käyttöjärjestelmillä koodia muuttamatta. Aluksi Qt tuki vain työpöytäkäyttöjärjestelmiä, mutta vuonna 2000 se julkaistiin sulautetuille järjestelmille ja vuonna 2006 Trolltech esitteli Linuxiin pohjautuvan matkapuhelimen. Vuonna 2008 Nokia osti Trolltechin ja esitteli Qt:n omille alustoilleen (Fitzek, Pennanen ja Rintamaki 2010). Nykyään kielen kehityksestä vastaa The Qt Company ja virallista tukea on saatavilla Windowsin ja Linuxin lisäksi Androidille, iOS:lle ja Windows Phonelle (Company 2015c).

Tietyssä mielessä Qt on alustariippumattomampi kuin Java. Koodi käännetään suoraan käyttöjärjestelmälle eikä välissä ole ylimääräistä kerrosta, kuten Javassa käytettävä virtuaalikone (Hammershoj, Sapuppo ja Tadayoni 2010). Tässä luvussa esitellään Qt:n perusrakenne ja sen sisarkieli Qt Quick sekä käydään läpi kielen tärkeimmät ominaisuudet. Seuraavat alaluvut perustuvat pääosin kirjaan, jossa Fitzek, Pennanen ja Rintamaki (2010) ovat käsitelleet mobiiliohjelmointia Qt-kielellä .

3.1 Perusrakenne

Qt-kehitys tapahtuu käyttäen C++-kieltä. Qt:n perusrakenteeseen kuuluvat koodissa käytettävien luokkien otsikkotiedot ja pääfunktio. Alla on esitetty koodi, jota voidaan käyttää, kun halutaan tulostaa Hello World Qt-kielellä.

```
#include<QApplication>
#include<QLabel>
int main(int argc,char* argv[]) {
QApplication a(arg c,arg v);
QLabel label("Hello World");
label.show();
return a.exec();
}
```

Ensimmäiset kaksi riviä ovat otsikkotiedot koodissa käytettäviin luokkiin, minkä jälkeen alkaa pääfunktio. Qt:ssa graafinen ohjelma ei toimi ilman QApplication-oliota, joten se luodaan pääfunktiossa ensimmäisenä. Tämän jälkeen luodaan label-objekti joka sisältää tulostettavan tekstin ja asetetaan se näkyväksi. Viimeisenä käynnistetään tapahtumasilmukka, joka hoitaa ohjelman tapahtumien välittämisen niitä tarvitseville olioille (Molkentin 2007).

3.2 Qt-oliomalli

Qt:n uudistukset rakentuvat oliomallin päälle, missä koodissa käytettävät oliot periytyvät QObject-luokasta. Uudistuksia tuodessa on kuitenkin pyritty säilyttämään C:n tehokkuus. Tärkeimmät Qt-oliomallissa lisätyt parannukset ovat:

- muistinhallinta
- signaalin ja lokeron konsepti
- ominaisuudet ja metatieto.

Olio-ohjelmoinnissa perinnällä tarkoitetaan käsitettä, jossa yksi luokka voi olla toisen erikoistapaus käyttäytyen kuitenkin samalla tavalla. Esimerkiksi neliö ja kolmio ovat muodon erikoistapauksia. Alaluokat voivat muuttaa yläluokan tarjoamia metodeja tai rajapintoja ja tarvittaessa metodit voidaan myös korvata kokonaan toisilla (Dovey ja Furrow 2012).

3.3 Muistinhallinta

Qt helpottaa ohjelmoijan työtä muistinhallinnan kanssa C:hen verrattuna. QObject-luokasta perityt oliot voidaan asettaa puumuotoon, minkä jälkeen Qt osaa solmuolioa poistettaessa automaattisesti hävittää tämän lapset. Automaattinen muistinhallinta kuitenkin vaatii, että oliossa luodut lapset ovat keossa. Pinoon esimerkiksi luokkakonstruktorissa luodut lapset hävitetään kääntämisen aikana (Molkentin 2007). QObject-olioiden luonti ja lapseksi lisääminen tapahtuu seuraavasti:

```
QWidget window;
QVBoxLayout* layout = new QVBoxLayout(&window);
QSpinBox* spinBox = new QSpinBox;
```

```
layout->addWidget(spinBox);
```

Yllä window-olio luodaan pinomuistiin. Tämän lapset luodaan käyttämällä *new*-sanaa, minkä seurauksena ne sijoitetaan kekoon vanhemman alle. Näin toimimalla roskienkeruu hävittää window-olion myötä myös muut koodissa luodut oliot.

3.4 Signaalin ja lokeron konsepti

Toisin kuin muiden kielten graafisten käyttöliittymien ohjelmoinnissa, Qt:ssa ei käytetä takaisinkutsuja tai tapahtumankuuntelijioita, vaikka niitä tuetaankin. Sen sijaan kielessä on käytössä signaalin ja lokeron konsepti (engl. *signal/slot*), missä olion metodi lähettää signaalin määrätyille vastaanottajille. Toisin kuin takaisinkutsuissa, yhteys katkaistaan heti, jos jompikumpi olio tuhotaan (Molkentin 2007).

Yksi takaisinkutsujen ongelma on, että niihin tarvitsee totetuttaa määrätty rajapinta. Lisäksi lähettäjän tarvitsee säilöä ja hallita osoittimia kaikille rekisteröidyille kuuntelijoille ja tarkistaa manuaalisesti, ovatko kuuntelijat olemassa ennen tiedon lähettämistä. C++ ei myöskään tee tyyppitarkistuksia funktio-osoittimia säilöessään. Toimintaa on pyritty signaalien ja lokeroiden avulla helpottamaan.

Alla olevassa koodissa on yksinkertainen esimerkki, missä yhdistetään napin painallus ohjelman sulkemiseen signaalia ja lokeroa hyödyntämällä.

```
QObject::connect(&button,SIGNAL(clicked()),&a,SLOT(quit()));
```

Kaksi ensimmäistä argumenttia määrittävät lähettäjän sekä lähetettävän signaalin. Toiset kaksi vastaanottajan ja signaalin vastaanottamisen jälkeen suoritetun metodin. Lähettäjä ja vastaanottaja ovat connect()-metodissa olioiden osoitteita, joten jos ne ovat jo valmiiksi osoittimia, &-merkkiä ei tarvita.

Signaali voidaan yhdistää moneen lokeroon ja yksi lokero voi vastaanottaa monta signaalia. Jos signaali lähetetään, mutta sille ei ole vastaanottajaa, mitään ei tapahdu. Tässä konseptissa lähettäjän ei tarvitse tietää, odottaako kukaan sen lähettämää signaalia eikä vastaanottajan tarvitse välittää, mistä signaali on peräisin. Kutsussa sekä signaali että lokero on kääritty

makroihin. Tämä tehdään, koska Qt olettaa, että funktioille lähetetetään merkkijonoja, mitkä täyttävät sen sisäiset standardit ja referoivat vastaavia lokeroita dynaamisesti. Kieli hoitaa kuitenkin näiden muodostuksen automaattisesti.

Myös parametreja voidaan lisätä, mutta niiden tyyppien täytyy täsmätä. Jos lokero ottaa vastaan merkkijonoja, siihen ei voida lähettää kokonaislukuja. Signaali voidaan kuitenkin yhdistää vähemmän parametreja ottavaan lokeroon, jolloin se ei välitä ylimääräisistä parametreista. Signal(int,double) voidaan siis lähettää lokeroihin slot(), slot(int) tai slot(int,double), muttei aiemmin mainitun perusteella lokeroon slot(double) (Molkentin 2007).

3.5 Ominaisuudet ja metatieto

Ominaisuudet ovat kolmas Qt-oliomallin tuoma uudistus. Ne voidaan ajatella jäsenmuuttujina, joita pystyy myös lisäämään ajon aikana ja perusoliolla näitä ovat esimerkiksi sijainti ja koko. Jokaisesta QObject-oliosta luodaan myös QMetaObject-instanssi, mikä sisältää tietoa olion luokasta, yläluokasta sekä metafunktioista.

3.6 Qt Quick

Qt Quickissa käytetään JavaScriptin kaltaista QML-kieltä (Rischpater 2013). QML on deklaratiivinen kieli, missä käyttöliittymät kuvaillaan visuaalisten komponenttien ja niiden välisten yhteyksien mukaan. Se suunniteltiin komponenttien dynaamiseen yhdistämiseen ja mahdollistaa helpon muokkaamisen ja uudelleenkäytön. QML:n syntaksi on JSON-tyylinen ja se tukee JavaScript-lausekkeita sekä dynaamista ominaisuuksien sidontaa (Company 2015b). Lisäksi koodiin pystyy tarvittaessa upottamaan C++-APIa hyödyntämällä C:tä (Company 2015d). Alla on yksinkertainen esimerkki Qt Quick-ohjelmasta (Rischpater 2013).

```
import QtQuick 2.0
Rectangle {
  width: 360
height: 360
Text {
```

```
text: qsTr("Hello World")
anchors.centerIn: parent
}
MouseArea {
anchors.fill:  parent
onClicked: {
Qt.quit();
 } } }
```

Aluksi koodissa ladataan import-lauseella QtQuick-moduuli käyttöön. Tämän jälkeen luodaan oliohierarkia, missä nelikulmion sisällä on tekstialue sekä hiirialue, jota painamalla ohjelma sulkeutuu. Nelikulmiosta tulee automaattisesti sen alla olevien olioiden vanhempi ja olioista nelikulmion lapsia (Company 2015a).

4 Käyttöliittymät

Älypuhelimille on muodostunut niiden ilmestymisen jälkeen toimivaksi havaittuja käyttöliittymämalleja. Ohjelmistojen suunnitelijoiden tarvitsee myös saada palautetta käyttöliittymien toiminnasta parantaakseen niitä ja tietotekniikan aloilla palautetta kerätään prototyyppien avulla. Tutustutaan alla käyttöliittymämalleihin sekä mobiilialan käyttämiin prototyyppeihin.

4.1 Käyttöliittymämallit

Kolme suurinta ongelmakohtaa mobiilikäyttöliittymän suunnittelussa ovat ruututilan käyttö, interaktiomekanismit sekä yleinen suunnittelu (Nilsson 2009). Asioita, joihin kannattaa käyttöliittymää tehdessä kiinnittää huomiota ovat muun muassa:

- elementtien esittäminen listassa
- tiedon ryhmittäminen
- tekstinsyötön mekanismit
- tilanhallinta ohjelmistonäppäimistön ollessa esillä
- toiminta puhelimen ollessa pysty-sekä vaakatasossa
- vuorovaikutus pitkien operaatioiden aikana.

Nilsson (2009) esittää artikkelissaan myös muita ongelmakohtia ja ratkaisuja niiden selvittämiseen. Esitelmien, jotka materiaalin pohjalta oli pidetty, palautteessa ongelmat olivat koettu olennaisina palautteenantajien työssä ja ratkaisuehdotukset oltiin nähty pääosin hyödyllisinä. Täten yllämainitut kohdat on syytä pitää mielessä käyttöliittymää luodessa.

Balagtas-Fernandez, Forrai ja Hussmann (2009) toteuttivat tutkimuksen, jonka tulokset voivat olla sovelluskehityksessä hyödyllisiä. Tutkimuksessa käyttäjäryhmä testasi kahta erilaista käyttöliittymää. Toinen käyttöliittymä oli vieritettävä, päivämäärä syötettiin näppäimistön kautta ja asetuksiin pääsi puhelimen valikkonapin avulla. Toisessa taas oli välilehtiä, modaalinen päivänvalinta ja asetukset olivat kontekstimenun alla. Käyttäjät suosivat näistä vaihtoehdoista käyttöliittymää, jossa oli välilehtiä, asetuksiin pääsi puhelimen napin kautta ja

jossa päivämäärä syötettiin näppäimistön avulla.

Mendoza (2013) esittää seitsemän perusmallia, joista lähteä rakentamaan käyttöliittymää. Kolme yleisintä on lueteltu alla ja erikoisemmissa malleissa hyödynnetään mm. puhelimen kiihtyvyysanturia ja kameraa.

- Käynnistin. Tämä perustuu eri puhelinmallien kotivalikkoon, mikä tekee siitä helpomman käyttää. Se on jaettu kahteen osaan, otsakkeeseen ja työalueeseen. Otsake-alueelle sijoitetaan ohjelman navigointi. Työalueelle sijoitetaan ikoneja, joista pystyy ajamaan ohjelman eri osasia.
- Tarjotin. Tämäkin on jaettu kahteen osaan: painikkeeseen, joka on selvästi eroteltu muusta sisällöstä ja painiketta painettaessa näkyville tulevasta alueesta, ns. tarjottimesta. Tätä aluetta voi käyttää mm. navigointiin, asetusten esittämiseen tai tarjota käyttäjille työkaluja.
- Lista. Tässä tarkoituksena on jakaa sisältö useammalle sisäkkäiselle sivulle. Ensimmäisellä sivulla sijaitsee yleisluontoinen lista sovelluksen sisällöstä. Jokin alkio valittaessa siirrytään seuraavalle sivulle, jossa sijaitsee kyseisen alkion alalista. Kolmannella sivulla sijaitsee itse asian ydin, esimerkiksi kuva tai tuote.

Mallit, jotka Mendoza (2013) esittelee ovat laajasti käytössä sovelluksissa. Sahami Shirazi ym. (2013) kävivät koneellisesti läpi 400 suosituimman ilmaisen Android-sovelluksen käyttöliittymän rakennetta. Havainnoissaan Sahami Shirazi ym. (2013) toteavat, että kaksi suosituinta rakennetta olivat useampi pohja sisäkkäin ja pohja, minkä sisään oli sijoitettu kuvia ja tekstiä. Käynnistin ja lista ovat siis suosittuja malleja, sillä ne olivat laajasti edustettuina suosittujen Android-sovellusten keskuudessa. Ne ovat siis hyvä valinta käyttöliittymän pohjaksi.

Käyttöliittymä voidaan myös suunnitella käyttäjän ehdoilla. Holtzblatt (2005) ryhmineen toteutti pesäpallostatistiikkasovelluksen hyödyntäen kontekstuaalista suunnittelua (engl. *Contextual Design*).Tämä on asiakaslähtöistä suunnittelua, missä kehitetään sovellusta keräämällä kohderyhmästä tietoja haastattelemalla heitä luonnollisessa ympäristössään ja luomalla sovellusvaatimukset näiden tietojen pohjalta. Suunnittelijat kävivät läpi muutaman haastatteluiteraation ja käyttäjät pitivät lopullista sovellusta helppokäyttöisenä ja intuitiivisena.

4.2 Prototyypit

Mendoza (2013) esittää, että kolme prototyyppien käyttämisen syytä ovat palautteen saaminen sivujen välillä liikkumisesta, käyttöliittymän vuorovaikutuksesta ja laitevuorovaikutuksesta. Prototyyppi ei mobiilikäyttöliittymäsuunnittelussa välttämättä kuitenkaan tarkoita ohjelmaprototyyppiä, joita myös käytetään, vaan se voi tarkoittaa esimerkiksi paperiprototyyppiä, jossa tarkoituksena on piirtää jokainen sovelluksen ruutu erilliselle paperille. Tämä tuottaa nopeasti palautetta, on helppo toteuttaa ja taipuu hyvin improvisaatioon.

Mendoza (2013) toteaa, että paperiprototyyppi voidaan myös siirtää työpöydälle hyödyntämällä esityöstyökaluja. Työpöydällä voidaan myös hyödyntää kehitysympäristöjen emulaatio-ohjelmia. Ne ovat kuitenkin rajoitettuja, suorituskyvyltään huonoja ja tukevat vain perustoimintoja.

Prototyyppejä voidaan myös yhdistää. Bolchini, Pulido ja Faiola (2009) esittävät artikkelissaan prototyyppimallin, missä paperiprototyypistä otetaan valokuvia, jotka siirretään puhelimelle. Tämän jälkeen kuvia voi selata puhelimella ja sovelluksen toiminnasta saa paremman käsityksen. Mendoza (2013) ehdottaa myös Internet-sivun luomista, jolloin prototyyppiin saadaan interaktiivisuutta.

Ince ja Yengin (2014) suorittivat käyttäjäryhmällä tutkimuksen, missä ryhmä kokeili eri prototyyppejä. Tarkasteltavat prototyypit olivat paperiprototyyppi, tietokonesimulaatio ja ohjelma itse laitteella. Käyttäjien mielestä tietokonesimulaatio oli näistä käytettävin. Toisaalta Virzi, Sokolov ja Karis (1996) toteuttivat tutkimuksen, jossa tarkasteltiin käyttöliittymäongelmien havaitsemista eri prototyyppejä käytettäessä. Käyttöliittymäongelmien havaitsemisen määrässä yksinkertaisen ja monimutkaisen prototyypin välillä ei kuitenkaan havaittu merkittäviä eroja. Täten olisi kannattavaa valita käytettävät prototyypit niiden toteuttamisen yksinkertaisuuden mukaan, jolloin puhelinta hyödyntävät kuvaprototyypit nousevat esiin yhdistäessään yksinkertaisuuden ja havainnollistavuuden.

5 Yhteenveto

Mobiilikielten opettelu ei olekaan niin vaativaa, kuin tutkielmaa aloittaessa oletettiin. On suuri mahdollisuus, että Java on mobiiliohjelmointiin vaihtavalle ohjelmoijalle tuttu jo työpöytäpuolelta. Objective-C ja Qt on rakennettu C++:n päälle ja molempiin on lisätty omanlaisiansa parannuksia, joten jos C++ on hallussa, myös nämä kielet luonnistuvat helposti. Lisäksi iPhone ja Android tukevat C/C++ -kehitystä, mikä helpottaa siirtymistä näiden alustojen kehittäjäksi. Qt-lähdekoodia pystyy myös hyödyntämään monella alustalla, kunhan sen kääntää uudelleen ja Javalla kehittäessä virtuaalikone huolehtii yhteensopivuudesta.

Toimivia käyttöliittymämalleja ovat käynnistin ja lista. Myös tarjotin on toimiva malli ja lisäksi käyttöliittymää kehittäessä on hyvä pitää mielessä, miten ruututilan käyttö ja interaktiomekanismit aiotaan toteuttaa. Sovellus voidaan toteuttaa myös kontekstuaalista suunnittelua, jossa käyttäjähaastattelujen avulla luodaan sovellusvaatimukset ja toteutetaan ohjelma niiden pohjalta.

Mobiilialalla käytettäviä prototyyppejä ovat paperiprototyyppi ja ohjelmaprototyyppi. Halutessaan voi myös hyödyntää puhelinemulaattoreita. Prototyyppejä voi myös yhdistää siirtämällä kuvia ohjelman ruuduista puhelimeen, mikä saattaa parantaa niiden käyttömukavuutta. Ongelmien havaitsemiseen prototyypin monimutkaisuudella ei kuitenkaan näytä olevan vaikutusta, joten suunnittelun alkuvaiheessa on kannattavaa hyödyntää yksinkertaisia prototyyppejä, siirtyen kehityksen edetessä tarvittaessa monimutkaisempiin.

Jatkotutkimuskohteita voisi olla esimerkiksi akunkeston parannuksen tutkiminen, sillä aika, minkä puhelin pysyy päällä, vaikuttaa käytettävyyteen. Lisäksi koodin nopeutuksen tarkastelusta voi hyötyä, sillä ihmiset eivät käytä pitkiä aikoja puhelinsovellusten kanssa, jolloin nopea toimivuus on tarpeen. Johonkin tässä esitettyyn kieleen paneutuminen syvällisemmin on myös mahdollisuus. Tässä tutkielmassa ei myöskään käsitelty kaikkia mobiiliohjelmointikieliä. Microsoftin puhelimille ohjelmointiin tutustuminen on siis mahdollinen jatkotutkimusaihe.

Lähteet

Azimzadeh, E., M. Sameki ja M. Goudarzi. 2012. “Performance analysis of Android underlying virtual machine in mobile phones”. Teoksessa *Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on,* 292–295. Syyskuu. doi:10.1109/ICCE-Berlin.2012.6336470.

Balagtas-Fernandez, Florence, Jenny Forrai ja Heinrich Hussmann. 2009. “Evaluation of User Interface Design and Input Methods for Applications on Mobile Touch Screen Devices”. Teoksessa *Human-Computer Interaction – INTERACT 2009,* toimittanut Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, RaquelOliveira Prates ja Marco Winckler, 5726:243–246. Springer Berlin Heidelberg. doi:10.1007/978-3-642-03655-2_30.

Bolchini, Davide, Diego Pulido ja Anthony Faiola. 2009. “"Paper in Screen"Prototyping: An Agile Technique to Anticipate the Mobile Experience”. *interactions* (New York, NY, USA) 16, numero 4 (heinäkuu): 29–33. doi:10.1145/1551986.1551992.

Company, The Qt. 2015a. *First Steps With QML | Qt 5.4.* Saatavilla WWW-muodossa, http://doc.qt.io/qt-5/qmlfirststeps.html, viitattu 14.4.2015.

———. 2015b. *Qt QML 5.4.* Saatavilla WWW-muodossa, http://doc.qt.io/qt-5/qtqml-index.html, viitattu 14.4.2015.

———. 2015c. *Qt - Qt for Application Development.* Saatavilla WWW-muodossa, http://www.qt.io/application-development/, viitattu 22.4.2015.

———. 2015d. *Qt Quick 5.4.* Saatavilla WWW-muodossa, http://doc.qt.io/qt-5/qtquick-index.html, viitattu 14.4.2015.

Dovey, James, ja Ash Furrow. 2012. *Beginning Objective-C.* Berkeley, CA: Apress. ISBN: 978-1-4302-4368-7. doi:10.1007/978-1-4302-4369-4.

Fitzek, Frank H.P., Harri Pennanen ja Jarmo Rintamaki. 2010. *Qt for Symbian.* Hoboken, NJ, USA: John Wiley & Sons.

Flynt, John P., ja Martin J. Wells. 2007. *Java ME Game Programming.* Thomson Course Technology. ISBN: 9781598633894. http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=263353&site=ehost-live.

Hammershoj, Allan, Antonio Sapuppo ja Reza Tadayoni. 2010. "Challenges for mobile application development". Teoksessa *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference on,* 1–8. Lokakuu. doi:10.1109/ICIN.2010.5640893.

Holtzblatt, Karen. 2005. "Customer-centered design for mobile applications". *Personal and Ubiquitous Computing* 9, numero 4 (heinäkuu): 227–237. http://search.proquest.com/docview/208975153?accountid=11774.

Ince, IbrahimFurkan, ja Ilker Yengin. 2014. "Subjective Evaluation of Prototypes and Task Complexities for Mobile Phone Usability". Teoksessa *Intelligent Computing Theory,* toimittanut De-Shuang Huang, Vitoantonio Bevilacqua ja Prashan Premaratne, 8588:582–593. Springer International Publishing. doi:10.1007/978-3-319-09333-8_64.

Knaster, Scott, Waqar Malik ja Mark Dalrymple. 2012. *Learn Objective-C on the Mac.* Apress. doi:10.1007/978-1-4302-4189-8.

Mendoza, Adrian. 2013. *Mobile User Experience : Patterns to Make Sense of It All.* Morgan Kaufmann. ISBN: 9780124095144. http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=527073&site=ehost-live.

Molkentin, Daniel. 2007. *The Book of Qt 4 : The Art of Building Qt Applications.* No Starch Press. http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=440025&site=ehost-live.

Nilsson, Erik G. 2009. "Design patterns for user interface for mobile applications". *Advances in Engineering Software* 40 (12): 1318–1328. http://www.sciencedirect.com/science/article/pii/S0965997809000428.

Parsons, David. 2012. *Foundational Java.* Springer London. ISBN: 978-1-4471-2478-8. doi:10.1007/978-1-4471-2479-5.

Rischpater, Ray. 2013. *Application Development with Qt Creator.* Olton, Birmingham, GBR: Packt Publishing Ltd.

Sahami Shirazi, Alireza, Niels Henze, Albrecht Schmidt, Robin Goldberg, Benjamin Schmidt ja Hansjörg Schmauder. 2013. “Insights into Layout Patterns of Mobile User Interfaces by an Automatic Analysis of Android Apps”. Teoksessa *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems,* 275–284. London, United Kingdom: ACM. doi:10.1145/2494603.2480308.

Virzi, Robert A., Jeffrey L. Sokolov ja Demetrios Karis. 1996. “Usability Problem Identification Using Both Low- and High-fidelity Prototypes”. Teoksessa *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 236–243. Vancouver, British Columbia, Canada: ACM. doi:10.1145/238386.238516.