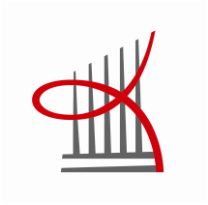


ANDROID-OHJELMISTON MODULARISOINTI

Mishka Reinman

Opinnäytetyö
Helmikuu 2015
Tietotekniikka
Sulautetut järjestelmät

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Sulautetut järjestelmät

MISHKA REINMAN:
Android-ohjelmiston modularisointi

Opinnäytetyö 31 sivua, joista liitteitä 3 sivua
Helmikuu 2015

TRESTIMA metsänmittausjärjestelmä on kehitetty korvaamaan yli 60 vuotta käytössä olleet perinteiset metsänmittaustyökalut. Opinnäytetyössä kerrotaan Trestima OY:lle tehdystä modulaarisesta Android-sovelluksesta ja sen julkisista rajapinnoista. Varsinaisen ohjelmoinnin lisäksi työssä esitellään modularisoinnin tarkoitus, hyödyt ja haitat sekä modularisointi Java-ohjelmointikielessä. Opinnäytetyössä esitellään myös TRESTIMA metsänmittausjärjestelmä, josta modulaarinen versio tehtiin. Työhön kuuluu myös ohjelmiston dokumentaation esittely.

Ohjelmointityötä varten annettiin tavoite valmiille sovellukselle, ohje rajapinnan toiminnoista, ja kehitystä seurattiin viikottaisissa palavereissa. Android-sovelluksen modularisointi vaati paljon uusien asioiden opettelua, muun muassa dynaamisten viittausten osalta. Tavoitteena oli helposti käytettävä moduli, jota voidaan käyttää muissa Android-sovelluksissa. Valmiin moduulin lisäksi jokainen metodi oli dokumentoitava kattavasti sekä esitettävä niistä esimerkkejä.

Opinnäytetyössä esitellään testikäyttöön valmistunutta modulaarista versiota, jonka metodit saattavat muuttua ohjelmiston julkaisun jälkeen. Modulaarinen ohjelmisto voidaan liittää mihin tahansa Android-sovellukseen ja sillä voidaan mitata puustoa TRESTIMA-palvelun avulla, sekä saada metsäraportteja ohjelmistoa käyttävään laitteeseen. Sovelluksen käyttö edellyttää kuitenkin modulaarisen ohjelmiston mukana tulevan dokumentaation huolellista seuraamista.

Asiasanat: modularisointi, Android, Java, Trestima OY, ohjelmointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information technology
Embedded systems

MISHKA REINMAN:
Modularization of an Android application

Bachelor's thesis 31 pages, appendices 3 pages
February 2015

TRESTIMA forest inventory system is a mobile application meant to replace tools that have been in use for over 60 years. This thesis will explain the basics of modular programming and how it works in Java programming language. Thesis will also present the modularization process of an Android application and how it is documented.

The need for a modular application came from the Trestima company. Guidelines for the methods to be used in the software were given and development process and more ideas were planned in the weekly meetings. Modularization process required quite a lot of studying of new things, for example how to make resource declarations dynamically. The goal was to create a module that could be used in any Android application with little to none effort.

The thesis will introduce the modularized application which is completed for testing purposes. Many of the methods may change after the full release. This modular application can be added to any existing application and it can be used for forest inventory like the original TRESTIMA application. Reports can be shown in any device that uses the modular version in their devices. Adding the modular program to existing application requires strict following of the documentation included in the module.

Key words: modularization, Android, Java, Trestima Ltd, programming

SISÄLLYS

LYHENTEET JA TERMIT

Java	Laitteistoriippumaton oliopohjainen ohjelmointikieli
GPS	Global Positioning System, eli maailmanlaajuinen sateliittiperustainen paikallistamisjärjestelmä
XML	Extensible Markup Language, eli merkintäkieli, jossa tiedon merkitys kuvataan tiedon sisällä ja jota voidaan käyttään tiedon välittämiseen
Eclipse	Ohjelmointiympäristö, joka tukee mm. Java-ohjelmointikieltä

1 JOHDANTO

Opinnäytetyössä esitellään toimeksiantaja Trestima OY ja TRESTIMA metsänmittausjärjestelmä, sekä ohjelmoinnin modulaarisuusajattelu. Lisäksi työssä tehdään olemassa olevasta Android-sovelluksesta modulaarinen versio ja esitellään valmiin modulaarisen ohjelman toimintaa käyttöönotosta dokumentaatioon.

Korkeakoulujen ohjelmointiopetukseen kuuluu erilaisten luokkien luonti ja käyttö, josta ei ole pitkä matka modulaarisuuteen. Luokkatiedostot on helppo jakaa toiminnallisuuksien mukaan moduuleihin ja ohjelmien luominen funktio kerrallaan yksinkertaistaa paloittelua. Modulaarisuusajattelu oli tuttua jo ennen opinnäytetyön tekemistä ja sitä on hyödynnetty pienissä sulautettujen järjestelmien projekteissa, joissa erilaisten toimintojen ohjaus on jaettu omiin luokkiinsa, joita kutsutaan rajapinnan kautta. Ideana on, että samoja rajapintoja voidaan käyttää myös muissa sovelluksissa.

Opinnäytetyö on tehty kirjoitushetkellä julkaisemattomasta modulaarisesta ohjelmistosta, joten työssä esitetyt metodit ja niiden toiminnallisuudet saattavat muuttua kehitystyön jatkuessa. Opinnäytetyön työosuus perustuu modulaarisen ohjelmiston versioon v0.90, joka on luotu 18.12.2014.

2 TRESTIMA OY

Trestima OY on vuonna 2012 perustettu Tamperelainen teknologiayritys. Yrityksen päätoimiala on ohjelmistotuotanto ja palvelut erityisesti metsäteollisuudelle. Yrityksen erityisosaaminen painottuu konenäköön, pilvipalveluihin, paikkatietoon ja mobiiliteknologiaan. Trestima OY:n merkittävin tuote on TRESTIMA metsänmittausjärjestelmä.

2.1 TRESTIMA metsänmittausjärjestelmä

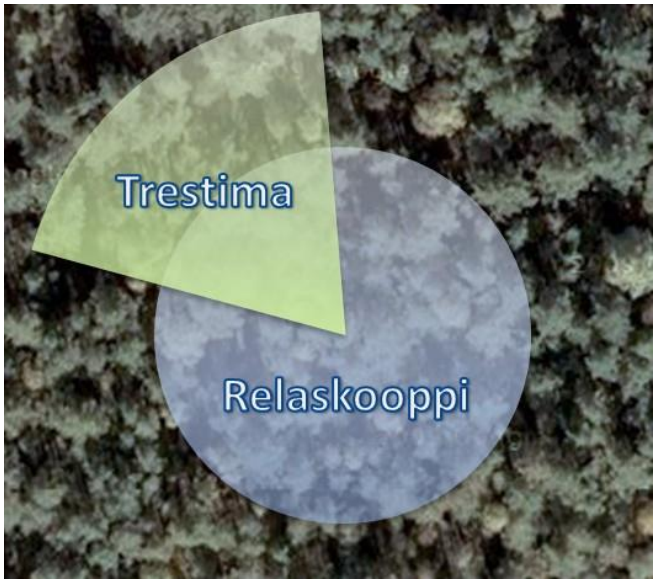
TRESTIMA metsänmittausjärjestelmä on tehokas ja tarkka työkalu metsän inventointiin. Metsänmittaussovellus ja lisävarusteena saatava TRESTIMA-mittakeppi korvaavat perinteisesti metsänmittauksessa käytetyt työkalut, kuten relaskoopin, mittasakset ja hypsometrin. TRESTIMA metsänmittausjärjestelmällä voidaan mitata puiden pohjapinta-alaa, runkolukua, korkeutta, läpimittaa, tilavuutta, peittokatvetta sekä metsän arvoa. (Trestima 2013.)

Sovellus on käytettävissä Windows Phone sekä Android –käyttöjärjestelmillä, joko älypuhelimilla tai tablet-laitteilla. Sovelluksen käyttöön vaaditaan laitteelta kameraa, GPS- ja kiihtyvyyssensoria sekä gyroskooppia laitteen asennon tunnistamiseksi. Asentotiedolla määritellään kuvaustilanteessa kuvan tyyppi, eli onko se pystykuva vai pohjapinta-alaa mittaava kuva.

2.2 TRESTIMA-sovellus

Käyttäjän ottama kuva tallennetaan mobiililaitteeseen sekä mahdollisuuksien mukaan lähetetään TRESTIMA-pilvipalveluun, jossa ihmisavusteisen hahmontunnistuksen avulla valokuvasta tunnistetaan rungot ja lasketaan niiden perusteella puustotunnistusraportit. Lasketut tulokset välitetään takaisin mobiililaitteeseen ja ne päivittyvät sovelluksen kartalle automaattisesti. Järjestelmällä voidaan saavuttaa alle 5 % keskivirhe puuston pohjapinta-alalle ja mittauksen tarkkuutta voidaan parantaa kuvien määrää kasvattamalla. (Trestima 2013.)

Metsänmittausjärjestelmä käyttää laitteen kameralla otettua kuvaa ja laskee yksittäisistä kuvista puustoarvot. Yksittäisestä kuvasta tunnistettavat rungot mitataan tarkemmin ja järjestelmä mahdollistaa perinteistä relaskooppia pienempien ja suurempien runkojen tarkan tunnistamisen. Perinteisellä relaskoopilla mitattaessa hahlon täyttävä runko lasketaan relaskooppikertoimen mukaan pohjapinta-alaan ja näin lasketaan yhteen täyden ympyrän alueelta mitatut rungot (Anttonen & Sopanen 2002.). TRESTIMA metsänmittausjärjestelmällä mitattaessa on otettava huomioon, että yksittäinen kuva on ympyrää kapeampi segmentti ja kuvia on otettava useampi, jos halutaan mitata rungot samalta ympyrän alueelta. Relaskoopin ja TRESTIMA metsänmittausjärjestelmän yhden mittauksen mittausalue on esitetty kuvassa 1. (Rouvinen 2014.)

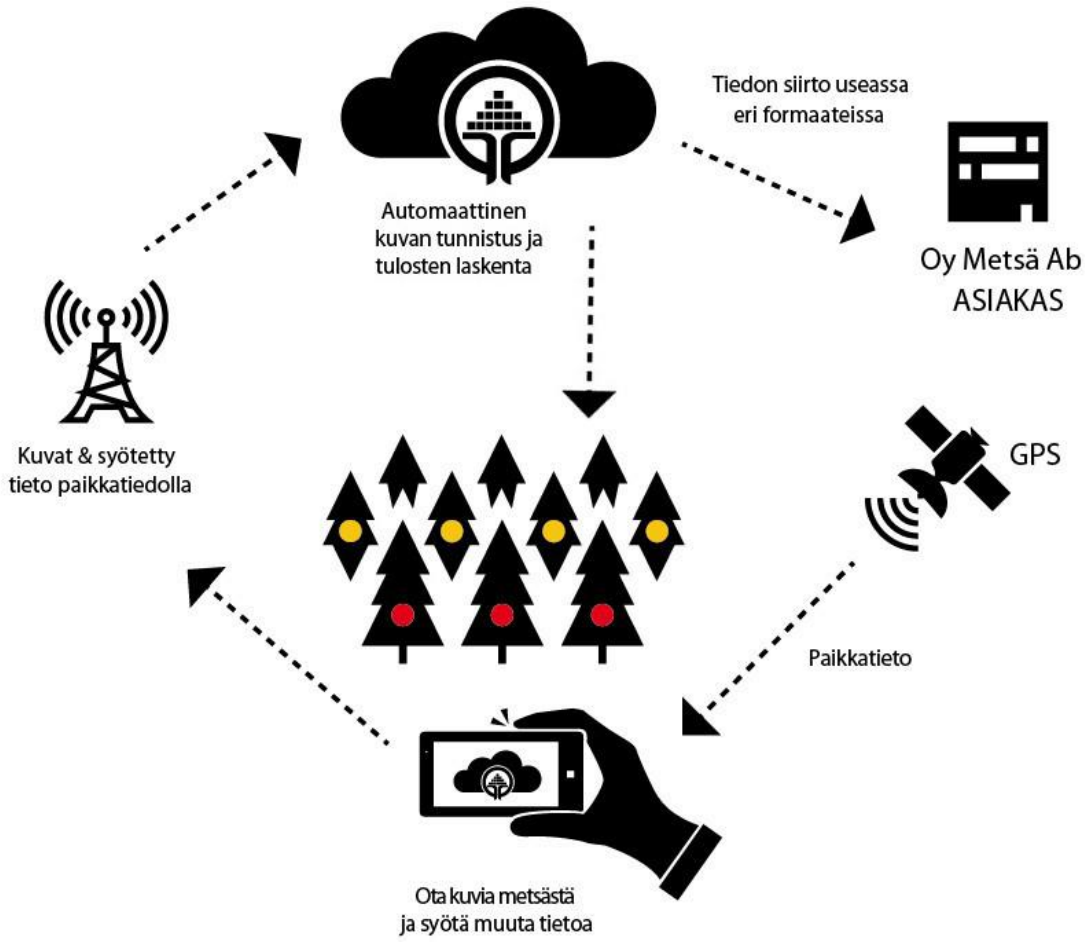


KUVA 1. TRESTIMAn ja relaskoopin mittausalue. Lähde: Trestima OY

2.3 TRESTIMAn käyttö

Yksinkertaisimmillaan TRESTIMA metsänmittausjärjestelmän käyttö on helppoa. Ensin käyttäjän tarvitsee luoda käyttäjätunnus TRESTIMAn verkkosivuilla ja ladata sinne olemassa oleva metsätieto esimerkiksi metsään.fi –palvelusta. Seuraavaksi ladataan ja asennetaan TRESTIMA-sovellus mobiililaitteeseen ja siihen kirjaudutaan edellä mainitulla tunnuksella. Laite hakee ja päivittää laitteen sijainnin GPS-tiedon perusteella, ja otetut valokuvat yhdistetään sijainti- ja asentotietoon sekä lähetetään pilvipalveluun. Pilvestä saadaan mittaustulokset ja ne päivittyvät mobiililaitteeseen, jolla voidaan katsoa senhetkisten mittaustietojen raporttia. Kokonaisraportin saa

mittauksen päätteeksi ladattua TRESTIMAn verkkopalvelusta. Kuvausprosessin kaavio on esitetty kuvassa 2.

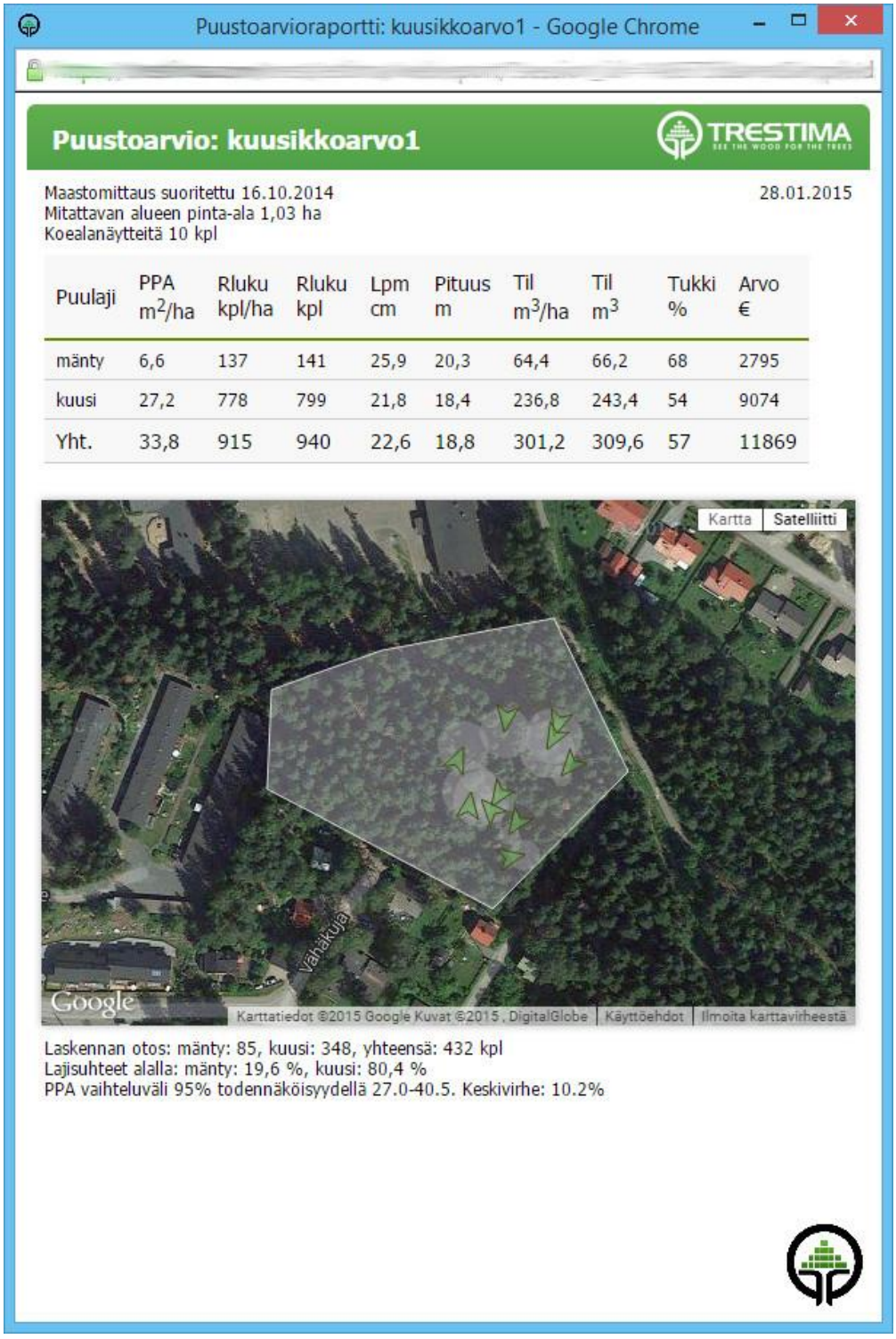


KUVA 2. Kuvausprosessi kaaviona. Lähde: Trestima OY

TRESTIMAlla luotu metsätietoraportti sisältää jokaisesta konenäön tarkastamasta kuvasta saatujen tietojen avulla lasketut numeeriset arvot metsälle. Raportissa on mm. puiden pohjapinta-ala, joka kertoo yksittäisten puiden rinnankorkeudelta mitattujen poikkileikkausalojen summan neliömetreinä hehtaaria kohden. Puiden korkeus arvioidaan rungon läpimittojen perusteella puiden maantieteellisen sijainnin mukaan tai vaihtoehtoisesti hahmontunnistuksella TRESTIMA-mittakepin avulla.

Raportin runkolukuarvio perustuu hahmontunnistuksen keskimääräiseen runkomäärään ja se ilmoitetaan runkoina per hehtaari. Puiden halkaisijat puolestaan arvioidaan konenäön avulla kuvista puiden leveytenä ottaen huomioon optiset vääristymät ja

puiden ympyrämäisen muodon suhteessa kuvakulmaan. Raportti koostaa myös maantieteellisen sijainnin perusteella arvion tukkitilavuudesta sekä metsän arvosta säännöllisesti päivittyvin hinnoin. Raportti kokonaisuudessaan on esitetty kuvassa 3.



Puulaji	PPA m^2/ha	Rluku kpl/ha	Rluku kpl	Lpm cm	Pituus m	Til m^3/ha	Til m^3	Tukki %	Arvo €
mänty	6,6	137	141	25,9	20,3	64,4	66,2	68	2795
kuusi	27,2	778	799	21,8	18,4	236,8	243,4	54	9074
Yht.	33,8	915	940	22,6	18,8	301,2	309,6	57	11869

KUVA 3. Valmis raportti mitatuista tuloksista. Lähde: Trestima OY

3 MODULAARINEN OHJELMOINTI

Modulaarinen ohjelmointi tarkoittaa ohjelmointitapaa, jossa ohjelman erilaiset toiminnot jaetaan omiin moduuleihin, joista jokainen suorittaa tiettyä funktiota ja sisältää vain funktion tarvitsemat muuttujat ja ohjelmakoodin. Modulaarinen ohjelmointitapa, eli ohjelmiston pilkkominen pieniin osiin, on selkeä tapa ylläpitää ja kehittää suurta ohjelmistoa, joka olisi ilman modularisointia valtava ja epäselvä kokonaisuus. Virheiden etsintä helpottuu ohjelman ollessa pienemmissä paloissa verrattuna yhteen kokonaisuuteen. (Nguyen 2004.)

3.1 Modulaarisuuden hyödyt

Modulaarisuuden hyödyllisyys näkyy käytännössä ohjelmiston eri kokonaisuuksien erotteluna ja selkeänä järjestelmänä erilaisten funktioiden jaotteluun. Jokaista moduulia voidaan testata ja kehittää erikseen. Jos ohjelmaa kehitetään käytettäväksi kirjastona, se on jo valmiiksi modulaariajattelulla toteutettuna. Moduuleita voidaan kehittää yksittäin tai useita kerrallaan ja niillä on yleensä versionumerot. Versioimalla varmistetaan erilaisten kokoonpanojen toimivuus dokumentoimalla käytössä olleet versiot. Moduuliriippuvuudet versionumeroilla ilmaistuna kertovat pienimmän toimivan versionumeron ja useimmiten uudemmat versiot toimivat samalla tavalla, mutta sisältävät uusia funktioita tai optimointeja. (Blewitt 2009.)

Modulaarinen ohjelmistorakenne mahdollistaa samanaikaisen kehitystyön eri tiimeissä. Tämä on etu suurissa yrityksissä, joissa on useita eri kokonaisuuksia, sillä kokonaisuuksia pilkkomalla saadaan ajallisesti paljon tehokkaampaa työtä, kuin yhden kokonaisuuden kehitys vuorotellen kehitystiimien kesken. (Blewitt 2009.)

3.2 Modulaarisuuden haitat

Modulaarista ohjelmistoa tehdessä dokumentaation merkitys korostuu monoliittiseen ohjelmointiin verrattuna, sillä varsinkin suljetussa moduulissa metodien toimintaa ei voida tutkia koodin perusteella. Metodien dokumentaation tulee olla kattavaa, jotta moduulia voidaan käyttää tutkimatta lähdekoodia. Samalla rajapinnan metodien nimeämiseen tulee kiinnittää enemmän huomiota, sillä rajapinta saattaa olla moduulin ainoa näkyvä osa. Rajapinnan metodien nimeämisen lisäksi metodien toiminta tulee testata erittäin hyvin, jotta virhetilanteet pystytään estämään tai hallitsemaan. Moduulin sisäisten kutsujen on myös oltava huolellisesti testattuja, jotta kokonaisuuden toiminta voidaan taata. (Garza 2011.)

Pienissä tai yksinkertaisissa ohjelmissa modularisointi aiheuttaa ylimääräistä työtä, mikäli ohjelmistoa ei ole tarkoitus kehittää tai käyttää uudelleen muissa ohjelmistoissa. Modulaarinen ohjelmointi lisää myös tiedostokokoa marginaalisesti, sillä metodit tarvitsevat erillisen rajapinnan ohjausta varten. Joissain ohjelmointikielissä moduulien käyttämät muuttujat saattavat sekoittua pääohjelman globaaleiden muuttujien takia. Ohjelman suoritusaika saattaa pidentyä, kun metodeita pitää kutsua erillisen rajapinnan kautta. (Dhillion 1987, 189.)

3.3 Javan modulaarisuus

Java-ohjelmointikieltä ei alun perin suunniteltu modulaarisuutta ajatellen, mutta käytännössä Java-ohjelmointi perustuu hyvin pitkälti moduuleihin, sillä Javassa erilaiset luokat ovat omia tiedostojaan ja luokkatiedostot ovat tietyn paketin sisällä, jolloin yksittäinen paketti muodostaa moduulin. Olio-ohjelmoinnissa moduuleilla on usein standardisoitu viittaustapa, mutta Javassa ei sellaista varsinaisesti ole, vaan Javan moduuliviittaukset ovat suoraan pakettien luokkatiedostoihin viittauksia sekä kirjastojen lisäämistä. (Blewitt 2009.)

Java-ohjelmoinnissa käytettävät kirjastot ovat käytännössä valmiita moduuleita. Kirjastojen ja pakettien julkisia funktioita voidaan käyttää pääohjelmassa ja yksityiset funktiot jäävät vain luokkien sisäiseen käyttöön. Toisaalta myös paketit ja kirjastot voivat olla riippuvaisia toisistaan, joka myöskin tukee moduuliajattelua. Joissain

tapauksissa kirjastojen väliset riippuvuudet voivat olla vapaaehtoisia, eli vaikka jokin pyydetty kirjasto puuttuisi, voidaan silti käyttää osaa toiminnoista. Virhetilanne syntyy vasta, kun yritetään viitata puuttuvan kirjaston funktioon. (Blewitt 2009.)

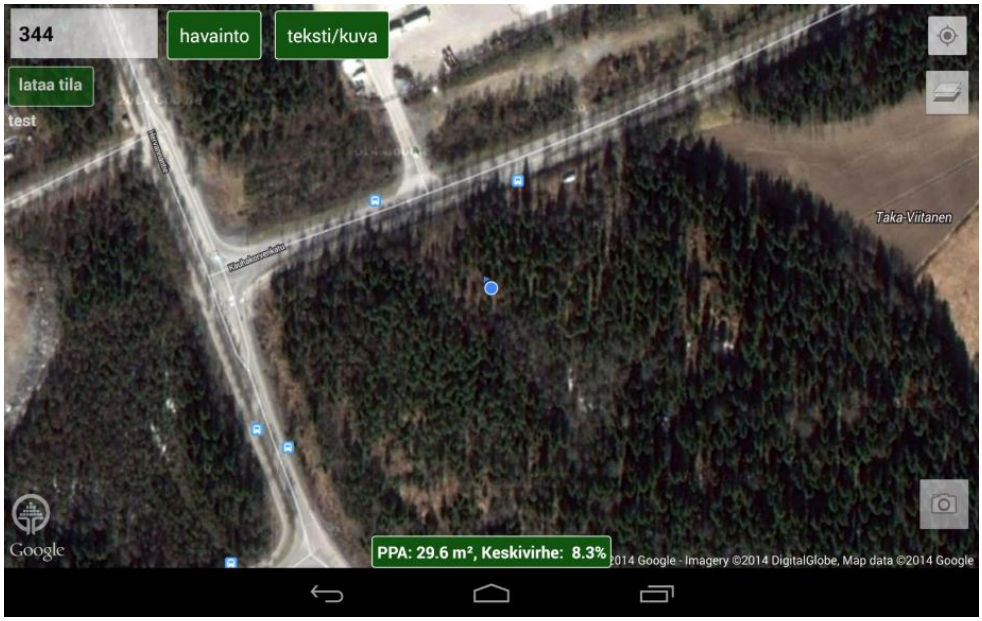
Luokkien ja pakettien näkyvyys moduuleissa määritellään funktioiden ja luokkien käyttöluokituksella. Pakettien käyttöluokitusten näkyvyydet on listattu taulukossa 1. Javassa käyttöluokituksia on neljä: julkinen, suojattu, pakettikohtainen sekä yksityinen. Julkiset funktiot ja luokat ovat kaikkialle näkyviä ja niitä voidaan käyttää moduulin ulkopuolelta. Pakettikohtaiset funktiot näkyvät kaikille paketin alaluokille, mutta toisesta paketista niitä ei voida käyttää. Käyttöluokitukseltaan suojatut funktiot ovat käytännössä samanlaisia kuin pakettikohtaiset, mutta lisäksi suojattujen luokkien alaluokkia voidaan käyttää myös toisesta paketista. Yksityisiä funktioita voidaan käyttää ainoastaan luokan sisällä. (Ganesh 2014.)

Taulukko 1. Käyttöluokitusten näkyvyys Javassa

Modifier	Class	Package	Subclass	World
public	visible	visible	visible	visible
package-private	visible	visible	visible	not visible
protected	visible	visible	not visible	not visible
private	visible	not visible	not visible	not visible

4 TRESTIMA-SOVELLUKSEN MODULARISOINTI

TRESTIMA metsänmittausjärjestelmän Android-sovellus koostuu useista luokkatiedostoista, jotka käsittelevät ohjelman sisäistä dataa ohjelman toiminnan edellyttämillä tavoilla. Toiminnallisuudet kuvan ottamisesta tiedoston lähetykseen ja raportin käsittelyyn tehdään luokkiin jaetussa ohjelmistossa. Sovellus tarvitsee käyttöjärjestelmältä käyttöoikeudet mm. kameraan, tallennusmediaan sekä internet-yhteyden tilan tarkkailuun. Kuvassa 4 on kuvankaappaus alkuperäisestä TRESTIMA metsänmittaussovelluksesta.



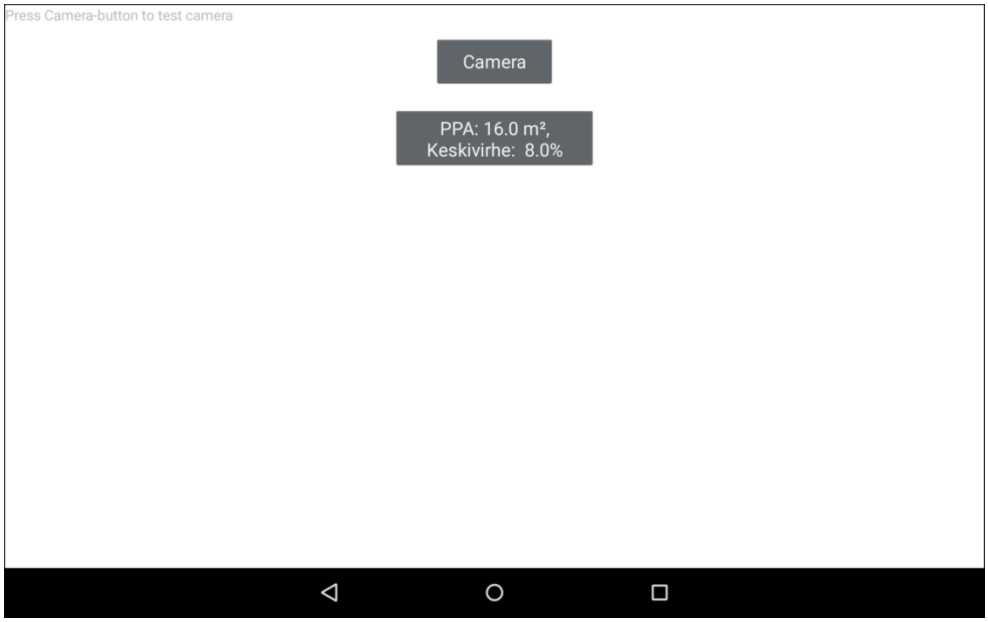
KUVA 4. Kuvankaappaus TRESTIMA metsänmittausjärjestelmästä

Tarve modulaariselle ohjelmistolle tuli yrityksen sisältä ja ohjelmiston modularisointi aloitettiin suunnittelemalla ohjelmiston rajapinta. Rajapinnan suunnittelussa ehtoina olivat kehittäjän näkökulmasta helppokäyttöisyys ja yksinkertaisuus, jotta moduulia olisi mahdollista käyttää missä tahansa Android-sovelluksessa. Ohjelmiston olisi oltava myös tarpeeksi kattava ollakseen hyödyllinen, mutta alkuperäisen sovelluksen kaikkia ominaisuuksia ei kuitenkaan otettu mukaan, sillä kaikille ominaisuuksille ei ollut tarvetta ja loppukäyttäjä voi tehdä itse haluamansa lisäykset, kuten kartan, omalla tavallaan ohjelmistoonsa.

4.1 Rajapinnat

Modulaarinen versio koostuu koko toiminnallisuutta ohjaavasta taustapalvelurajapinnasta ja kameraa ohjaavasta rajapinnasta. Taustapalvelu käynnistetään pääohjelman käynnistyessä ja sille syötetään kirjautumistiedot käynnistyskäskyn yhteydessä. Palvelu on toteutettu ”sticky”-palveluksi, jolloin se pysyy taustalla käynnissä niin kauan, kunnes se erillisellä käskyllä lopetetaan (Google 2015). Kamerarajapinta puolestaan ohjaa TRESTIMA metsänmittausjärjestelmässä käytettyä kamerasovellusta, jossa näytetään horisontin suoruutta kuvaava tähtäin, kuvatyyppi-ilmaisin sekä kosketuspainike kuvan ottamista varten. Rajapintaa ohjataan yksinkertaisella Android-aktiviteetilla. Kuvan ottamisen ja tallentamisen jälkeen kamera-aktiviteetti sulkeutuu ja palaa pääohjelmaan, josta rajapintaa kutsuttiin.

Modulaarinen versio käyttää monia samoja luokkatiedostoja kuin alkuperäinen metsänmittausohjelmisto, mutta ainoat loppukäyttäjälle näkyvät luokat ovat TrestimaService ja TrestimaCamera-rajapinnat ja niiden sisällä on rajoitettu määrä käyttäjälle näkyviä metodeita. Modulaarinen ohjelmisto toimitetaan sinetöitynä jar (Java Archive) –tiedostona, joka lisätään haluttuun ohjelmistoon, jolloin tiedostossa olevat kirjastot (rajapinnat) ovat käytettävissä. Kuvassa 5 on kuvankaappaus rajapintoja käyttävästä yksinkertaisesta testisovelluksesta, jossa käytetään TrestimaCamera-viittausta ja TrestimaServiceListener-viittausta raportin (kuva 6) luomiseksi. TrestimaService-rajapinnan käyttäjälle näkyvät metodit ovat listattu taulukoihin 2 ja 3.



KUVA 5. Kuvankaappaus testisovelluksesta, jossa käytetään rajapintoja

Taulukko 2. TrestimaServiceListener –kuuntelijan metodit

Metodi	Kuvaus
`public interface TrestimaServiceListener{}`	Tämä on TrestimaServicen-palvelun kuuntelijafunktio, joka palauttaa kahdelle alla mainitulle funktiolle arvoja aina, kun kyseisten funktioiden muuttujissa tapahtuu muutos.
`void reportChanged(String onelinereport, String fullHtmlReport)`	TrestimaServiceListenerin ensimmäinen palautusfunktio, joka palauttaa metsätietoraportin loppukäyttäjälle kahdessa muodossa, yhden rivin raporttina ja täysikokoisena HTML-raporttina.
`void queueChanged(int queueCount, boolean isSending, boolean networkError)`	TrestimaServiceListenerin toinen palautusfunktio, joka ilmaisee loppukäyttäjälle kuvajonon tilan. Tilassa ilmoitetaan jonossa olevien kuvien lukumäärä, onko kuvan lähetys käynnissä ja onko lähetyksessä tapahtunut yhteysvirheitä.

KUVA 6. TrestimaServiceListenerin kautta saatu metsätietoraportti

Taulukko 3. TrestimaService –palvelun julkiset metodit

Metodi	Kuvaus
`public static void setSessionId(final String sessionId)`	Asettaa kuvaussessiolle ID-numeron string-muodossa. ID toimitetaan yleensä metsätilatiedoissa numerona.
`public static void setSessionName(final String sessionName)`	Asettaa kuvaussessiolle string-tyyppisen nimen, jos ID:tä ei ole saatavilla.
`public static void addListener(TrestimaServiceListener listener)`	Lisää TrestimaServiceListener-kuuntelijan, joka palauttaa loppukäyttäjälle tietoja palvelun tilasta.
`public static void removeListener()`	Poistaa TrestimaServiceListener-kuuntelijan käytöstä, jolloin loppukäyttäjä ei enää saa päivityksiä palvelun tilasta.

4.2 Palvelun luominen

TrestimaService käynnistetään seuraavasti:

```
Bundle b = new Bundle();
b.putString("username", "yourUsrName");
b.putString("password", "yourPwd");
Intent serviceIntent = new Intent(getBaseContext(), Tresti-
maService.class);
serviceIntent.putExtras(b);
startService(serviceIntent);
```

Kyseinen koodi on TrestimaService-taustapalvelun käynnistyskutsu, joka suoritetaan pääohjelmaa käynnistäessä. Palvelulle annetaan TRESTIMA metsänmittausjärjestelmään rekisteröity käyttäjänimi ja salasana, jotka lisätään palvelun käynnistyskutsuun. Mikäli ohjelmalla ei ole kirjautumisen aikana yhteyttä TRESTIMAn palvelimiin, ohjelma koittaa kirjautua vertaamalla uutta kirjautumisyritystä edellisellä onnistuneella kirjautumiskerralla tallennettuihin tietoihin. Jos kirjautumistiedot täsmäävät tallennettuihin tietoihin, käyttäjä pääsee käyttämään ohjelmaa. Mikäli

kirjautumisyritys ei täsmää edelliskertaisiin tietoihin ja yhteyttä ei saada palvelimiin, kirjautumisyritys lopetetaan eikä palvelua käynnistetä. Tällöin loppukäyttäjä ei saa otettua kuvia eikä haettua raportteja TRESTIMAn pilvipalvelusta.

TrestimaService lopetetaan seuraavasti:

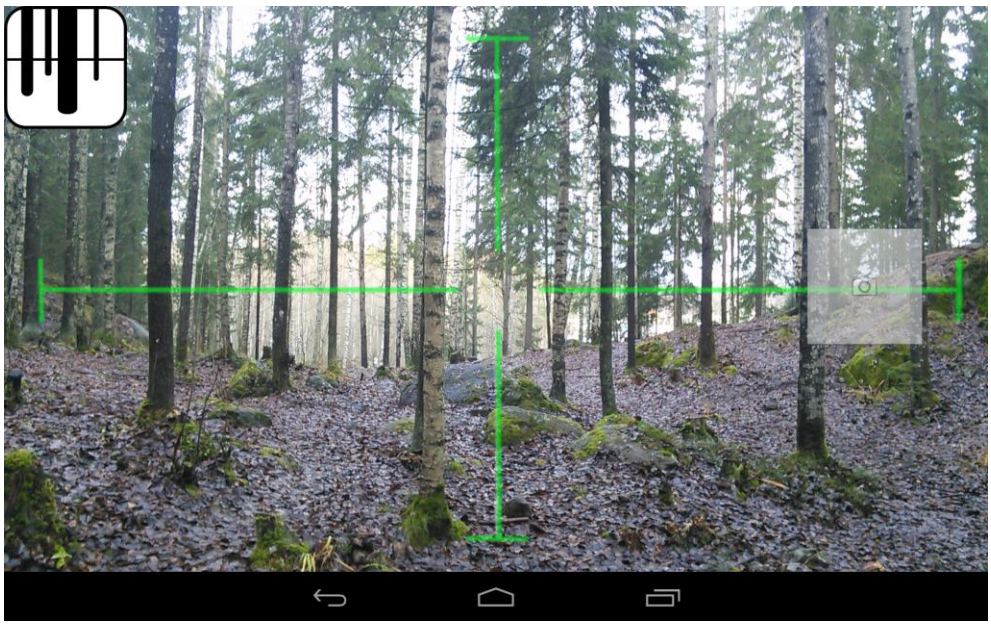
```
stopService(new Intent(getBaseContext(), TrestimaService.class));
```

Lopetuskäsky ajetaan pääohjelmasta, kun pääohjelma suljetaan tai kun palvelu halutaan lopettaa käsin. Mikäli palvelua ei lopeteta, se jää taustalle käyntiin ja hakee metsätietoja aktiivisesti pääohjelman sulkemista edeltävien tietojen perusteella ja saattaa kuluttaa akkua tarpeettoman nopeasti.

TrestimaCamera-rajapinta sisältää vain yhden julkisen metodin, kamera-aktiviteetin käynnistyksen. Kamera-rajapinta käynnistetään seuraavalla komennolla pääohjelmasta tai muusta halutusta sovelluksen tilasta:

```
Intent intent =new Intent(MainActivity.this, Trestima-
Camera.class);startActivity(intent);
```

TrestimaCamera käynnistyy uuteen aktiviteettiin, joka täyttää ruudun kameranäkymällä. Kameranäkymä koostuu kameran kennolle piirtyvästä reaaliaikaisesta näkymästä, tähtäimestä, kuvatyyppi-indikaattorista sekä laukaisinnapista (kuva 7). Kuvan tarkennettua kuuluu sitä indikoiva ääni ja kuvan ottamista ilmaisee toinen ääni. Kuvaa tallentaessa bittikartta muutetaan jpg-formaattiin ja kuva pakataan, jotta pilvipalveluun lähettäminen ei kestäisi kauaa ja samalla säästetään pilvipalvelimien levytilaa sekä käyttäjän oman laitteen tallennustilaa ja akkua. Kuva lisätään kuvajonoon, jonka tila ilmaistaan TrestimaServiceListenerin kautta loppukäyttäjälle. Kuvan ottamisen jälkeen aktiviteetti sulkeutuu ja palaa sitä kutsuneeseen aktiviteettiin.



KUVA 7. TrestimaCamera-rajapinnan käynnistämä kameranäkymä

4.3 Sovelluksen käyttöoikeudet

Modulaarinen ohjelmisto voidaan lisätä mihin tahansa Android-sovellukseen, sillä se on suunniteltu toimivaksi itsenäisenä kokonaisuutena, eikä käyttäjän tarvitse kommunikoida moduulin kanssa aktiivisesti. Ohjelma on käyttövalmis, kun sille annetaan käynnistyskäsky ja sen mukana kirjautumistiedot, sekä nimi sessiolle. Modulaarinen sovellus käyttää omia metodeja hakeakseen tarkan sijainnin, sensoritiedot, kameradatan ja verkon tilan. Modulaarinen versio vaatii sovellukselta erilaisia käyttöoikeuksia. Tarvittavat käyttöoikeudet ja toiminnot ovat esitetty taulukossa 4.

Taulukko 4. Modulaarisen sovelluksen vaatimat käyttöoikeudet ja toiminnallisuudet

Käyttöoikeus	Kuvaus
android.permission.INTERNET	Käyttöoikeus verkkoliikennettä varten, tämän avulla kuvat ja raportit liikkuvat laitteen ja pilvipalvelun välillä.
android.permission.NETWORK_STATE	Käyttöoikeus verkon tilan tarkastelemista varten sekä sensoritietoja varten.
android.permission.WRITE_EXTERNAL_STORAGE	Käyttöoikeus laitteen muistiin kirjoittamiselle, sovelluksella otetut kuvat tallennetaan paikallisesti laitteeseen, ennen kuin ne lähetetään pilvipalveluun.
android.permission.ACCESS_COARSE_LOCATION	Käyttöoikeus karkean sijainnin hakemiselle GPS:n avulla.
android.permission.ACCESS_FINE_LOCATION	Käyttöoikeus tarkan sijainnin hakemiselle, GPS:n apuna on myös tunnettujen WIFI-yhteyksien ja mobiiliverkkojen sijaintitiedot.
android.permission.ACCESS_NETWORK_STATE	Käyttöoikeus verkon tilatietojen sekä sensoritietojen käyttämiselle.
android.permission.CAMERA	Käyttöoikeus laitteen kameralle, mittausprosessi perustuu kokonaan kameran käyttöön ja hahmontunnistukseen.
android.hardware.camera	Käyttöoikeus kameran esikatselunäkymän näyttämiselle.
android.hardware.camera.autofocus	Käyttöoikeus laitteen kameran automattitarkennuksen käyttämiselle.

4.4 Dynaamisuus

Alkuperäisen TRESTIMA-sovelluksen muuntaminen modulaariseksi vaati suuria muutoksia luokkaviittauksiin ja esimerkiksi kamerarajapinnan näkymät piti muuttaa kaikki dynaamisiksi, jotta niitä voidaan käyttää ulkoisen kirjaston kautta. Android-kehitysalusta käytetty Eclipse luo koodin käännösvaiheessa R.java-tiedoston, joka muuntaa kaikki viittaukset ulkoisiin resurssitiedostoihin, kuten kuviin ja äänitiedostoihin id-numeroiksi. Jokaisella resurssilla on siis käännösvaiheen jälkeen kiinteä id, johon viitataan aina kyseisiä tietoja käyttäessä. R-tiedosto ja samalla jokainen asetettu id muuttuu, kun resurssitiedostoja lisätään tai vähennetään ja jokaisella ohjelmalla on oma R.java-tiedosto. (Google 2015.)

Androidin tavallista viittaustapaa ei siis voida käyttää modulaarisessa ohjelmistossa, sillä se edellyttäisi R.java-tiedoston liittämistä uuteen kehitysympäristöön ja jokaisen resurssitiedoston pitäisi olla käytännössä sama ilman minkäänlaista eroavaisuutta uudessa ympäristössä. Tämä aiheuttaa konfliktin, sillä modulaarinen ohjelmisto olisi tällä tavalla toteutettuna kiinteä sovellus, jota ei pystyisi käyttämään kirjastona mille tahansa sovelluspohjalle. Androidille tyypillinen viittaustapa resurssille on:

```
R.drawable.imagetype_height
```

Komennosta huomataan, että siinä viitataan *R*-tiedostoon, jotta saadaan id-numero *drawable*-tyyppiselle tiedostolle *imagetype_height.* Dynaamisessa versiossa R-viittaus jätetään kokonaan pois ja resurssitiedostoon viitataan seuraavalla tavalla:

```
context.getResources().getIdentifier("imagetype_height" , "drawable", context.getPackageName())
```

Viittaustapa on huomattavasti pidempi, sillä *R*-tiedostoon viittausta ei voida käyttää. Uusi keino *imagetype_height*-tiedoston kutsumiselle on käyttää kohdeympäristön *context*-tietoa, joka antaa tiedon ajonaikaisista muuttujista ja pyytää siltä resurssia komennolla *getResources()* ja pyytää resurssille id:tä komennolla *getIdentifier(...)*. *GetIdentifier(...)*-viittaus saa argumenteiksi tiedoston nimen, tiedoston tyypin sekä sen paketin sijainnin, josta tiedostoa pitää hakea (Google 2015). Täydelliseen dynaamisuuteen käskyllä ei päästä, sillä se edellyttää vieläkin, että kehitysympäristö, joka ottaa modulaarisen paketin käyttöönsä, sijoittaa *imagetype_height*-tiedoston oman

kehitysympäristönsä resurssikansioon sille kuuluvaan alakansioon, tämän tiedoston tapauksessa:

```
CurrentProject/res/drawable/imagetype_height.png
```

Dynaaminen viittaustapa on siis vapaa kiinteistä id-viittauksista, mutta vaatii kehittäjältä sen, että jokainen ohjelman käyttämä resurssitiedosto sijaitsee vaadituissa kansioissa. Modulaarisen paketin mukana toimitetaan kaikki tarvittavat tiedostot ja toisaalta tiedostot saavat olla loppukäyttäjän omia grafiikoita sillä ehdolla, että niillä on sama tiedostonimi ja sijainti. Kuvat skaalataan sovelluksessa samankokoisiksi, kuin alkuperäisessä sovelluksessa. Tämä mahdollistaa loppukäyttäjän graafisen käyttöliittymän mukauttamisen muuttamatta modulaarisen ohjelman toimintaa millään tavalla.

R.java sisältää myös ulkoasuun käytettyjen xml-tiedostojen sisältämät graafisten kenttien id:t (Google 2015). Modulaarisen ohjelman kannalta tämä tarkoittaa, että jokainen ulkoasuun liittyvä komponentti on luotava dynaamisesti ilman xml-tiedostoviittauksia. Tavallisesti ulkoasuun liittyvät viittaukset avautuvat aktiviteetissa fokukseen asetetun xml-tiedoston perusteella automaattisesti näytölle, mutta dynaamisia komponentteja käyttäessä ne pitää ensin luoda ja sitten antaa parametrit ja sijainti käsin. Tyypillinen xml-tiedostoon viittaava ulkoasua muokkaava komento on seuraavanlainen:

```
mCameraView = (CameraView) findViewById(R.id.cameraView);
```

Kyseisessä käskyssä CameraView-tyyppinen näkymämuuttuja mCameraView saa sisällökseen viittauksen xml-tiedostoon määritellyn CameraView-näkymän parametrit. Dynaamista näkymää luotaessa alustukseen on lisättävä myös xml-tiedostossa esitellyt parametrit ja sijainti:

```
mCameraView = new CameraView(this, null);
RelativeLayout.LayoutParams lp = new RelativeLay-
out.LayoutParams(LayoutParams.FILL_PARENT, Layout-
Params.FILL_PARENT);
mCameraView.setLayoutParams(lp);
this.setContentView(mCameraView);
```

Uudelle näkymälle annetaan parametreina hallitsevan näkymän täyttäminen, ja alustettu näkymä asetetaan lopuksi päänäkymäksi, jonka päälle uudet dynaamiset näkymät luodaan.

Varsinaisessa kameranäkymässä näytölle piirretään fyysisen kameran kennolta luettava bittikartta, joka päivittyy laitteen prosessointitehosta ja kameramoduulista riippuen noin 15-30 kertaa sekunnissa. Kuvatyypin tunnistukseen käytetään laitteessa olevia kiihtyvyysanturia ja magnetometriä, jotka päivittyvät jatkuvasti kameranäkymän ollessa käynnissä. Kiihtyvyys- ja magnetometridatan perusteella lasketaan jatkuvasti kameran asentoa, jonka perusteella edelleen päätetään kuvatyyppi. Sensoridatan liukulukuarvoista kootaan kolmiakselisen XYZ-koordinaatiston mukaiset kokonaisluvut, jotka kertovat kameran suunnan kohtisuorassa kennoa vasten suoraan asteina. Kameran asennon avulla määritellään kuvatyypin lisäksi myös maaston kaltevuus, josta lasketaan pohjapinta-alalle korjauskerroin.

4.5 Käyttöönotto

Sovelluksen käyttöönotto tapahtuu Eclipse-ohjelmointiympäristössä joko luomalla uusi projekti tai lisäämällä TRESTIMA-moduuli jo olemassa olevaan projektiin. Moduuli toimitetaan paketissa, joka sisältää TrestimaModule-v0.90.jar-tiedoston, käyttöohjeet ja muut tarvittavat tiedostot, kuten resurssitiedostot. Käyttöohjeissa kerrotaan, että ensimmäiseksi projektiin on lisättävä kaikki paketin mukana tulleet .jar-tiedostot. Seuraavaksi on varmistettava, että projektiin lisätyt kirjastot ovat käännösvaiheessa käännettävä ennen varsinaista projektia.

AndroidManifest-määrittelytiedostoon on lisättävä kirjaston käyttöoikeudet laitteen toiminnallisuuksille sekä lisättävä ohjelman aktiviteetille esittely unohtamatta palvelun esittelyä. Aktiviteetille on lisätty määrittelyyn, että se aukeaa vain vaakatasossa, sillä alkuperäisen TRESTIMA-sovelluksen kamera on suunniteltu vaakatasossa käytettäväksi. Aktiviteetille asetetaan myös tyylimäärittely, joka asettaa kameranäkymän koko ruudun kokoiseksi ja poistaa valikot ja Androidin ilmoituspalkin:

```
<application>
        …
        <activity
        android:name=”com.trestima.TrestimaCamera”
        android:screenOrientation=”landscape”
        android:theme=”@android:style/Theme.Holo.NoActionBar.Fullscreen”>
        </activity>
        <service android:name=”com.trestima.TrestimaService” />
</application>
```

Ohjeissa neuvotaan, että pääaktiviteetille lisätään komento, joka estää aktiviteetin uudelleenlatauksen aina, kun näppäimistö poistuu näkyvistä, ruutua käännetään tai jos näytön kokoa muutetaan. Tätä lisäystä ehdotetaan siksi, että pääaktiviteetin mukana käynnistettävä TrestimaService ei käynnistyisi uudelleen jokaisen edellä mainitun tapahtuman seurauksena.

Resurssitiedostojen kansiorakenteet ja tiedostonimet neuvotaan yksityiskohtaisesti ja mainitaan, että tiedostot voidaan korvata käyttäjän omilla tiedostoilla, kunhan nimet ja sijainnit ovat samat.

Lopuksi ohjeissa neuvotaan lisäämään TrestimaService, TrestimaCamera ja TrestimaServiceListener siihen aktiviteettiin, josta TRESTIMA-moduulia halutaan käyttää. Moduulipaketissa on annettu myös täydelliset esimerkit MainActivity.java ja activity_main.xml –tiedostojen mukana. Mukana toimitettavissa tiedostoissa esitellään kaikkien julkisten metodien toiminnot lyhyesti, mutta käytännöllisesti. (Trestima 2015.)

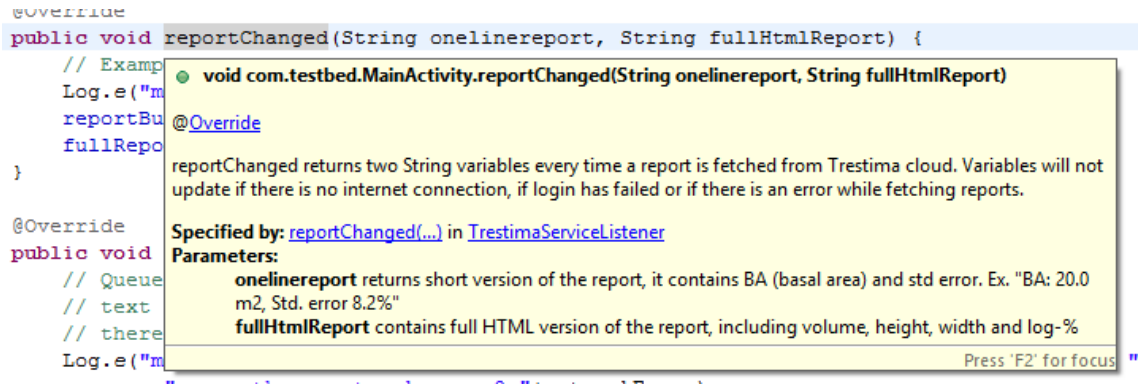
4.6 Testaus

Modulaarisen ohjelmiston testausta tehtiin lähes jatkuvasti ohjelman kirjoittamisen ohella. Aina kun uusi metodi luotiin, sitä testattiin ja koitettiin etsiä ulkoasun virheet, mahdolliset väärinkäytön tavat ja suorat virhetilanteet. Ohjelmiston kehittyessä uudet metodit saivat jo luomisvaiheessa aiempien testausten kokemuksella kattavan virhetilanteiden määrittelyn ja palautumistoiminnot. Valmiiseen modulaariseen sovellukseen rajapintojen julkisia metodeita tuli varsin vähän, joten niiden testaaminen oli tiimin osalta hyvin helppoa. Virhetilanteita voidaan välttää myös kirjoittamalla kattava dokumentointi metodien käytöstä.

4.7 Dokumentaatio

Moduulin metodeita tutkittaessa jokaiselle metodille on kirjoitettu oma Javadoc-tyyppinen, kattava ohjeistus ja käyttöohje lyhyin esimerkein, joita on myös tässä työssä esitelty. Ohjeistuksessa esitellään metodin päätoiminta, jokaisen syötettävän muuttujan tyyppi ja merkitys, sekä palautusarvon tyyppi ja sisältö. Javadoc-dokumentaatio tarkoittaa metodien ja luokkien lähdekoodiin integroitua dokumentaatiota. Sen avulla voidaan kirjoittaa kattavia käyttöohjeita funktioille ja esitellä luokan toiminta kokonaisuudessaan sekä viitata käytössä oleviin metodeihin.

Javadoc-dokumentaatio kirjoitetaan kerrottavan funktion yläpuolelle ja siihen voidaan lisätä erilaisia viittauksia, kuten tekijän nimi, syötettävät parametrit, palautustyyppi ja virhetilanteet. Javadoc sopii paremmin suljettujen funktioiden dokumentointiin kuin esiteltävän lähdekoodin kommentointiin, sillä Jacadoc ei ole lähdekoodissa esillä kommenttien muodossa pidentämässä koodin rivimäärää. Dokumentaatio integroituu käännösvaiheessa funktiolle, jolloin se on luettavissa moduulin Javadoc-tiedostosta tai viemällä kursori funktion päälle (kuva 8). (Heckman 2004.)



KUVA 8. Javadoc-dokumentaatio näkyy, kun kursori viedään metodin päälle.

5 POHDINTA

Idea opinnäytetyölle tuli Trestima OY:stä. Puhtaasti modulaarinen ohjelmisto olisi helpompi ylläpitää ja kehittää sekä tehdä sille automaattista testausta julkisten metodien kautta. Modulaarista ohjelmistoa olisi mahdollista käyttää myös muuhun käyttöön ja sen voisi liittää mihin tahansa olemassa olevaan ohjelmistoon hyvin yksinkertaisesti.

Modulaarinen versio valmistui testikäyttöön 18.12.2014, jolloin ohjelmaa oli kehitetty noin kolme täyttä työviikkoa. Merkittävin hyöty on ohjelman uudelleenkäyttöä ajatellen sen rajapinnassa ja sen julkisiksi jaetuissa metodeissa. Samalla ne metodit, joita loppukäyttäjää ei haluta päästää tarkastelemaan, jäävät piiloon, ja ohjelmakokonaisuus toimii, jos moduulin käyttöohjeita noudatetaan.

Sovelluksen tekemiselle annettiin vapaat kädet ja vastuu rajapinnan suunnittelulle ja toteuttamiselle. Ohjeita ja vinkkejä annettiin kokonaisuutta ajatellen, mutta yksityiskohdat jätettiin tekijälle. Ohjelmointi osoittautui välillä hankalaksi, ja esimerkiksi dynaamisia resurssiviittauksia varten tarvittavia funktioita piti etsiä internetistä ja useiden esimerkkien kautta ohjelmaan löytyi sopiva tapa. Karkeasti voitaisiin pyöristää, että noin puolet työajasta meni uusien tietojen etsimiseen ja opetteluun. Jäljelle jäävästä työajasta noin puolet meni ohjelmointiin ja testaukseen, ja loput työajasta joko uuden metodin suunnitteluun tai virheenetsintään.

Ohjelman kehitystä ohjasi säännölliset kehityskeskustelut ja kehittävää palautetta annettiin metodien toiminnasta ja mahdollisista puutteista. Versionhallinnassa käytössä on Bitbucket.org:n GIT-järjestelmä. Järjestelmään voidaan lisätä pienet ja suuret muokkaukset jokaisesta tiedostosta ja muokkaukset näytetään rinnakkain edellisen version kanssa. GIT-järjestelmä sopii myös palautteenantoon, sillä jokaista muokkausta on mahdollista kommentoida. Ohjelman virheet ja metodien muokkaukset on helppo tehdä, kun kommentit voidaan osoittaa rivinumeron tarkkuudella jokaiseen tiedostoon.

Ohjelmoinnin ja uusien asioiden opettelun rinnalla sovellusta testattiin jatkuvasti. Moduulia käytetään kahden rajapinnan avulla, joista toinen ohjaa kaikkia TRESTIMA-sovelluksessa käytettyjä taustaprosesseja ja tiedonkeruuta, ja toinen ohjaa kameraa ja kuvankäsittelyä. Ensimmäisenä rajapintana toteutettiin kamerarajapinnan, sillä se oli

selkeä yksittäinen kokonaisuus, eikä kameraluokalla ollut montaa riippuvuutta toisiin luokkiin. Kamerarajapintaa varten piti myös luoda kolmannen osapuolen sovellusta imitoiva pääohjelma, joka myöhemmin muokkautui esimerkkisovellukseksi, johon tehtiin kaikkien julkisten metodien esimerkit ja käyttökohteet. TrestimaService-rajapinta oli kokonaisuutena monimutkaisempi ja sen kehittäminen vaati monen alkuperäisen sovelluksen luokkatiedoston muuttamista siten, että ne toimisivat modulaarisessa versiossa ja dynaamisilla resursseilla.

Palvelurajapinnan kehityksessä ongelmana oli saada se pysymään ohjelman suorituksen ajan päällä. Esimerkki, jota käytettiin palvelun pohjana, suoritti vain yhden funktion ja lopetti palvelun. Oikeanlaisen taustaprosessin luomiselle löytyi toimiva keino, ja prosessi vaatii nyt erillisen lopetuskäskyn sammuakseen. Mikäli käsky unohtuu, jää prosessi taustalle käyntiin ja saattaa kuluttaa akkua ja prosessoritehoa tarpeettomasti.

Lopulta toimivaa testiversiota kokeiltiin kirjoitettujen käyttöohjeiden mukaan kehitystiimissä ja testin perusteella moduuli oli helppo ottaa käyttöön. Itse työn osuus oli haastava, mutta opettavainen. Opinnäytetyötä varten opetelluista keinoista ja metodeista on hyötyä jatkokehityksessä, sillä modulaarisia metodeita ei tarvitse enää opetella uudelleen ja niiden toteuttaminen vie kokemuksen myötä huomattavasti vähemmän aikaa.

LÄHTEET

Trestima. 2013. Trestima OY. Yritystiedot. Luettu 22.1.2015. https://www.trestima.com/company/

Anttonen, M., Sopanen, J. 2002. Relaskooppiarviointi. MetsäVerkko. Luettu 22.1.2015. http://virtuoosi.pkky.fi/metsaverkko/metsan_mittaus/relaskooppiarviointi/relaskooppiarviointi.htm

Rouvinen, T. 2014. Kuvia metsästä. Metsätieteen aikakausikirja 2/2014. Luettu 12.2.2014. http://www.metla.fi/aikakauskirja/full/ff14/ff142119.pdf

Nguyen, B. 2004. Modular Programming. Linux Dictionary. Luettu 22.1.2015. http://www.tldp.org/LDP/Linux-Dictionary/html/index.html

Blewitt, A. 2009. Modular Java: What Is It? InfoQ. Luettu 22.1.2015. http://www.infoq.com/articles/modular-java-what-is-it

Garza, G. 2011. Disadvantages of Structured Programming. Bright Hub. Luettu 3.2.2015. http://www.brighthub.com/internet/web-development/articles/73920.aspx

Dhillion, B. S., 1987. Reliability in Computer System Design.
New Jersey: Ablex Publishing Corporation.

Ganesh, S. 2014. Public, private, protected modifiers in Java. Java Samples. Luettu 22.1.2015. http://www.java-samples.com/showtutorial.php?tutorialid=655

Google. Service. Android Developers. Luettu 22.1.2015. http://developer.android.com/reference/android/app/Service.html

Google. Accessing Resources. Android Developers. Luettu 22.1.2015. http://developer.android.com/guide/topics/resources/accessing-resources.html

Google. Resources. Android Developers. Luettu 22.1.2015. http://developer.android.com/reference/android/content/res/Resources.html

Google. UI Overview. Android Developers. Luettu 22.1.2015. http://developer.android.com/guide/topics/ui/overview.html

Trestima OY, 2015. Yrityksen sisäinen materiaali. TrestimaModule/Readme.txt. Luettu 22.1.2015.

Heckman, S., Ho, D. & Williams, L. 2004. Javadoc. North Carolina State University. Luettu 22.1.2015. http://agile.csc.ncsu.edu/SEMaterials/tutorials/javadoc/

LIITTEET

Liite 1. Esimerkki moduulia käyttävästä pääohjelmasta

```
package com.testbed;

import com.trestima.TrestimaCamera;
import com.trestima.TrestimaService;
import com.trestima.TrestimaService.TrestimaServiceListener;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.LinearLayout;

// A way to include TrestimaServiceListener so all the methods can be inherited.
// It is not necessary for TrestimaCamera to run but is needed for reports.
public class MainActivity extends Activity implements TrestimaServiceListener{

    private Button mCamButton;              // button is inconvenient way to launch camera
    private Button reportButton;            // and so is for reports
    private String oneliner = "No Report";  // this one line report could be used as button text
    private String fullReport = "";         // full HTML format report
    private WebView reportView;             // this is a way to display the HTML report
    private LinearLayout repLayout;         // Layout used to change parameters

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Service intent needs Strings "username" and "password" in order to log in
        //to Trestima. Photos and reports will be added to this account.
        Bundle b = new Bundle();
        b.putString("username", "test.trestima.com##myUserName");
        b.putString("password", "abc123");
        Intent serviceIntent = new Intent(getBaseContext(), TrestimaService.class);
        serviceIntent.putExtras(b);
        startService(serviceIntent);
        // Manual session name can be changed, default name is "default"
        // If parcel exists and there are session IDs, the session id can
        // be changed to Trestima with:
        // TrestimaService.setSessionId("12345");
        TrestimaService.setSessionName("testi");
```

```
// Listener needs to be launched in order to get updates of reports and queue status.
TrestimaService.addListener(this);

mCamButton = (Button)findViewById(R.id.camButton);
mCamButton.setOnClickListener(new OnClickListener() {
    public final void onClick(View v) {
        // TrestimaCamera is an activity that needs to be launched with an intent.
        Intent intent =new Intent(MainActivity.this, TrestimaCamera.class);
        startActivity(intent);

        // This is to remove transition animation so the Camera starts faster
        overridePendingTransition(0,0);
    }
});

// Prepare layout for full HTML report
repLayout = (LinearLayout)findViewById(R.id.reportLayout);
repLayout.setVisibility(View.INVISIBLE);
reportView = new WebView(this);
repLayout.addView(reportView);

reportButton = (Button)findViewById(R.id.repButton);
reportButton.setText(oneliner);
reportButton.setOnClickListener(new OnClickListener() {
    public final void onClick(View v) {
        // Example of opening full HTML report with webView component
        if(fullReport != null && !fullReport.isEmpty()){
            reportView.getSettings().setLoadWithOverviewMode(true);
            reportView.getSettings().setUseWideViewPort(true);
            reportView.loadData(fullReport, "text/html; charset=UTF-8", null);
            LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.MATCH_PARENT,
                LinearLayout.LayoutParams.MATCH_PARENT);
            reportView.setLayoutParams(lp);
            repLayout.setVisibility(View.VISIBLE);
            reportView.setFocusable(true);
            reportView.setFocusableInTouchMode(true);
            reportView.requestFocus();
            // TODO: handle Back-key press
        }
    }
});
```

```
    }

    @Override
    protected void onDestroy()
    {
        super.onDestroy();
        // TrestimaServiceListener need to be removed
        TrestimaService.removeListener();
        // TrestimaService needs to be shut down because it's configured as
        // Sticky service which stays on for as long as it's stopped.
        // If it doesn't get stop call, it will stay running on the background
        // even if the main program is shut down.
        stopService(new Intent(getBaseContext(), TrestimaService.class));
    }

    @Override
    public void reportChanged(String onelinereport, String fullHtmlReport) {
        // Example of using one line report as a button text
        Log.e("main", "oneliner: "+onelinereport);
        reportButton.setText(onelinereport);
        fullReport = fullHtmlReport;
    }

    @Override
    public void queueChanged(int queueCount, boolean isSending, boolean networkError) {
        // Queue status provided, queue count could be displayed over camera button,
        // text color could be changed while queue is sending photo to cloud or if
        // there is a network error.
        Log.e("main", "queue status changed, queue count: " + queueCount+ " and is sending: "
        +isSending +
                ", was there network error? "+networkError);
    }
}
```