Matti Kivelä & Harri Närvänen

LIVE TRACKER

Opinnäytetyö
Tietojenkäsittelyn Koulutusohjelma

Maaliskuu 2010

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** 8. Maaliskuuta 2010
Tekijä(t) Matti Kivelä ja Harri Närvänen	**Koulutusohjelma ja suuntautuminen** **Tietojenkäsittelyn koulutusohjelma**
Nimeke Live Tracker	

Tiivistelmä

Opinnäytetyömme tarkoituksena oli yhdistää mobiililaitteella kerätyt paikkatiedot Google Mapsiin. Tämän ajatuksen pohjalta aloimme kehittää mobiilisovellusta, joka keräisi paikkatietoja ja jonka avulla ne olisi mahdollista lähettää edelleen palvelimelle ja näyttää reaaliajassa Google Maps-karttapohjalla Internet-sivulla. Opinnäytetyömme toimeksiantajana toimi Mikkelin ammattikorkeakoulu.

Opinnäytetyömme teoriaosuudessa perehdyimme satelliittipaikannuksen historiaan ja GPS-järjestelmän toimintaan, Google Mapsin ohjelmointirajapintaan sekä J2ME-mobiiliohjelmointiin. Ohjelmointiosuudessa tekemämme mobiilisovelluksen toteutimme Java-mobiiliohjelmointia ja siihen liittyviä luokkakirjastoja hyväksikäyttäen. Palvelinpään ratkaisut teimme PHP-ohjelmointia, xml-tiedostoja sekä Google Mapsin ohjelmointirajapintaa yhdistelemällä.

Opinnäytetyömme lopputuloksena saimme toteutettua tavoitteemme mukaiset sovellukset. Paikkatietoja hyödyntävien sovellusten jatkuvasti lisääntyessä tarjoaa opinnäytetyömme tarvittavat pohjatiedot tämäntyyppisten sovellusten jatkokehittämistä varten.

Asiasanat (avainsanat) Ohjelmointi, J2ME, Google Maps		
Sivumäärä 42 s. + liitteet 12 s.	**Kieli** Suomi	**URN** URN:NBN:fi:mamkopinn201091736
Huomautus (huomautukset liitteistä)		
Ohjaavan opettajan nimi Jukka Selin	**Opinnäytetyön toimeksiantaja** Mikkelin ammattikorkeakoulu	

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 8 March 2010
Author(s) Matti Kivelä and Harri Närvänen	**Degree programme and option** Business Information Technology
Name of the bachelor's thesis Live tracker	

Abstract

The purpose of our bachelor's thesis was to combine the geographic information collected on the mobile device to Google Maps. On the basis of this idea we begun to develop an application for mobile devices that would collect geographic information and which would be able to transmit them to server and show them in real-time on a Google Maps-based website. The bachelor's thesis was assigned by Mikkeli University of Applied Sciences.

The theoretical part of our bachelor's thesis introduced the history of satellite navigation systems and the functions of GPS, Google Maps application programming interface and J2ME Java platform for mobile devices. The mobile application that we made in the programming part of our studies was programmed on the Java mobile platform by using its class libraries. The server solutions were programmed by combining PHP programming with XML files and the Google Maps application programming interface. The applications we made were successful. With the increasing amount of applications that use geographic information, our bachelor's thesis provides basic information for developing these kinds of applications.

Subject headings, (keywords)

Programming, J2ME, Google Maps

Pages	**Language**	**URN**
42 pgs. + appendices 12 pgs.	Finnish	URN:NBN:fi:mamkopinn2010 91736

Remarks, notes on appendices

Tutor	**Bachelor´s thesis assigned by**
Jukka Selin	Mikkeli University of Applied Sciences

SISÄLTÖ

1 JOHDANTO

Opinnäytetyömme tutkimusongelmana on selvittää, kuinka mobiililaitteella kerättävät paikkatiedot saadaan yhdistettyä Google Mapsiin. Tarkoituksenamme on rakentaa tämän pohjalta mobiilisovellus, joka kerää ja lähettää reaaliaikaista paikkatietoa palvelimelle. Toisena tavoitteenamme on rakentaa Internet-selaimella toimiva sovellus, joka hakee ja näyttää paikkatiedot sekä mahdollistaa lähes reaaliaikaisen seurannan (”tracking”) Google Maps-sovelluksen avulla.

Opinnäytetyömme toimeksiantajana toimii Mikkelin ammattikorkeakoulu. Idea työstä on jalostunut mielissämme perehdyttyämme mobiilisovellusten tekoon ”Mobiili- ja tietoliikenne-ohjelmointi”-kurssilla. Aihetta tutkittuamme löysimme Ville Jukaraisen opinnäytetyön vuodelta 2006, joka käsittelee tiedonkeruuta mobiililaitteella. Opinnäytetyön jatkokehitysnäkymissä esitetään yhtenä vaihtoehtona tällaisen sovelluksen kehittämistä.

Opinnäytetyömme teoriaosuuden alussa esittelemme yleisesti satelliittipaikannusta ja sen historiaa. Sen jälkeen pureudumme syvemmälle GPS-järjestelmän rakenteisiin. Tämän jälkeen esittelemme Google Mapsia ja sen ohjelmointirajapintaa. Viimeisenä osana käymme läpi J2ME-mobiiliohjelmointia ja sen ominaisuuksia.

Rajaamme työmme teoriaosuudesta pois käyttöliittymän suunnittelun, tietoturvallisuuden, web-ohjelmoinnin ja sovellusten visuaalisen puolen pois, koska tällaisia opinnäytetöitä on jo tehty. Emme myöskään käy Google Mapsia kovinkaan yksityiskohtaisesti läpi, koska siitä on jo olemassa opinnäytetyö: ”Google Maps ja Google Maps API”, Antti Marttinen, huhtikuu 2007.

Teoriaosuuden kirjoittamisen jaoimme kahteen osaan. Harri kirjoitti satelliittipaikannuksesta ja Matti J2ME-ohjelmoinnista. Varsinaisen tutkimusongelman teorian kirjoitimme yhdessä, eli kuinka Google Mapsin ohjelmointirajapintaa hyväksikäyttäen saamme kerätyt paikkatiedot näkyviin kartalle.

2 SATELLIITTIPAIKANNUS JA GPS-JÄRJESTELMÄT

Satelliittipaikannuksella tarkoitetaan paikanmääritystä satelliittijärjestelmän avulla. Satelliittipaikannuksen avulla käyttäjä voi määrittää sijaintinsa, nopeutensa sekä saada tarkan ajan. Lähes kaikki satelliittipaikannusjärjestelmät on kehitetty alun perin sotilaskäyttöön. Eurooppalainen Galileo on ensimmäinen satelliittipaikannusjärjestelmä, joka on kehitetty alusta alkaen siviilien käyttötarpeet huomioon ottaen. Autonavigaattorien ja GPS-paikantimilla varustettujen matkapuhelimien yleistyessä sekä käsikäyttöisten paikantimien hintojen laskiessa on satelliittipaikannus lisääntynyt siviiliväestön keskuudessa räjähdysmäisesti.

2.1 Satelliittipaikannuksen historiaa

Satelliittipaikannuksen juuret ulottuvat aina 1960-luvun alkuun, jolloin Yhdysvaltain laivasto aloitti Transit-järjestelmänsä toiminnan. Järjestelmään kuului kuusi satelliittia, jotka kiersivät noin 1100 kilometrin korkeudessa. Vuonna 1967 sen käyttö sallittiin myös siviileille. Järjestelmän käyttö ja ylläpito loppui GPS:n syrjättäessä sen vuonna 1996. Yhdysvaltain puolustusministeriö teki päätöksen uudesta järjestelmästä vuonna 1973 ja ensimmäinen Navstar I- satelliitti laukaistiin avaruuteen vuonna 1978. GPS-järjestelmästä kerrotaan tarkemmin omassa luvussaan. (Poutanen 1998, 15; Miettinen 2006, 21.)

Vuonna 1976 aloitti toimintansa Neuvostoliiton armeijan kehittämä Tsikada. Järjestelmästä tuli hyvin pitkälti Transitin kaltainen. Tsikadaa seurasi ranskalaisten ja yhdysvaltalaisten yhdessä kehittämä Argos. Vuonna 1979 toimintansa aloittanut Argos keräsi tietoa maailmanlaajuisesti ja sitä hyödynnettiin paikannuksen lisäksi tieteellisen tiedon keräämisessä kuten maaperän ja merten tutkimisessa. Sillä päästiin noin 500 metrin paikannustarkkuuteen. (Poutanen 1998, 15 - 16; Miettinen 2006, 21.)

Vuonna 1982 oli vuorossa Cospas Sarsat. Sen kehittivät Yhdysvallat yhteistyössä Neuvostoliiton, Ranskan ja Kanadan kanssa palvelemaan pelastustoimintaa maailmanlaajuisesti. Cospas Sarsat-järjestelmä pitää sisällään hätälähettimiä laivoissa ja lentokoneissa. Ne reagoivat automaattisesti vedenpaineeseen tai törmäykseen. Cospas Sarsat on edelleen toiminnassa ja organisaation Internet-sivujen (www.cospas-sarsat.org)

mukaan järjestelmän avulla oli pelastettu vuoden 2008 loppuun mennessä 26 779 ihmistä 7 268 hätätilanteessa. Suomessa järjestelmän palveluja ei ole tarvittu viime vuosina. (Miettinen 2006, 22.)

Glonass (*Globalnaya Navigatsionnaya Sputnikova Sistema*) on Neuvostoliiton aikaan sotilaskäyttöön kehitetty satelliittipaikannusjärjestelmä. Sen toteutus on paljolti GPS:n kaltainen. Järjestelmän kehittäminen aloitettiin vuonna 1982, jolloin Neuvostoliiton ensimmäisen Glonass-satelliitti laukaistiin avaruuteen. Järjestelmä saavutti lopullisen laajuutensa vasta vuoden 1996 alussa. Tänä päivänä myös järjestelmän siviilikäyttö on sallittu ilman rajoituksia. Venäjällä on ollut kuitenkin suuria vaikeuksia pitää järjestelmä toimintakuntoisena. Toimintakuntoisten satelliittien vähäisestä määrästä johtuen paikannustarkkuus on ollut heikko. (Poutanen 1998, 25.)

Kehitteillä oleva Galileo on Euroopan Unionin ja ESA:n (*European Space Agency*) yhteinen hanke. Järjestelmän olisi tarkoitus olla yhteensopiva Glonass- ja GPS-järjestelmien kanssa ja se on nimetty italialaisen tieteilijän Galileo Galilein mukaan. Tarkoituksena on ollut kehittää alusta alkaen satelliittipaikannusjärjestelmä siviilikäyttöön, joka tarjoaa parempaa tarkkuutta ja toiminnan vakautta edeltäjiinsä verrattuna. (Miettinen 2006, 182 - 184.)

Galileo tulee tarjoamaan kolmenlaisia palveluja: avoimia, kaupallisia ja rajoitettuja. Avoin palvelu (*Open Service*) tulee olemaan ilmainen ja suunnattu yleishyödyllisiin tarkoituksiin kuten autonavigointi ja matkapuhelinpalvelut. Palvelun tarkkuus on 5 - 20 metriä. Kaupallinen palvelu (*Commercial Service*) on ammattikäyttöön tarkoitettu alueellinen ja maksullinen palvelu. Se tarjoaa käyttäjälleen parempaa tarkkuutta (0,1 - 10 metriä) sekä käyttövarmuutta. Julkisesti säännelty rajoitettu palvelu (*Public Regulated Service*) on tarkoitettu viranomaiskäyttöön ja sen kehittämisessä on kiinnitetty erityistä huomiota häirinnältä suojautumiseen. Tarkkuudeksi luvataan 1 - 6 metriä ja palvelun tarjoamat paikannukset sisältävät laatutakuun. (Airos, ym. 2007, 34 - 35.)

2.2 GPS

GPS on kaikkialla maailmassa toimiva satelliittipaikannusjärjestelmä. Se sai alkunsa vuonna 1973, kun Yhdysvaltain laivaston Timation- ja ilmavoimien Program 612B-ohjelmat päätettiin keskeyttää ja aloittaa korvaavan järjestelmän kehittäminen. Järjestelmä sai nimekseen Navstar (*Navigation System Using Timing and Ranging*) GPS (*Global Positioning System*). Ajatuksena oli saada järjestelmä, joka toimii maailmanlaajuisesti ja jonka toiminta perustuu tarkkaan ajanmääritykseen ja etäisyyden mittaukseen. (Miettinen 2006, 23.)

Sotilaallisiin tarkoituksiin kehitetyn järjestelmän tavoitteena oli kestää hyvin vihollisen häirintää sekä erilaisia luonnonilmiöitä. Ensimmäinen Navstar I-satelliitti laukaistiin avaruuteen vuonna 1978 ja ensimmäiset siirrettävät vastaanottimet otettiin käyttöön vuotta myöhemmin. Kuitenkin vasta vuonna 1995 järjestelmän katsottiin olevan täysin valmis. (Poutanen 1998, 19.)

2.2.1 Osat

GPS-järjestelmä koostuu kolmesta eri osasta:

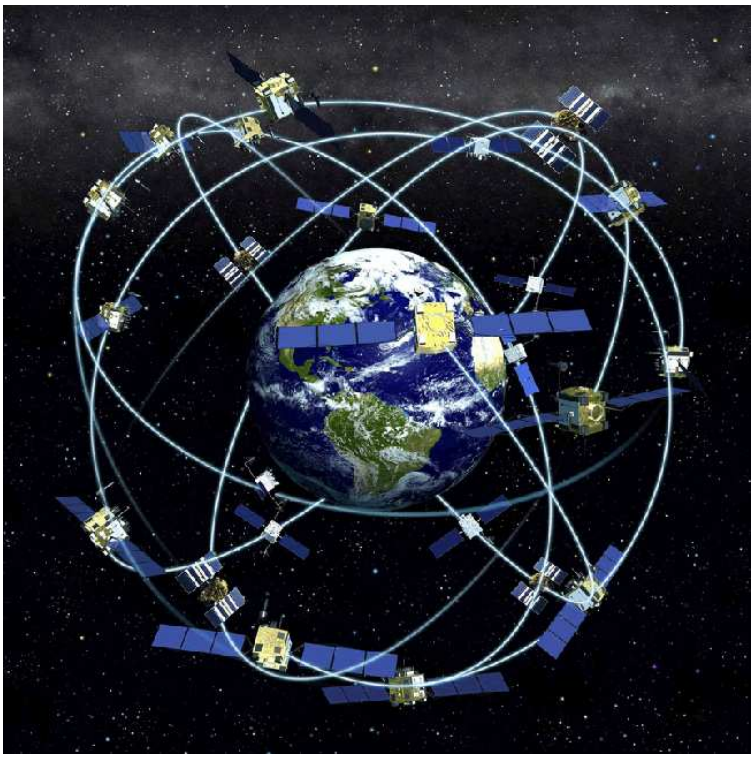
1. Avaruusosa (space segment)
2. Valvontaosa (control segment)
3. Käyttäjäosa (user segment)

Avaruusosaan kuuluvat satelliitit. Valvontaosa pitää sisällään satelliittien ja järjestelmän hallinnan. Käyttäjäosa tekee varsinaisen paikannuksen. (Miettinen 2006, 32.)

Avaruusosa

Avaruusosan muodostaa 24 toiminnassa olevaa satelliittia, jotka sijaitsevat 20 200 kilometrin korkeudessa. Järjestelmässä on myös varasatelliitteja korvaamassa vioittuneita satelliitteja sekä täydentämässä niiden sijoittumista taivaalle, joten yleensä satelliittien kokonaismäärä on noin 26 - 28. Kuvassa 1. on kuvattu, kuinka satelliitit sijoittuvat avaruudessa. Jokainen satelliitti kiertää maapallon kaksi kertaa vuorokaudessa ja niiden liikeradat ovat lähes samat päivästä toiseen. Tarkoituksena on, että käyttäjällä

olisi aina näkyvyys vähintään neljään satelliittiin ja parhaimmillaan jopa kahdeksaan. (Airos, ym. 2007, 20.)



KUVA 1. GPS-satelliittien sijoittuminen taivaalla

Valvontaosa

Valvontaosan muodostavat keskusasema, joka ohjaa ja valvoo järjestelmän toimintaa sekä 5 maa-antennia, jotka seuraavat tapahtumia taivaalla ja keräävät tietoja lähetettäväksi keskusasemalle ja 3 maa-antennia, jotka välittävät tiedot satelliitteihin. Päävalvontakeskus sijaitsee Colorado Springsissä Yhdysvalloissa, jossa sijaitsee myös yksi maa-antenneista. Muut maa-antennit sijaitsevat Havaijilla, Ascension Islandilla, Diego Garcialla sekä Kwajaleinillä. Ascension Islandilla, Diego Garcialla ja Kwajaleinillä on molemmat sekä tietoja keräävät, että lähettävät maa-antennit. (Poutanen 1998, 19 - 21; Miettinen 2006, 33.)

Käyttäjäosa

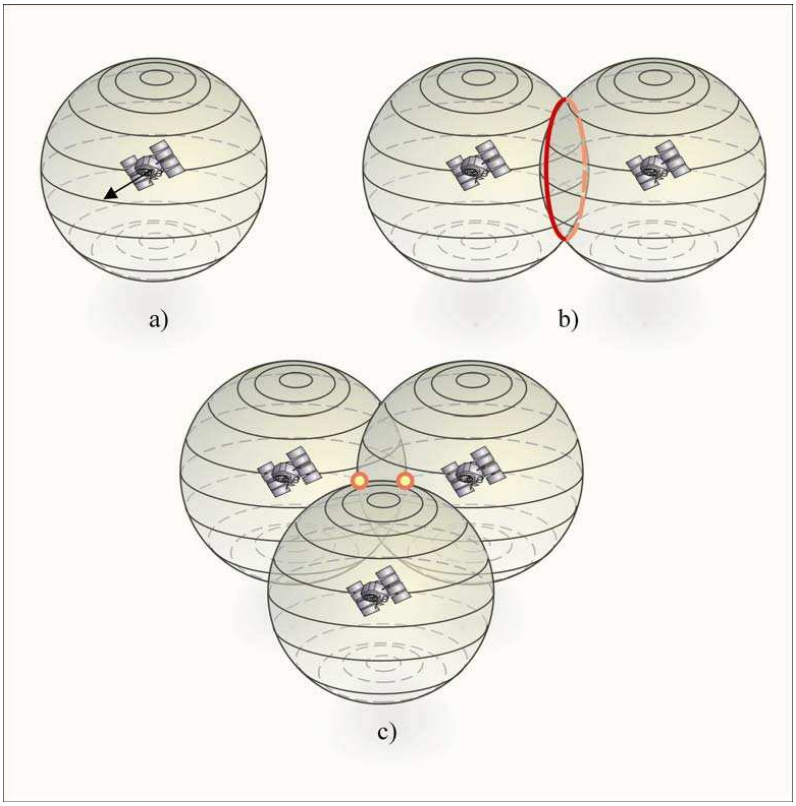
Käyttäjäosan muodostavat GPS-päätelaitteet. Päätelaitteille on tarjolla kahdenlaista palvelua. SPS (*Standard Positioning Service)* siviilikäyttöön ja PPS (*Precise Positioning Service*) Yhdysvaltain puolustusministeriön valtuuttamille käyttäjille (ns. GPS-

järjestelmän sotilaallinen osa). PPS on salattu siviilikäyttäjiltä ja ainoastaan puolustusministeriön valtuuttamilla käyttäjillä on avaimet salauksen purkamiseen. Salauksesta käytetään lyhennettä A/S (*Anti-Spoofing*). (Miettinen 2006, 33.)

2.2.2 Tärkeimmät periaatteet ja etäisyyden mittaus

Satelliittipaikannuksen kolme tärkeintä perusperiaatetta ovat:

1. GPS:n toiminta perustuu satelliittien ja vastaanottimen välisen etäisyyden avulla tehtävään kolmiomittaukseen eli trilateraatioon.
2. Etäisyys mitataan radiosignaalin kulkuajasta.
3. Etäisyyden tarkkaan mittaamiseen tarvitaan tarkkoja kelloja.



KUVA 2. Paikannuksen geometria

GPS-vastaanottimen sijainti määritetään mittaamalla etäisyydet taivaalla näkyviin satelliitteihin. Etäisyyden mittausta on havainnollistettu kuvassa 2. palloesimerkin avulla. Yhden satelliitin avulla tiedetään, että oma paikka on jossakin pallopinnalla mitatun etäisyyden päässä satelliitista (kohta a.). Kun mitataan etäisyys toiseen satelliittiin, saadaan rajattua sijainti kahden pallon leikkauspisteisiin eli ympyränkehälle (kohta b.). Etäisyydellä kolmanteen satelliittiin saadaan pallonpinta, joka leikkaa ympyränkehän kahdes-

sa pisteessä (kohta c.). Sijainti voidaan määrittää yksiselitteisesti, koska toinen pisteistä on joko syvällä maapallon pinnan alla tai äärettömän kaukana avaruudessa. GPS-vastaanotin tarvitsee toimiakseen tiedot vähintään kolmesta satelliitista, mutta korkeustietojen selvittämiseen tarvitaan vielä etäisyys neljänteen satelliittiin. Lisäksi tarkkuus paranee, mitä useampaan satelliittiin saadaan yhteys. (Airos, ym. 2007, 14.)

2.2.3 Tarkkuus

Siviilikäytössä GPS:n tarkkuus on vaakasuunnassa muutama metri ja korkeussuunnassa noin 2 - 3 metriä heikompi. Tarkkuus parani huomattavasti 1. toukokuuta 2000, jolloin Yhdysvaltain puolustusministeriö luopui tahallisen häirinnän eli SA:n (selective availability) käytöstä. Aiemmin tahallista häirintää käytettiin heikentämään siviilien käytössä olleiden paikantimien tarkkuutta. Tällöin ainoastaan Yhdysvaltain hallituksen valtuuttamilla käyttäjillä oli salakirjoitusavain, jolla SA:n vaikutus voitiin poistaa. (Miettinen 2006, 48.)

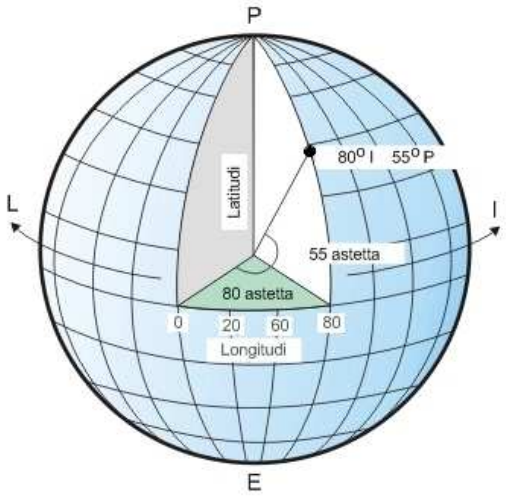
Tarkkuuteen vaikuttavat monet tekijät. Ilmakehä voi muuttaa radiosignaalin kulkemaa matkaa. Kellot aiheuttavat virhettä, vaikka kaikkinaiset sattumat ja virheet on pyritty teknisin keinoin poistamaan niin täydellisesti kuin mahdollista. Myös satelliittien kiertoradat saattavat muuttua. Ne on pyritty saamaan mahdollisimman vakioiksi ja pitkälle tulevaisuuteen ennustettaviksi, mutta avaruuden monet voimat voivat liikuttaa satelliitteja radaltaan. Muita tarkkuuteen vaikuttavia tekijöitä ovat paikantimen omat virheet, satelliittigeometrian vaikutus (satelliittien keskinäinen sijainti) sekä monitieheijastukset eli radiosignaalin heijastus jostakin lähellä olevasta pinnasta (esimerkiksi talon seinä, peltikatto, järven pinta, auton konepelti jne.). Tänä päivänä suurimman virheen tarkkuuteen aiheuttaa kuitenkin usein käyttäjä itse. Monesti käyttäjä voi asettaa laitteeseen vääränlaiset asetukset uutta paikanninta käyttöönotettaessa. Usein virhe tapahtuu valittaessa oikeaa koordinaattijärjestelmää, joten se ei ole sama jota Suomessa käytetään. (Miettinen 2006, 52 - 61.)

2.2.4 Koordinaatit

GPS käyttää pohjanaan maailmanlaajuista WGS84 (World Geodetic System 1984)-koordinaattijärjestelmää. Tähän ratkaisuun päädyttiin, koska haluttiin mahdollisimman tarkkaa paikkatietoa kautta maailman ja koska kansalliset koordinaattijärjestelmät kattavat ainoastaan hyvin pieniä alueita. Lisäksi WGS84 on oletuskoordinaatistona kaikissa GPS-paikantimissa. Koordinaatit muodostuvat latitudi- ja longituditiedoista. (Miettinen 2006, 166.)

Miettinen (2006) määrittelee latitudit eli leveyspiirit kirjassaan seuraavasti: ”Latitudit ovat maapallon ympärille kuviteltuja ympyröitä, jotka ovat tarkasti samansuuntaisia niin päiväntasaajan kuin toistensakin suhteen”. Latitudi ilmaisee paikan sijainnin maapallon pinnalla pohjois-eteläsuunnassa ja tarkoittaa kulmaa, jonka maapallon keskipisteestä sijaintipaikkaan piirretty vektori muodostaa päiväntasaaja-isoympyrän kanssa (kuva 3).

Longitudit eli pituuspiirit Miettinen (2006) määrittelee seuraavasti: ”Pituuspiirit ovat maapallon ympärille kuviteltuja ympyröitä, jotka kulkevat napojen kautta. Koska ne kulkevat sekä päiväntasaajan että napojen kautta, on pituuspiirin välimatka suurempi päiväntasaajalla. Mitä lähemmäs napoja tullaan, sitä lähempänä ne ovat toisiaan. Pituuspiirit yhtyvät maapallon navoilla”. Eli longitudi ilmaisee paikan sijainnin itä-länsisuunnassa ja tarkoittaa kulmaa, joka muodostuu valitun 0-meridiaanin ja sijaintipaikan kautta kulkevan isonympyrän väliin (kuva 3).



KUVA 3. Latitudi ja longitudi

2.3 GPS-havaintopisteiden välisen etäisyyden laskeminen

Jos halutaan tietää kahden GPS-havaintopisteen välinen etäisyys, se on mahdollista laskea tiedossa olevien havaintopisteiden koordinaateista. Koska maapallon pinta ei ole tasainen vaan kaareva, tarvitsee tämä ottaa huomioon laskettaessa etenkin pidempiä etäisyyksiä. Tätä varten on olemassa kaava, joka ottaa huomioon kaarevuuden. Kaavasta käytetään nimitystä Haversinen-kaava. Kaavan julkaisi yhdysvaltalainen tähtitieteilijä Roger W. Sinnott Sky & Telescope-lehden numerossa 68. vuonna 1984, vaikka kaava on oletettavasti paljon vanhempaa alkuperää. Nykyään kaavan hyödyntäminen on entistä helpompaa useiden eri ohjelmointikielien ansiosta. (Veness 2007.)

Kaava on seuraavanlainen:

$$a = \sin^2(\Delta lat / 2) + \cos(lat_1) \cdot \cos(lat_2) \cdot \sin^2(\Delta lon / 2)$$
$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)})$$
$$d = R \cdot c$$

, jossa:

$\Delta lat = lat_2 - lat_1$

$\Delta lon = lon_2 - lon_1$

R = maapallon keskimääräinen säde (6 371 km)

Kaava laskee matkan lyhintä reittiä eli linnuntietä. Se huomioi maapallon pinnan kaarevuuden, mutta ei ota huomioon muita korkeusvaihteluja, koska matkalle osuvien vuorien, notkojen yms. ennustaminen on laskennallisesti mahdotonta. Kulmien tulee olla laskutoimituksessa radiaaneina. (Veness 2007.)

3 GOOGLE MAPS API

Google Maps API (*Application Programming Interface*) on Googlen kehittämä ohjelmointirajapinta, jonka avulla pystytään käyttämään Googlen Mapsin tarjoamia karttapohjia millä tahansa Internet-sivulla javascriptia hyödyntäen. Kartalla on mahdollista

esittää paikkatietoja nuppineulojen (*markers*) tai reittiviivojen (*polylines*) avulla sekä liittää näihin informaatiota lisääviä puhekuplia (*info window*). Kartalla liikkuminen ja sen tarkempi tarkastelu onnistuu myös javascriptin avulla. Paikkatiedot on mahdollista kerätä GPS-vastaanottimella tai lisätä ne sivulle manuaalisesti. (Google Maps API 2009.)

Google Mapsin tarjoamat palvelut ovat käyttäjälleen ilmaisia, mikäli:

- sivustolle ei vaadita rekisteröitymistä ja se on kaikille käyttäjille julkisesti nähtävillä ja ilmainen
- sivustoa ei käytetä kaupallisiin tarkoituksiin tai yritykseen sisäiseen viestintään (intranet).

Käyttäjäryhmille, jotka eivät täytä näitä kriteerejä Google tarjoaa maksullista Google Maps API Premier-palvelua. (Google Maps API 2009.)

3.1 Käyttöönotto

Google Maps API:n käyttö vaatii rekisteröitymistä ja rekisteröitymisen yhteydessä saa käyttöönsä avainkoodin (*API key*), jonka avulla Google Mapsia on mahdollista hyödyntää omassa sovelluksessaan (kuva 4). Jotta pääsee rekisteröitymään ja saa avainkoodin käyttöönsä, tarvitaan myös Google-tili. Rekisteröitymisen yhteydessä kysytään Internet-sivun osoitetta, jolla Google Maps API-avainta on tarkoitus käyttää. Kun avainkoodi on luotu, niin sivulla näkyy myös muutama esimerkkikoodi siitä, kuinka avainkoodia voi hyödyntää. (Google Maps API 2009.)

KUVA 4. Google Maps API avainkoodi

3.2 Ohjelmointi

Googlen sivustolla on kattava ohjeistus esimerkkeineen siitä, kuinka mitäkin toimintoa voi hyödyntää ohjelmoitaessa sovellusta, joka käyttää Google Maps-ohjelmointirajapintaa. Jos ohjelmoitaessa ilmenee ongelmia, opastusta löytyy myös Googlen ylläpitämältä keskustelupalstalta, joka löytyy samasta osoitteesta. Opas löytyy osoitteesta http://code.google.com/intl/fi-FI/apis/maps/documentation. (Google Maps API 2009.)

Keskeisin elementti Google Maps API-sovelluksessa on itse kartta. Kartan ulkonäköä ja näkymää on mahdollista muokata kirjoittamalla javascript-koodiin GMap2-luokan eri muuttujia. GMap2-luokan eri metodit mahdollistavat mm. kartan skaalauksen, zoomauksen ja panoroinnin sekä kartan keskipisteen asettamisen. (Google Maps API 2009.)

TAULUKKO 1. GMap2-luokan addControl-metodi

Metodi	Palautusarvo	Kuvaus
addControl (GLargeMap-Control, GMapTypeCont-rol, GScaleControl)	Ei mitään	Metodi lisää ohjaimen karttaan, jolla voi esimerkiksi skaalata tai zoomata näky-mää.

Taulukossa 1 esitetyn metodin addControl-ohjaimilla voidaan muokata karttanäkymää ja lisätä siihen erilaisia säätimiä kartan tarkasteluun (taulukko 1). GLargeMapControl luo ohjaimen, jonka avulla kartan näkymää voidaan panoroida neljään suuntaan tai zoomata +/- sekä lukusäätimen zoomaukselle. GMapTypeControl-ohjain luo painikkeet, joiden avulla voidaan vaihtaa karttatyyppiä ja GScaleControl luo mittakaavajanan. (Google Maps API 2009.)

TAULUKKO 2. GMap2-luokan GEvent.addListener-metodi

Metodi	Palautusarvo	Kuvaus
GEvent.addListener	GEventListener	Metodi rekisteröi tapahtumankäsittelijän valitun objektin tapahtumalle.

Metodi GEvent.addlistener ”kuuntelee” sitä toimintoa, joka sille on annettu. Esimerkiksi hiiren klikkauksen havaittuaan se suorittaa siihen sisälletyn toiminnon. Metodin avulla voidaan esimerkiksi ohjelmoida toiminto, joka avaa puhekuplan tai infoikkunan, joka sisältää nuppineulaan liittyvää haluttua lisäinformaatiota, kun sitä klikataan hiirellä (taulukko 2). (Google Maps API 2009.)

TAULUKKO 3. GMap2-luokan GMarker-muodostin

Muodostin	**Kuvaus**
GMarker	GMarker merkkaa sijainnin kartalle luomalla nuppineulan (*marker*).

Nuppineulat havainnollistavat ja ilmaisevat halutun pisteen sijainnin kartalla (taulukko 3). GMarker-muodostimen avulla on mahdollista sijoittaa nuppineula kartalle paikoilleen havaintotietojen perusteella. Mikäli ei määritetä, minkälaista nuppineulaa halutaan käyttää, ohjelma käyttää oletuksena arvoa ”G_DEFAULT_ICON”. (Google Maps API 2009.)

4 J2ME

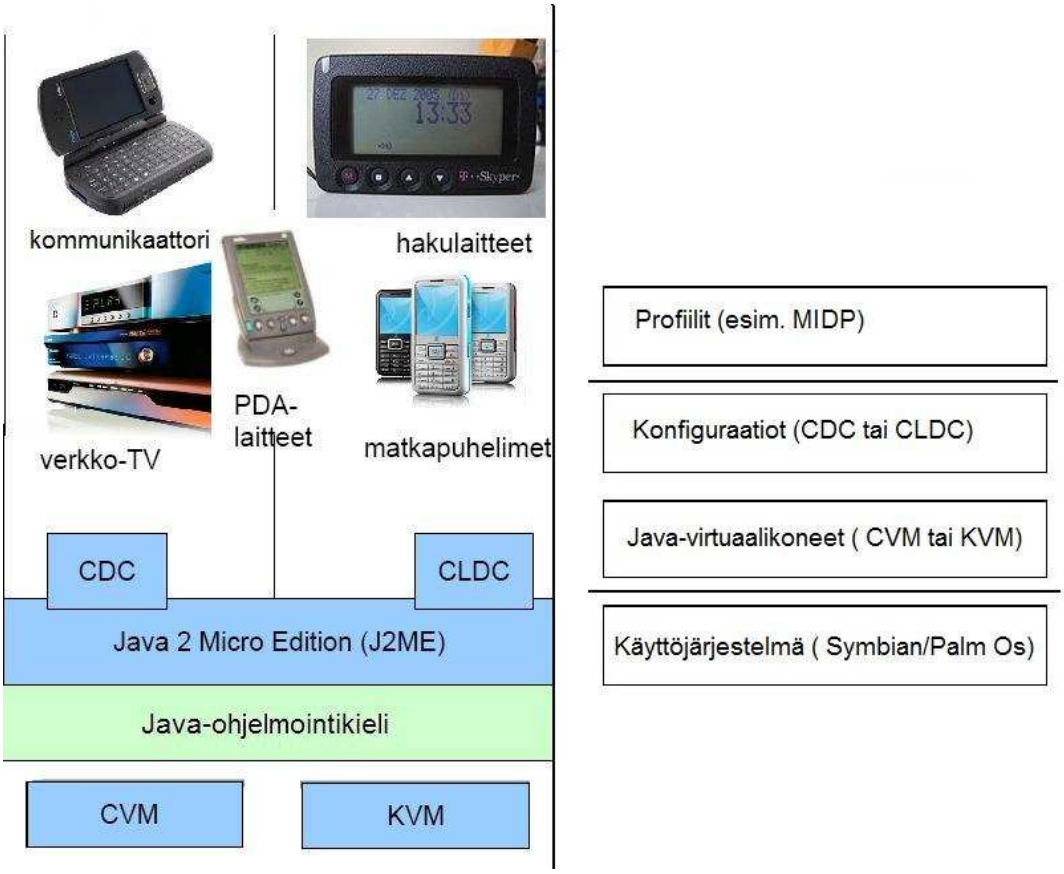
J2ME (Java 2 Micro Edition) on Sun Microsystemsin kehittämän Java-teknologian kevyehkö sovellusympäristö, joka on tarkoitettu sulautettujen järjestelmien sekä pienten, resursseiltaan rajoitettujen laitteistojen ohjelmistojen ohjelmointiin. Tämä tarkoittaa sitä, että esimerkiksi puhelimeen ohjelmia tehtäessä otetaan huomioon laitteen asettamat rajoitukset, kuten muistin ja näytön pieni koko ja silti saadaan aikaiseksi mahdollisimman käytettäviä ja hyödyllisiä sovelluksia. (Kontio 2002, 12.)

Koska kaikki kohdelaitteet eivät ole samanlaisia, vaan niissä saattaa olla hyvinkin suuria eroja, on J2ME enemmän tai vähemmän kompromissi. Laite-erojen vuoksi J2ME on jaettu arkkitehtuuriltaan eri tasoihin, joista jokainen hoitaa tiettyjä tehtäviä. Yhdessä ne muodostavat toimintaympäristön, jolloin ohjelmat toimivat eri laitteissa. (Peltomäki 2004, 5.)

Kuvan 5. vasemmalla puolella on esitetty J2ME:n arkkitehtuuri kohdealustoineen ja oikealla puolella sen eri kerrokset. Alimpana tasona on laitteen oma käyttöjärjestelmä

esimerkiksi Symbian OS tai Palm Os. Käyttöjärjestelmän päällä toimii jokin virtuaalikone, joita voivat olla esimerkiksi KVM tai CVM. Kuten arkkitehtuuri kuvasta näkyy, on siirrettävyys ratkaistu J2ME virtuaalikoneen ja konfiguraatioiden avulla. (Kontio 2002, 14.)

Jokin konfiguraatio saattaa vaatia tietynlaisen virtuaalikoneen. Tästä syystä kaikki J2ME-ohjelmat eivät ole täysin siirrettävissä järjestelmästä toiseen. Toisaalta tasot ovat ylöspäin yhteensopivia eli kaikki samaa konfiguraatiota tukevat laitteet tukevat siinä esitettyjä ominaisuuksia. (Peltomäki 2004, 5.) Virtuaalikoneen päällä arkkitehtuurissa on siis konfiguraatiot ja niiden päällä profiilit.



KUVA 5. J2ME:n arkkitehtuuri kohdealustoineen

4.1 Konfiguraatiot ja profiilit

Konfiguraatio määrittelee minimialustan laitteille, jotka voivat käyttää J2ME:tä. Toteutustasolla tämä tarkoittaa sitä, että tuoteryhmille, joilla on samanlaiset vaatimukset muistimäärälle ja prosessoriteholle, määritellään joukko ohjelmointirajapintoja. Kon-

figuraatio määrittelee Java-ohjelmointikielen, virtuaalikoneen ominaisuuksien tuen, tuettavat Java-kirjastot ja minimilaitteistovaatimukset. (Peltomäki 2004, 6.)

Konfiguraatio mahdollisuuksia on kaksi, CDC ja CLDC. CDC on tarkoitettu alustaksi kiinteästi verkkoon kytketyille 32-bittisille laitteille, joissa on yli kaksi megatavua muistia. Tällaisia laitteita ovat esimerkiksi Internetiin kytketty televisio, PDA-laitteet tai autonavigaattori. (Peltomäki 2004, 7.) CLDC on puolestaan tarkoitettu vielä rajoitetumpien laitteiden alustaksi, esimerkiksi kännyköille, älypuhelimille tai hakulaitteille. Käytännössä CLDC tarjoaa kokoelman luokkia, jotka on otettu suoraan J2SE:n API:sta tai lisätty siihen. (Kontio 2002, 14.)

Profiili on määritelmä (spesifikaatio), jossa kuvataan konfiguraation päälle rakentuva Java-rajapinta, joka tarjoaa tuen tietyntyyppiselle laitteelle esimerkiksi älypuhelimelle. Ohjelmat, jotka on tehty tietylle profiilille, toimivat jokaisessa saman profiilin tarjoavassa laitteessa. Profiilit laajentavat konfiguraatiota, ja niiden avulla voidaan hyödyntää laitteiden erityisominaisuuksia. Erilaisia profiileja voidaan käyttää esimerkiksi kahvinkeittimissä, pesukoneissa tai kännyköissä. Tunnetuin ja tärkein profiili on MIDP, joka on tarkoitettu mobiililaitteisiin, kuten hakulaitteet ja kännykät. Kaikki suurimmat matkapuheluvalmistajat tukevat sitä. (Peltomäki 2004, 12.)

4.2 MIDP

MIDP on CLDC:n päällä oleva profiili, siis standardi alusta, joka on tarkoitettu resursseiltaan pienten, pääasiassa langattomasti kytkettyjen mobiililaitteiden ohjelmointiin. MIDP-ympäristöön tehtyjä sovelluksia kutsutaan MIDleteiksi. Ne voivat olla yksittäisiä MIDletejä tai MIDlet Suiteja. Pääasiassa MIDletejä on kehitetty matkapuhelimiin, joissa on MIDP-tuki. (Peltomäki 2004, 20.)

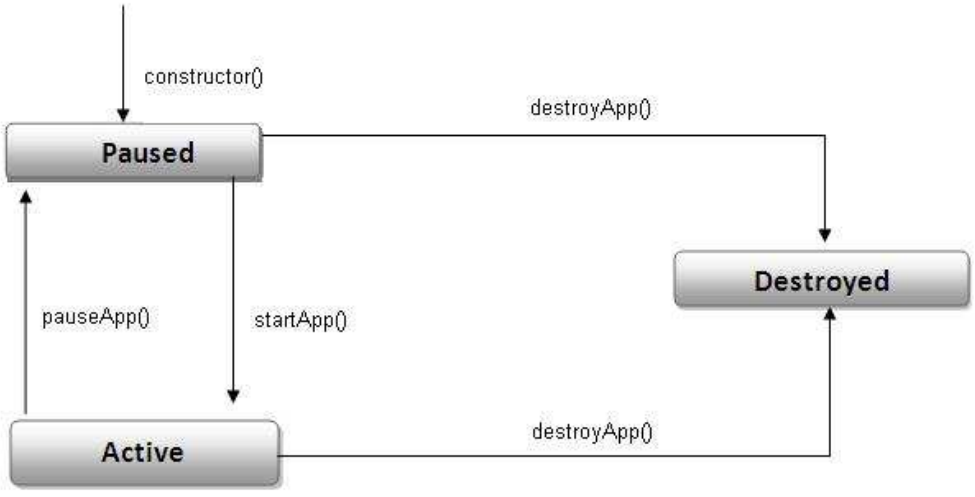
MIDP-profiilissa on kaksitasoinen käyttöliittymäarkkitehtuuri, joka on jaettu kahteen rajapintaan: matalan (low-level) ja korkean tason (high-level) APIin. Matalan tason API on tarkoitettu käytettävän sellaisissa sovelluksissa, joissa pitää hallita koko näyttöä ja piirtää grafiikkaa pikseli pikseliltä sekä käsitellä matalantason tapahtumia, kuten näppäinten painalluksia. Korkean tason API on tarkoitettu sovelluksille, joissa käyttöliittymä rakentuu valmiista graafisista komponenteista, kuten painikkeet, valintalistat

ja tekstikentät ja joiden ei tarvitse käsitellä käyttöliittymää kovin tarkasti. Tällaisia sovelluksia voivat olla esimerkiksi yrityssovellukset, joiden varsinainen raskas tietojenkäsittely, kuten tietokanta haut ja muut raskaat palvelut tapahtuvat palvelinkoneilla ja MIDP-laitteessa on vain kevyt palvelimia käyttävä asiakassovellus. (Kontio 2002, 66.)

4.2.1 MIDletin elinkaari

Kaikki MIDletit ohjelmoidaan javax.microedition.midlet.MIDlet-luokan ilmentyminä. Ennen kuin syvennytään MIDlet-ohjelmointiin, on syytä tuntea ja ymmärtää MIDletin elinkaari(malli), joka muistuttaa www-selaimella suoritettavia appletteja. MIDletien asennuksesta ja suorituksesta vastaa aina sovellusmanageri (application manager). MIDlet asennetaan siirtämällä luokkatiedostot päätelaitteelle, esimerkiksi kännykkään. Nämä luokkatiedostot voidaan pakata JAR-pakettiin (Java Archive) ja sen mukana kulkeva kuvailutiedosto (descriptor file) kuvaa kyseisen JAR-paketin sisällön. Tämän kuvailutiedoston tarkennin on .jad. (Peltomäki 2004, 22.)

MIDletejä ohjaa elinkaariajattelu. Jokaisen MIDP-sovelluksen pääluokan tulee periä MIDlet-luokka. Tämä luokka sisältää kolme tärkeää ylikirjoitettavaa metodia: StartApp(), destroyApp() ja pauseApp(). Nämä metodit kuvaavat eri tiloihin siirtymistä. Esimerkiksi pauseApp()-metodissa sovelluksen pitäisi vapauttaa kaikki käytetyt resurssit. MIDletin tilat ja niiden vaihtuminen näkyvät kuvassa 6. (Kontio 2002, 66.)



KUVA 6. MIDletin elinkaari

MIDletin elinkaari voi koostua seuraavista vaiheista:

1. Kun MIDletiä suoritetaan, luodaan MIDlet-luokan ilmentymä kutsumalla sen parametritonta konstruktoria. MIDletin alustajan suorittamisen jälkeen siirrytään Paused-tilaan.
2. Seuraavaksi sovellusmanageri kutsuu startApp() –metodia, joka siirtää MIDletin Active-tilaan.
3. Sovellusmanageri voi väliaikaisesti pysäyttää Active-tilassa olevan MIDletin pauseApp() –metodilla. Tällöin MIDlet siirtyy takaisin Paused-tilaan.
4. Sovellusmanageri voi lopettaa MIDletin suorituksen kutsumalla destroyApp() –metodia. Tällöin MIDlet tuhotaan laitteelta roskienkeruun (garbage collection) avulla ja se siirtyy Destroyed-tilaan. (Peltomäki 2004, 22.)

Kuvassa 7. näkyy MIDletin runko, jossa on konstruktori ja kaikki kolme tärkeää metodia. Esimerkin alussa näkyvät myös MIDletin tarvitsemat import-lauseet.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class midletinRunkoEsimerkki
    extends MIDlet
    implements CommandListener {
        public midletinRunkoEsimerkki() {
            // ...tänne jotain koodia
        }

        public void startApp() {
            // ...tänne jotain koodia
        }

        public void pauseApp() {
            // ...tänne jotain koodia
        }

        public void destroyApp(boolean b) {
            // ...tänne jotain koodia
        }
}
```

KUVA 7. MIDletin runko

4.2.2 Kirjastot

MIDP:ssä on käytössä CLDC:n luokat, mutta lisäksi MIDP:hen on määritelty joukko omia kirjastoja. Näistä ehkä tärkein on **javax.microedition.midlet**-paketti, joka sisältää **MIDlet**-luokan, josta kaikki MIDletit pitää periä. Käyttöliittymäluokat sijaitsevat **javax.microedition.lcdui**-paketissa. Tiedon tallennuksen mahdollistava RMS löytyy **javax.microedition.rms**-paketista. Viimeinen, verkkoyhteksien mahdollistava kirjasto, löytyy **javax.microedition.io**-paketista. (Kontio 2002, 75.) Näiden lisäksi on mahdollista linkittää ulkopuolisia kirjastoja sovellusten käyttöön. Tällaisia ovat esimerkiksi **java.util-paketti**, josta saadaan kalenterin- ja ajanhallintaan tarvittavia luokkia sekä **javax.microedition.location**-paketti, jonka avulla saadaan paikkatiedot sovelluksen käyttöön. Tulevissa luvuissa käymme taulukoin lävitse kirjastojen tärkeimmät rajapinnat ja luokat opinnäytetyömme kannalta. Emme listaa sellaisia rajapintoja ja luokkia, joita emme tarvitse opinnäytetyössämme. Tarvittavat metodit esittelemme opinnäytetyömme empiirisessä osassa. Kaikki rajapinnat ja luokat löytyvät muun muassa osoitteesta:
http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-4AEC8DAF-DDCC-4A30-B820-23F2BA60EA52/overview-summary.html.

Javax.microedition.lcdui

Tästä paketista löytyvät tarvittavat käyttöliittymäluokat. Ne jakautuvat korkean ja matalan tason luokkiin. Lcdui- ei kuulu CLDC:hen. (JSR-118 2007.)

TAULUKKO 4. Lcdui-paketin rajapinnat ja luokat

Paketista löytyvät rajapinnat:	
CommandListener	Rajapinnan avulla sovellus rekisteröidään kuuntelemaan korkean tason käyttöliittymäluokkien herättämiä tapahtumia.
Paketista löytyvät luokat:	
Command	Command on komentojen tiedot kapseloiva luokka.
Display	Hallitaan näyttöä. Jokaisella MIDletillä on vain yksi

	Display-luokan ilmentymä.
Displayable	Displayable on abstrakti luokka, josta kaikki koko näytön käyttävät korkean tason käyttöliittymäluokat on periytetty.
Form	Form-luokka mahdollistaa erilaisten lomakkeiden teon. Lomakkeet voivat sisältää mm. tekstikenttiä. Lomakkeella olevat elementit ovat Item-luokasta perittyjä.
Item	Item-luokka on lomakkeen jäsenten yläluokka, josta kaikki lomakkeelle asetettavat luokat peritään.
TextBox	TextBox on Screen-luokasta peritty, koko näytön käyttävä, muokattava tekstikenttä.
TextField	TextField on lomakkeelle sijoitettava tekstikenttä, johon voidaan syöttää muokattavaa tekstiä.

Javax.microedition.location

Location API on rajapinta lisäys J2ME:hen, joka mahdollistaa Java-kehittäjien tehdä paikkatietoisia sovelluksia resursseiltaan rajoittuneille laitteille. Se tarjoaa informaatiota laitteen fyysisestä sijainnista Java-ohjelmille. Location API on valinnainen paketti, jota voi käyttää monilla eri J2ME-profiileilla. Vähimmäisvaatimus toiminnalle on J2ME, CLDC v1.1 tai CDC v1.0 konfiguraatio. (JSR-179 2007.)

TAULUKKO 5. Location API:n rajapinnat ja luokat

Paketista löytyvät rajapinnat:	
LocationListener	Kuuntelija LocationProvider tyyppisille tapahtumille.
Paketista löytyvät luokat:	
Coordinates	Coordinates-luokka tarjoaa koordinaatit paikasta leveys-pituus-korkeus muodossa.
Criteria	Kriteerit joita käytetään valitessa palvelua, joka tarjoaa paikkatiedot, kuten sallittu heitto paikannuksessa.
Location	Location-luokka pitää sisällään perustiedot paikannusin-

	formaatiosta. Tämän luokan kautta haetaan muun muassa koordinaatit.
LocationProvider	Aloituspaikka ohjelmille, jotka käyttävät tätä rajapintaa. Tarjoaa myös paikannusinformaation lähteen. Location-Provider valitaan annettujen kriteerien perusteella.
QualifiedCoordinates	Leveys-pituus-korkeus tiedot tarkkuusarvon kanssa.

Java.util

Kalenteriluokat mahdollistavat päiväysten käsittelyn ja hallinnan. Tämän luokan avulla voidaan käsitellä ja tutkia päiväyksiä monipuolisesti. TimeZone-luokan avulla voidaan asettaa haluttu aikavyöhyke. (Kontio 2002, 58.)

TAULUKKO 6. Kalenterin ja ajanhallinnan luokat

Paketista löytyvät luokat:	
Calendar	Calendar-luokan avulla voidaan käsitellä ja tutkia päiväyksiä.
Date	Date-luokan avulla voidaan käsitellä ja tutkia päivänaikoja.
TimeZone	TimeZone-luokan avulla voidaan asettaa haluttu aikavyöhyke.

Java.lang/io/Javax.microedition.io

Java.io- ja java.lang-paketeista löytyvät seuraavat taulukossa 7. esitetyt luokat. HttpConnection-luokka löytyy javax.microedition.io-paketista. Luokat ovat erittäin tärkeitä esimerkiksi muodostettaessa yhteys WWW-palvelimeen. (Kontio 2002, 57.)

TAULUKKO 7. Java.io/lang/Javax.microedition.io-pakettien luokat

Paketeista löytyvät luokat:	
String	Merkkijono
StringBuffer	Mahdollistaa merkkijonojen yhdistämisen ilman +-operaatiota.
Thread	Mahdollistaa säikeitten ajamisen.
InputStream	Mahdollistaa tietovirran lukemisen.
HttpConnection	Mahdollistaa http-yhteyden muodostamisen.

Javax.microedition.rms

MIDP määrittelee yksinkertaisen tietuekannan, jonka avulla käyttäjät voivat tallentaa pysyvää tietoa laitteen muistiin. Tämän tietuekannan nimi on Record Management System. Tietuekanta on käytännössä tiedosto, joka koostuu tietueista. Tietuekantaan tallennettava tieto tallentuu tavuvektoreina. (Kontio 2002, 58.)

TAULUKKO 8. RMS-paketista löytyvät rajapinnat ja luokat

Paketista löytyvät rajapinnat:	
RecordEnumeration	Tarjoaa metodit tietuekannan läpikäyntiin molempiin suuntiin ja tietueiden hakemisen.
Paketista löytyvät luokat:	
RecordStore	Luokan avulla hallitaan tietuekantojen luominen, avaaminen ja poisto.

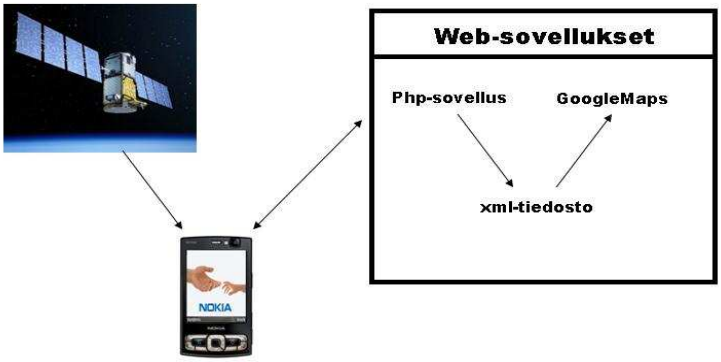
5 SOVELLUKSET

Aloitimme työmme tutustumalla jo olemassa oleviin paikkatietoja ja Google Mapsia hyödyntäviin sovelluksiin. Etsimme myös opinnäytetöitä, jotka mahdollisesti sivusivat aihettamme. Ensimmäinen työvaihe oli mobiilisovelluksen suunnittelu ja ohjelmointi ja tämän jälkeen Internet-selaimella toimivan sovelluksen teko. Viimeisenä työvaihee-

na tutustuimme Google Mapsin käyttöön ja opettelimme liittämään mobiililaitteella kerätyt tiedot siihen.

5.1 Sovellusten kuvaukset

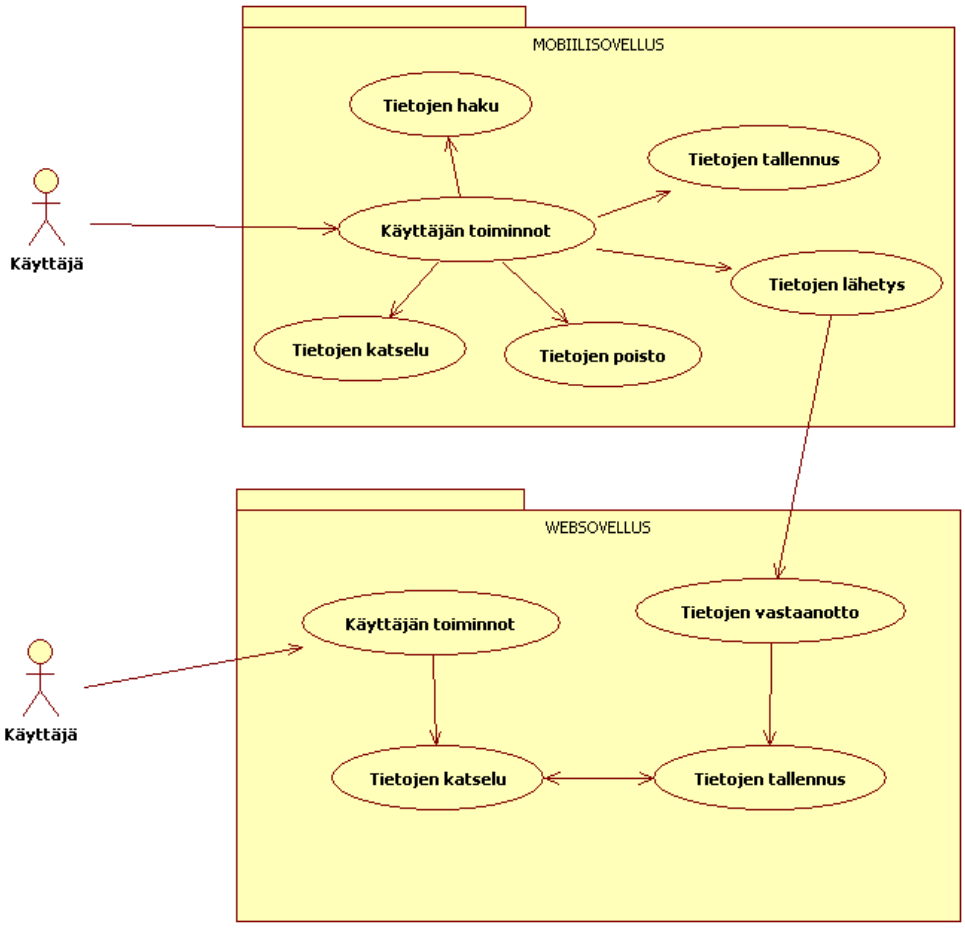
Opinnäytetyönämme toteutimme mobiililaitteella toimivan sovelluksen, jolla voi hakea, tallentaa tai lähettää paikkatietoja. Haetut paikkatiedot lähetetään puhelimelta palvelimelle, jossa palveluun vastaava sovellus tallentaa tiedot xml-dokumenttiin. Tämän jälkeen tiedot ovat käytettävissä Google Mapsilla tapahtuvaa seurantaa varten. Kuvassa 8. on esitelty yleiskuvaus kehittämästämme järjestelmästä.



KUVA 8. Järjestelmän yleiskuvaus

5.2 Suunnittelu

Aloitimme sovellusten suunnittelemisen laatimalla StarUML-ohjelmalla kuvan 9. mukaisen käyttötapauskaavion. Kuvan yläosassa on esitelty mobiililaitteella tapahtuvat käyttäjän toiminnot, joita ovat päivämäärän, kellonajan sekä latitudi- ja longitudikoordinaattitietojen hakeminen. Tämän jälkeen käyttäjä voi tallentaa tiedot puhelimen omaan muistiin, lähettää ne eteenpäin palvelimelle tai hakea ne vielä puhelimen muistista esiin näytölle ja poistaa ne. Kuvan alaosassa on kuvattu web-sovelluksen käyttötapaukset. Palvelimella toimiva php-ohjelma ottaa kännykältä saapuvan datan talteen ja tallentaa sen xml-tiedostoon. Tämän jälkeen käyttäjä voi tarkastella web-selaimen avulla, missä pisteet sijaitsevat kartalla.



KUVA 9. Käyttötapauskaavio

5.3 Toteutus

Tavoitteenamme oli rakentaa sellainen mobiilisovellus, jota pystyisimme ajamaan omilla matkapuhelimillamme, joita ovat Nokian N95 ja 6220 Classic. Molemmissa puhelinmalleissa on sisään rakennettu GPS-vastaanotin ja ne tukevat Java-sovelluksia.

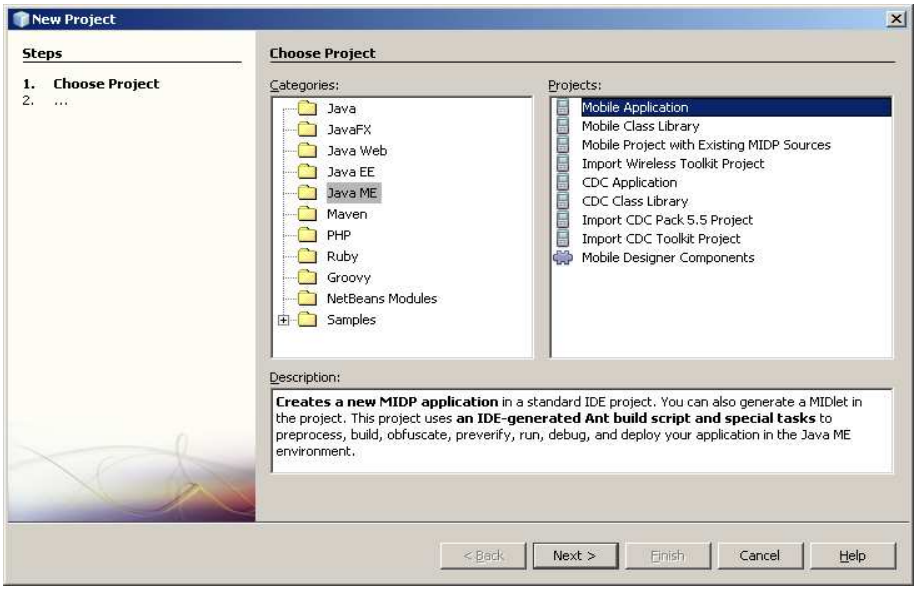
5.3.1 Kehitysympäristö

Mobiilisovelluksen toteutusta varten valitsimme kehitysympäristöksemme NetBeans IDE 6.7.1:n. Se on Sun Microsystemin tarjoama ilmainen kehitysympäristö/ohjelmointityökalu, jolla voidaan kirjoittaa, kääntää, debugata ja ottaa käyttöön sovelluksia. Ohjelma oli meille jo ennestään tuttu aiemmilta Java-ohjelmointikursseilta. NetBeansin voi ladata itselleen osoitteesta www.netbeans.org. Samalta sivulta löytyy myös tarvittava SDK-paketti, jossa on java-kääntäjä ja jonka avulla luodaan jar-

paketit. NetBeansiin on saatavana myös Mobility Pack-laajennus, joka mahdollistaa J2ME-sovellusten tekemisen. Tämä laajennus tuo NetBeanssiin kaikki J2ME-ohjelmoinnissa tarvittavat rajapinnat, kuten luvussa 4.3.2 esitellyn Location API:n. Netbeansin voi integroida myös J2ME Wireless Toolkit-työvälineeseen, joka tarjoaa joukon kehittyneitä ominaisuuksia kuten emulaattorit, joiden avulla tehtyjä sovelluksia testataan graafisesti ja toiminnallisesti oikeita matkapuhelimia muistuttavissa ympäristöissä. Tästä on suuri hyöty siinä, ettei sovelluksia tarvitse aina siirtää oikeaan matkapuhelimeen testausta varten.

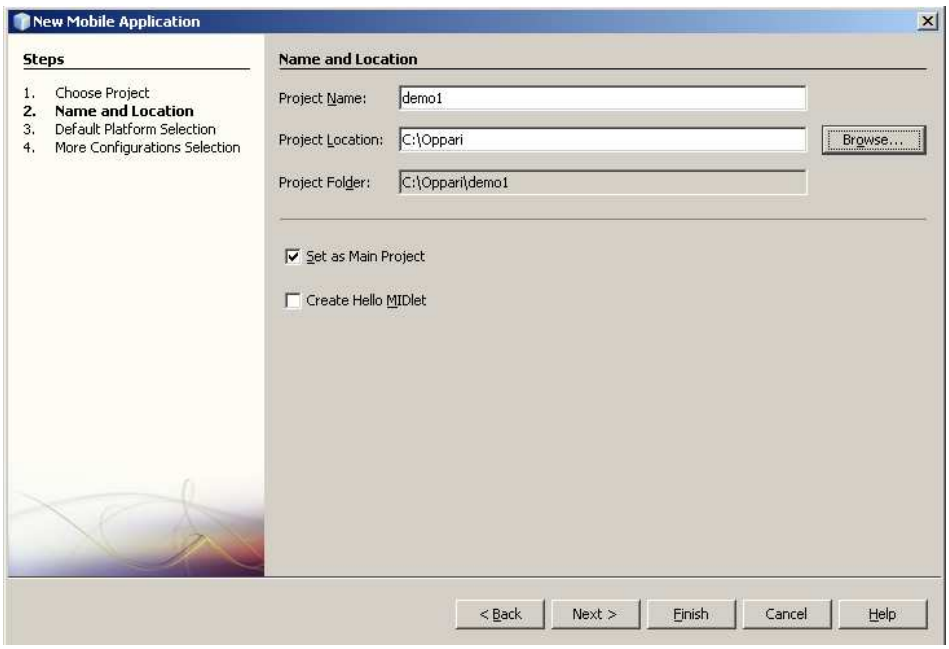
5.3.2 Uuden projektin luominen

Mobiilisovelluksen rakentamisen NetBeansilla aloitimme uuden projektin luomisella. Tämä tapahtui kuvan 10 mukaisesti valitsemalla: File >New Project >Java Me >Mobile Application ja painamalla Next>.



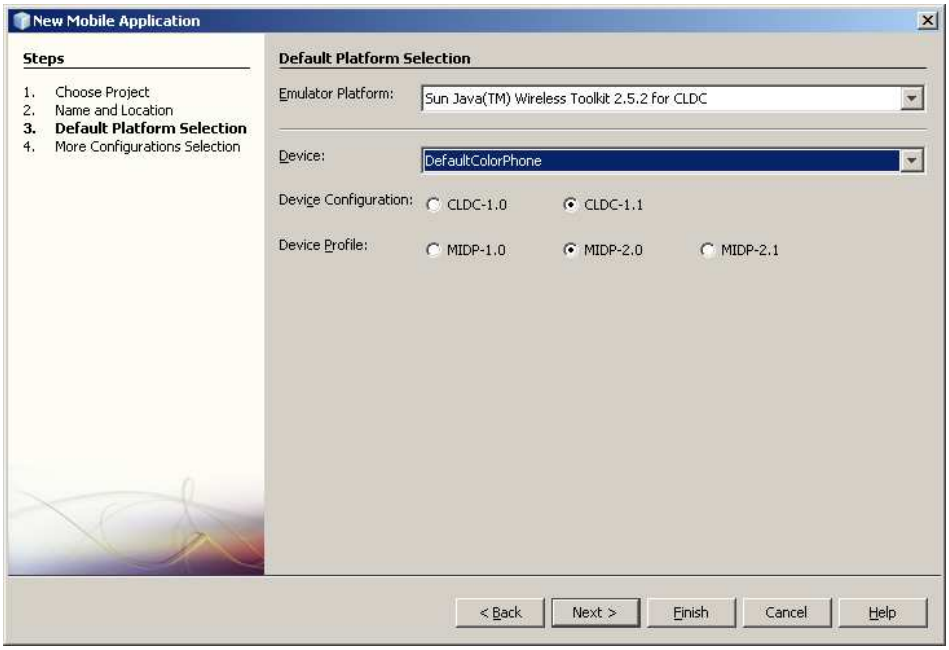
KUVA 10. Projektityypin valinta

Tämän jälkeen annoimme projektille nimen ja määritimme mihin kansioon projekti sijoitetaan. NetBeans asettaa demo1-projektin ns. pääprojektiksi (kuva 11).



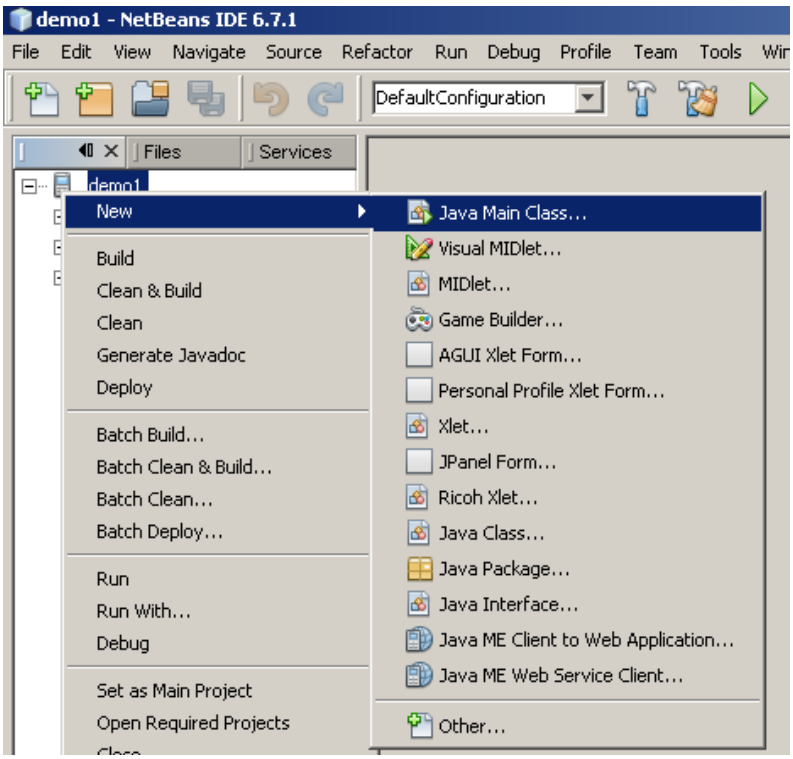
KUVA 11. Projektin nimeäminen

Seuraavaksi valitsimme emulaattorin, jolla oletusarvoisesti ajetaan sovellusta testausvaiheessa. Tärkeimmät kohdat, jotka pitivät täyttää, olivat konfiguraatio ja profiili. Näihin valitsimme CLDC-1.1:n ja MIDP-2.0:n (kuva 12).



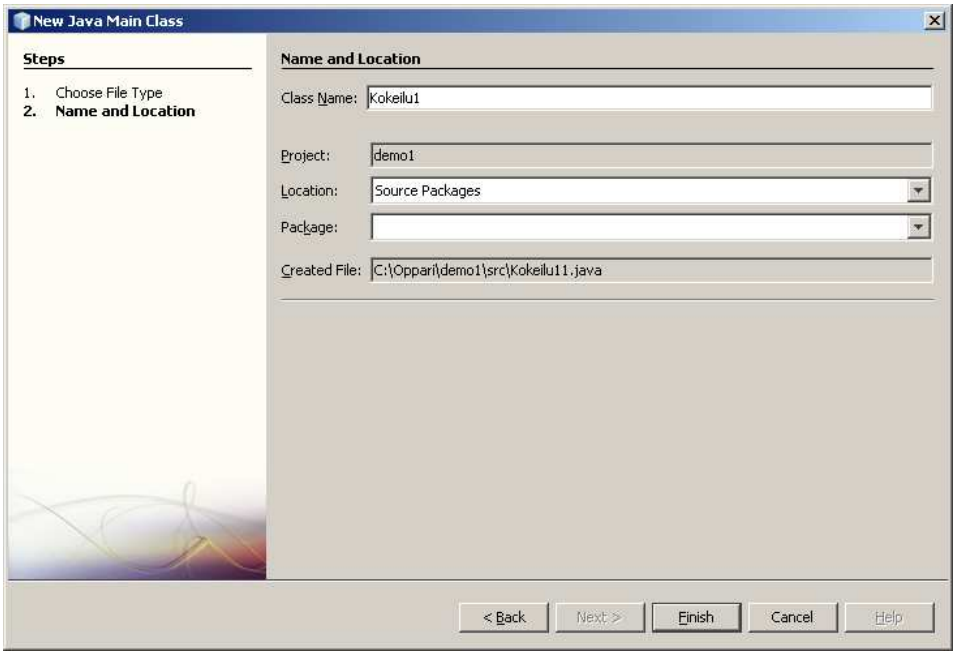
KUVA 12. Emulaattorin sekä konfiguraation ja profiili määrittäminen

Tämän jälkeen tulee vielä luoda uusi MIDlet projektiin painamalla demo1:n päällä hiiren oikeanpuoleista nappia ja valitsemalla: New >Java Main Class…(kuva 13).



KUVA 13. MIDletin luominen

Tästä aukeaa kuvan 14 mukainen näkymä, johon kirjoitetaan MIDletin nimi ja valitaan sille sijainti.



KUVA 14. MIDletin nimeäminen

5.3.3 Kokeilu1

Mobiilisovelluksemme koostui kolmesta erinäytöstä ja niihin liitetyistä seitsemästä käskystä. Ensimmäisen näytön rakensimme *Form*-luokalla, johon liitimme neljä *TextField*-luokan tekstikenttää ja viisi eri käskyä *CommandListener*-rajapintaa hyväksikäyttäen. Liitteessä 1. on esitelty lähdekoodi, jossa on näkyvissä tarvittavat importlauseet, elementit ja komennot. Lisäksi *startApp()*-metodissa näkyy, kuinka *TextField*-elementit lisätään *Form*ille ja lisätään käskyt läpipäästävä *CommantListener()*-metodi. Viimeiseksi asetetaan näyttö näkyviin *display.setCurrent()*-metodilla. Kuvan 15. vasemmalla puolella on näkymä sovelluksen ensimmäisestä näytöstä. **Menu**-valikkoa painamalla saa näkyviin lisää toimintoja kuvan 15 oikean puolen näytön mukaisesti.

KUVA 15. Ensimmäinen näyttö ja menu-valikon takaa löytyvät toiminnot

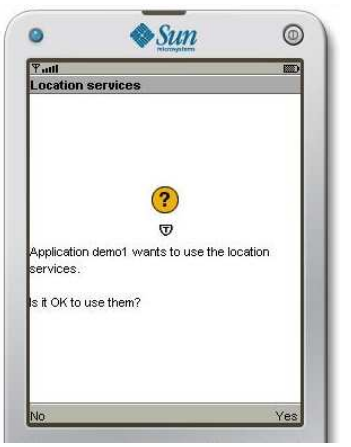
Lopeta-valikosta sovellus suljetaan ohjelmallisesti (kuva 16).

```
1  public void commandAction(Command c,Displayable s) {
2        if (c==poistu) {
3           destroyApp(false);
4           notifyDestroyed();
5        } // if poistu
6  }
7
```

KUVA 16. Sovelluksen lopettaminen ohjelmallisesti

Hae tiedot

Kun käyttäjä valitsee **Hae tiedot**-toiminnon, sovellus kysyy, saako se käyttää paikannuspalveluja (kuva 17. näyttö). Tähän käyttäjän tulee vastata ”kyllä”. Ohjelmallisesti sovellus luo uuden säikeen ja kutsuu *hae_tiedot()*-aliohjelmaa (kuva 17. koodi), jonka lähdekoodi on esitelty liitteessä kaksi. Tämä toiminto on ajettava omassa säikeessään, jotta koko ohjelman ajaminen ei vaarantuisi. Jos sovellus ei pysty ajamaan metodia, se ilmoittaa käyttäjälle, että ”Joku pätkii!!!”.




```
public void commandAction(Command c,Displayable s) {
    if (c==hae_tiedot) {
            Thread saie = new Thread() {
            public void run() {
            try {
                 hae_tiedot();
            } catch (Exception e) {
                 kentta3.setString("Joku pätkii !!!");
                }
             }
          };
             saie.start();
    } // if hae_tiedot
}
```

KUVA 17. Saako ohjelma käyttää paikannusta ja hae tiedot-komento koodissa

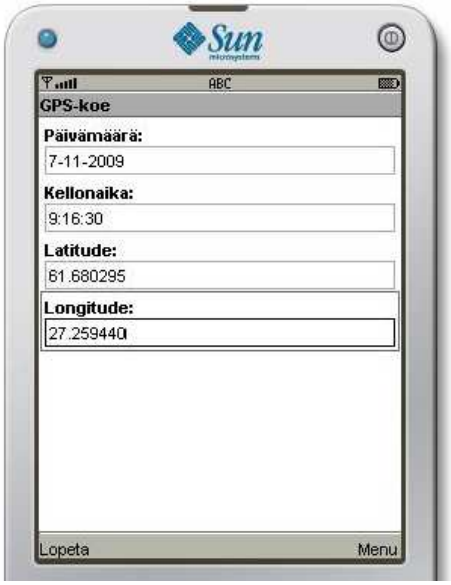
Kun *hae-tiedot()*-aliohjelmaa kutsutaan, se tutkii, tukeeko puhelin paikkatietojen hakua metodilla *System.getProperty*(”microedition.location.version”). Tämä metodi palauttaa arvonaan rajapinnan versionumeron. Jos puhelin ei tue rajapintaa tulee paluuarvona tyhjä merkkijono tai null-arvo ja tästä ilmoitetaan käyttäjälle: "Puhelimellasi ei voi hakea paikkatietoja!!!". Jos puhelimesta löytyy tarvittava rajapinta, niin luodaan uusi olio, jolla määritetään paikannusehdot. Tämä tapahtuu *Criteria*-luokan avulla. Kun on määritelty halutut ehdot, joilla paikkatietoja haetaan, voidaan *LocationProvider*-luokalta pyytää ilmentymä. Paluuarvona on *LocationProvider*-luokan ilmentymä määrätyillä kriteereillä.

Kun paikannustietojen välittäjä on saatu, voidaan sen kautta pyytää paikannusta. Paikannustiedot saadaan *LocationProvider* ilmentymästä metodilla *getLocation(int timeOut)* joka palauttaa arvonaan *Location*-luokan ilmentymän, jonne paikannustiedot on tallennettu. Kokonaisluku, joka on mahdollista antaa parametrina *getLocatio*-me-

todille, kertoo ohjelmalle missä ajassa paikannuksen tulisi viimeistään tapahtua. Tämä parametri annetaan sekunteina.

Seuraavaksi paikannustiedot otetaan talteen luomalla *Coordinates*-luokan ilmentymä, kutsumalla *location.getQualifiedCoordinates()*-metodia ja tämän jälkeen tiedot saadaan esiin koordinaatteina. Tämän jälkeen koordinaatit otetaan esiin *coordinates.getLatitude()*- ja *coordinates.getLongitude()*-metodeilla ja asetetaan haluttuun kenttään.

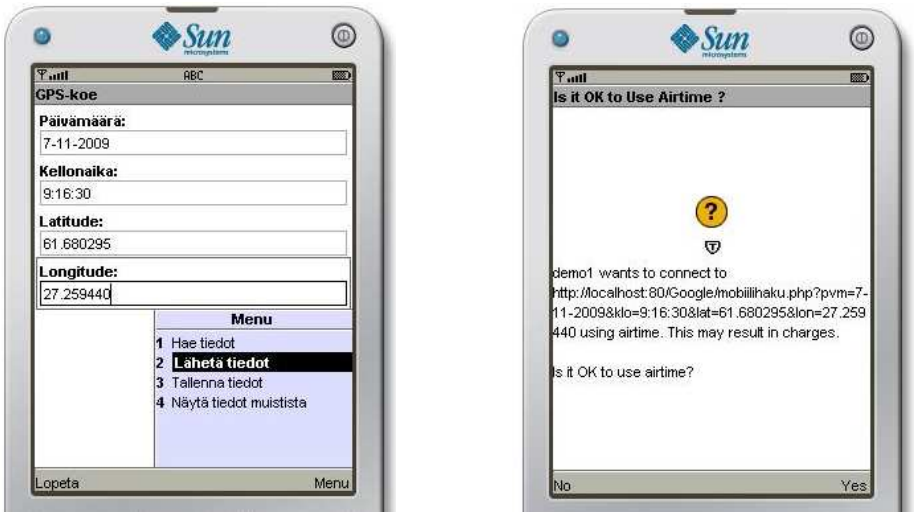
Kun paikkatiedot on saatu, sovellus hakee päivämäärätiedot *Calendar*-luokan ilmentymästä. Se luodaan *Calendar.getInstance()*-metodilla. Tämän jälkeen luodaan *Date*- ja *TimeZone*-luokan ilmentymät ja lisätään ne kalenteriin setTimeZone()- ja setTime()-metodeilla. Viimeisenä sovellus hakee aikatiedot kalenterista ja asettaa ne haluttuun kenttään. Jos kaikki on sujunut onnistuneesti, tulevat haetut tiedot näkyviin käyttäjälle kuvan 18. mukaisesti.



KUVA 18. Haetut tiedot näytöllä

Lähetä tiedot

Jos käyttäjä haluaa lähettää tiedot palvelimelle, hän valitsee Manu-valikon alta **Lähetä tiedot**-toiminnon (kuva 19. vasen näyttö). Tämän jälkeen sovellus kysyy käyttäjältä, saako se muodostaa yhteyden palvelimeen, johon käyttäjän tulee vastata ”kyllä” (kuva 19. oikea näyttö).



KUVA 19. Lähetä tiedot ja saako muodostaa yhteyden

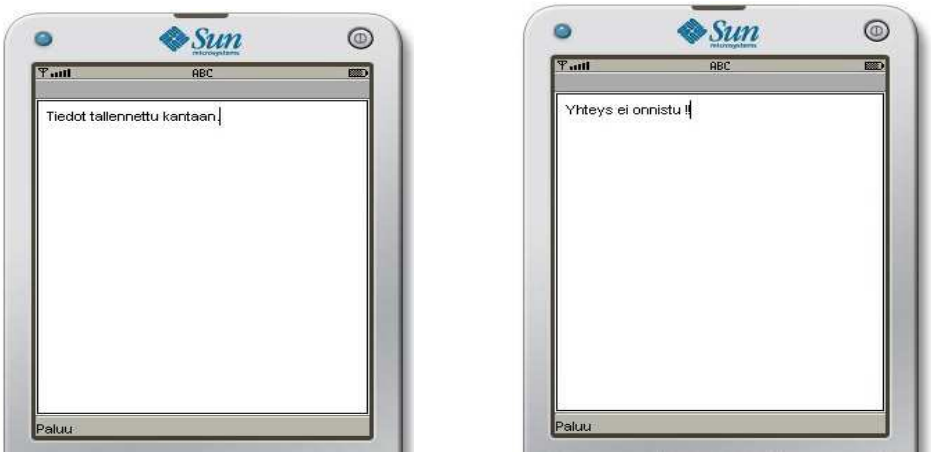
Kun käyttäjä on hyväksynyt yhteyden muodostuksen, sovellus luo ohjelmallisesti uuden säikeen kuvan 20. mukaisesti. Tämän jälkeen ohjelma ottaa lähetettävät tiedot talteen *form*:n kentistä *getString*-metodilla ja liittää ne *URL*-osoitteeseen, joka lähetetään parametrina kutsuttaessa *laheta_tiedot*-aliohjelmaa.

```
public void commandAction(Command c,Displayable s) {
if (c==laheta_tiedot) {
           tb1.setCommandListener(this); // Kuulee "käskyt"
              display.setCurrent(tb1);

           Thread saie = new Thread() {
                public void run() {
                   try {
                    String pvm=kentta1.getString();
                    String klo=kentta2.getString();
                    String lat=kentta3.getString();
                    String lon=kentta4.getString();


                    laheta_tiedot("http://localhost:80/Google/mobiili.php?pvm="+pvm+"&klo="+klo+"&lat="+lat+"&lon="+lon);
                   }catch (Exception e1){
                    tb1.setString("Yhteys tökkii !!");
                   } // catch
             } // run
          };
          saie.start();
 } // if lähetä_tiedot
}
```

KUVA 20. Säikeen muodostus ja tietojen lähetys

Liitteessä kolme on esitelty *laheta_tiedot*-metodin lähdekoodi. Kun aliohjelmaa kutsutaan, se ottaa vastaan parametrina lähetetyn *String*-tyypin merkkijonon. Tämän jälkeen esitellään *StringBuffer*-luokan puskuri, jolla otetaan vastaan palvelinpään lähettämä data, *HttpConnection*-luokan yhteys olio ja *InputStream*-luokan tietovirtaolio. Seuraavaksi ohjelma muodostaa yhteyden *HttpConnection*-rajapintaa hyväksikäyttäen ja aset-

taa käytettäväksi metodiksi GET-metodin. Viimeiseksi avataan tietovirta *openInputStream()*-metodilla. Jos kaikki menee hyvin, luetaan palvelinpään lähettämä vastaus ja asetetaan se *TextBox*iin sekä suljetaan tietovirta (kuva 21. vasen näyttö). Jos palvelussa tapahtuu virhe, siitä ilmoitetaan käyttäjälle kuvan 21. oikean puoleisen näytön mukaisesti.

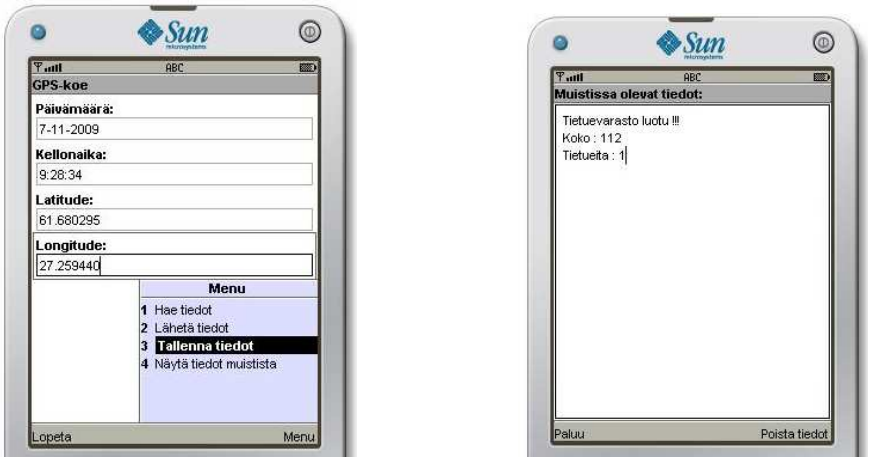


KUVA 21. Tallennus onnistunut ja yhteyden luomisessa tapahtunut virhe.

Testaus-luvussa kerromme, kuinka rakentamamme palvelinpään-ohjelma ottaa tiedot vastaan ja tallentaa ne MySQL-tietokantaan.

Tallenna tiedot

Jos käyttäjä jostain syystä ei halua lähettää tietoja palvelimelle, sovellus tarjoaa mahdollisuuden tallentaa ne puhelimen omaan muistiin (kuva 22. vasen näyttö).



KUVA 22. Tallenna tiedot-valikko ja tiedosto luotu onnistuneesti.

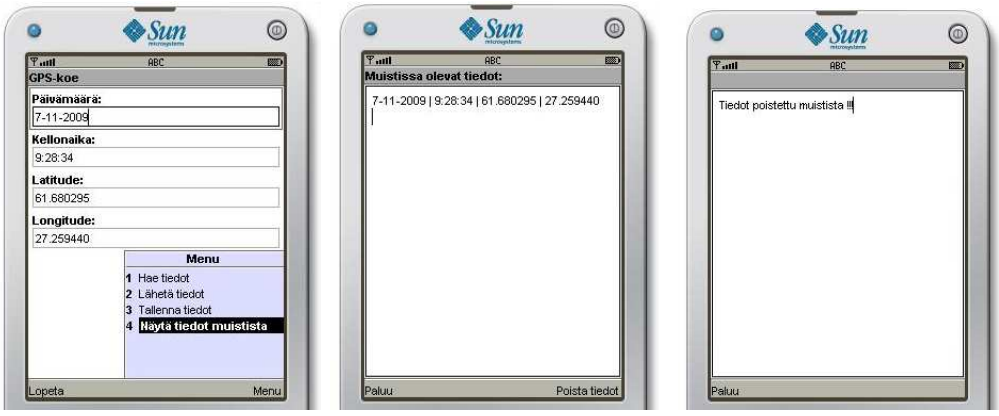
Kun käyttäjä on valinnut **Lähetä tiedot**-komennon, sovellus luo uuden *RecordStore*-luokan tietovaraston *RecordStore.openRecordStore(String, boolean)*-metodilla (kuva 23). *Boolean*-tyyppisen parametrin arvon tulee olla *true*. Tämän jälkeen otetaan halutut tiedot *String*-luokan merkkijonoon ja lisätään |-merkki tietojen väliin. Seuraavaksi merkkijono lisätään tietovarastoon *addRecord()*-metodilla, jossa merkkijono muunnetaan ensin tavuvektoriksi. Tämän jälkeen käyttäjälle ilmoitetaan kuvan 22. oikean puoleisen näytön mukaisesti, että tietuevarasto on luotu. Tietovaraston koko saadaan esille *getSize*-metodilla ja tietueitten määrä *getNumRecords*-metodilla, myös nämä tiedot näytetään käyttäjälle.

```
public void commandAction(Command c,Displayable s) {
if (c==tallenna_tiedot) {
           tb2.setString("");
           display.setCurrent(tb2);

           // Luodaan RMS ja tallennetaan
           RecordStore varasto = null;
           String jono1=kentta1.getString() + " | " + kentta2.getString() + " | " + kentta3.getString() + " | " + kentta4.getString();
           try {
              // Avataan tai luodaan
              varasto = RecordStore.openRecordStore("koe1",true);
              varasto.addRecord(jono1.getBytes(),0,jono1.length());

              tb2.setString("Tietuevarasto luotu !!!\nKoko : "
                    + varasto.getSize() + "\nTietueita : " + varasto.getNumRecords());
           } catch (Exception e) { tb2.setString("Joku mättää !!!");
           } finally {
              try {
                varasto.closeRecordStore();
              } catch (Exception e) {}
           }
  } // if tallenna_tiedot
}
```

KUVA 23. Tallenna tiedot-komennon lähdekoodi

Näytä tiedot muistista

Kun käyttäjä haluaa tarkastella tallentamiaan tietoja, hän valitsee **Näytä tiedot muistista**-komennon (kuva 24. vasen näyttö).



KUVA 24. Näytä tiedot, tallennetut tiedot ja tiedot poistettu muistista

Näytä tiedot muistista-komennon lähdekoodi on esitelty liitteessä 4. Tietovarasto avataan *RecordStore.openRecordStore()*-metodilla ja sen tietueet käydään läpi *enumerateRecords()*-metodin avulla. Ohjelmallisesti luodaan *while*-silmukka, jossa käydään kaikki varaston tietueet läpi ja tulostetaan ne käyttäjälle. Jos tässä tapahtuu virhe, siitä ilmoitetaan käyttäjälle. Kun kaikki on sujunut ongelmitta, tietovarasto suljetaan *closeRecordStore()*-metodilla.

Kuvan 24. keskimmäisen näytön oikeassa alareunassa on näkyvissä, että käyttäjällä on mahdollisuus poistaa tiedot tietovarastosta. Jos käyttäjä valitsee tämän toiminnon, sovellus poistaa tiedot ohjelmallisesti *delete.RecordStore()*-metodin avulla (kuva 25). Onnistuneesta poistosta ilmoitetaan käyttäjälle kuvan 24. oikean puoleisen näytön mukaisesti.

```
public void commandAction(Command c,Displayable s) {
    if (c==poista_tiedot) {
            tb1.setString("");
            display.setCurrent(tb1);

            RecordStore varasto = null;
            try {
               // Tuhotaan
               varasto.deleteRecordStore("koe1");
               tb1.setString("Tiedot poistettu muistista !!!");
            } catch (Exception e) { tb1.setString("Joku mättää !!!");}
        } // if poista_tiedot
}
```

KUVA 25. Tietojen poisto puhelimen muistista

5.3.4 Testaus

Ohjelmaa tehdessämme testasimme sen toimintaa emulaattorilla. Kun olimme saaneet rakennettua testiympäristössä toimivan version ohjelmastamme, siirsimme sen omiin puhelimiimme ja testasimme ohjelman toimivuutta niillä. Sovelluksen siirtoon on käytettävissä useita eri mahdollisuuksia. Itse siirsimme sovelluksen datakaapelia hyväksikäyttäen. Siirtämistä varten tietokoneelle pitää asentaa jokin tiedostonsiirron hallintaohjelma, esimerkiksi Nokian PC Suite. Asennustiedostot, jotka puhelimelle siirretään, ovat sovelluskehittimen kääntämät jad- ja jar-tiedostot. Näistä jad-tiedosto sisältää metatietoa sovelluksesta, kuten ohjelman nimen, jar-paketin sijainnin sekä sovelluksen version. Jar-paketti sisältää sovelluksen ja sen tarvitsemat resurssit. Itse sovelluksen

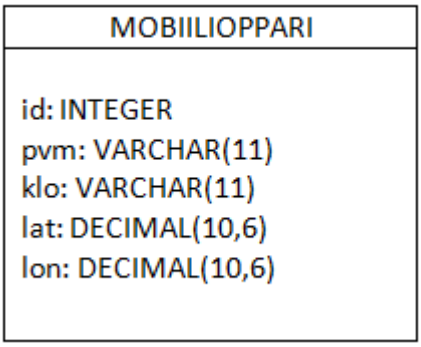
asennus pitää tehdä vielä puhelimen omassa käyttöliittymässä. Asennuksen jälkeen sovelluksen voi käynnistää puhelimen päävalikosta, jonne sen kuvake on ilmestynyt.

Kun olimme saaneet ohjelman onnistuneesti siirrettyä puhelimillemme, keräsimme ja tallensimme ohjelman avulla niiden muistiin paikkatietoja. Tämän jälkeen siirsimme tiedot tietokantaan emulaattorin avulla, josta ne olivat myöhemmin käytettävissämme.

MySQL-tietokanta

Kuten *Lähetä tiedot*-luvun lopussa mainitsimme, meidän piti rakentaa toimiva ympäristö, johon saisimme lähetettyä paikannustiedot myöhempää käyttöä varten. Luontevimmalta valinnalta tuntui rakentaa MySQL-tietokantaa hyväksikäyttävä sovellus, koska se oli meille tuttu aiemmilta ohjelmointikursseilta. Koska tämä ei ollut meidän lopullinen tavoitteemme toimintaympäristöksi päättötyössämme, esittelemme rakennetut ratkaisut lyhyesti.

Liitteessä 5. on esitelty Sql-tietokannan luontiin tarvittava skripti. Kun tämä skripti ajetaan, muodostuu kuvan 26. rakenteen mukainen tietokantataulu. Skriptin lopussa on esimerkkidataa, joka lisätään luotuun tietokantaan.



KUVA 26. Käytetyn tietokannan rakenne

Liitteessä 6. on esitelty rakentamamme php-ohjelman lähdekoodi. Tämä ohjelma ottaa vastaan puhelimelta lähetetyn datan ja lisää sen tietokantaan. Lopuksi ohjelma lähettää puhelimelle tiedon, onko lisäys onnistunut vai ei.

5.3.5 Palvelinpään ohjelmat

Opinnäytetyömme lopulliseen versioon suunnittelimme xml-tiedoston, johon puhelimelta saatava data tallennetaan. Suunnittelun pohjana käytimme jo tekemäämme Sql-tietokannan rakennetta. Tämän jälkeen rakensimme php-ohjelman, joka ottaa vastaan puhelimelta tulevat paikannustiedot ja tallentaa ne xml-tiedostoon. Php-ohjelma toimii muutoin samoin, kuin liitteessä 6. esittelemämme ohjelma, mutta se tallentaa datan xml-tiedostoon. Viimeisenä vaiheena toteutimme ohjelman, joka hakee paikannustiedot xml-tiedostosta ja näyttää ne Google Mapsissa.

Data.xml

Kuvassa 27. on esitelty tekemämme xml-tiedosto, johon saapuva data tallennetaan. Tiedoston rakenne vastaa Sql-tietokannan taulun rakennetta. Data koostuu latitudi- ja longitudi paikannustiedoista, päivämäärästä ja kellonajasta.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<markers>

    <marker>
        <lat>63.083761</lat>
        <lon>25.840770</lon>
        <pvm>12-01-2010</pvm>
        <klo>17:34:31</klo>
    </marker>
</markers>
```

KUVA 27. Xml-tiedoston rakenne

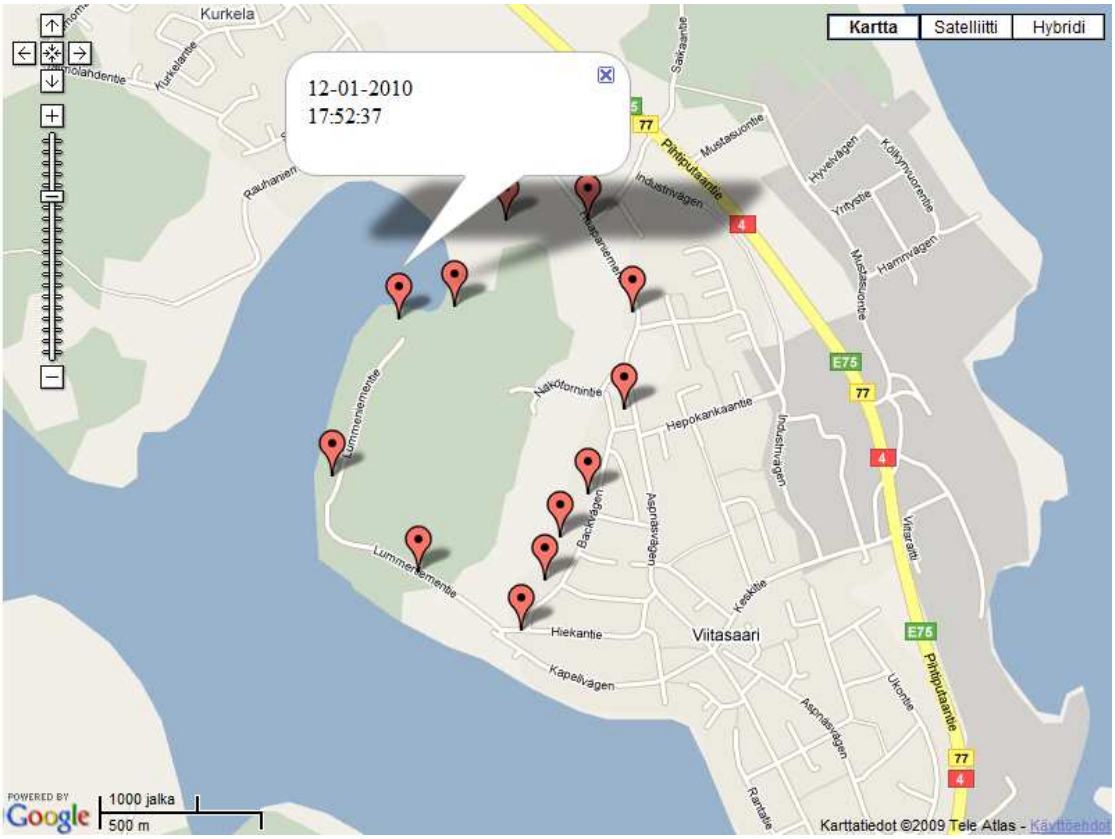
Mobiili_xml.php

Liitteessä 7. on esitelty rakentamamme php-ohjelma, joka ottaa puhelimelta saapuvan datan talteen ja tallentaa ne xml-tiedostoon. Ohjelma on toiminnoiltaan vastaava, kuin liitteessä 6. esittelemämme skripti. Ohjelma ottaa vastaan saapuvan datan get-muotoisena tietona. Tämän jälkeen ohjelma lataa data.xml-tiedoston SimpleXML-luokkaa hyväksikäyttäen. Tämän jälkeen ohjelma luo tarvittavat tiedot ja tallentaa tehdyt muutokset data.xml-tiedostoon.

Kartta_xml.php

Liitteessä 8. on esitelty tekemämme sivusto, jossa näytetään kartalla xml-tiedostossa olevat paikat. Ensimmäiseksi luodaan kartta Google Mapsin tarjoamilla työkaluilla. Tässä vaiheessa määrittelimme, että käyttäjä pystyy GLargeMapControl-ohjaimen avulla panoroimaan kartan näkymää neljään suuntaan ja lukusäätimen avulla zoomaamaan lähemmäs tai kauemmas karttatasolla. GMapTypeControl-ohjain luo painikkeet, joiden avulla voidaan vaihtaa karttatyyppiä katukartan, satelliittikartan ja hybridikartan välillä. GScaleControl luo mittakaavajanan.

Seuraavaksi määritimme nuppineulat GMarker-muodostimella, joka käyttää hyväksi xml-tiedostosta haettua dataa. Ohjelma lataa xml-tiedoston ja se käydään läpi foreach-silmukassa. Kartan keskipiste määräytyy viimeisimmän paikkatiedon mukaan. Nuppineulan sijainti määräytyy tiedostosta löytyvien koordinaattitietojen mukaan. Jokaiseen nuppineulaan on ohjelmoitu click-funktion avulla infoikkuna, mikä avautuu kun nuppineulaa klikataan hiirellä. Infoikkunaan on haettu tiedot minä päivänä ja kellonaikana kyseisen nuppineulan sijainti on otettu.



KUVA 28. Xml-tiedostosta haetut pisteet kartalla

Etaisyys.php

Liitteessä 9. on esitelty skripti, jonka avulla voi hakea tietokannasta kahden pisteen tiedot ja laskea niiden välisen etäisyyden. Skripti, jolla etäisyys voidaan laskea, hyödyntää kappaleessa 2.3 esitettyä Haversinen kaavaa. Vastaus ilmoitetaan kilometreinä kolmen desimaalin tarkkuudella.

6 PÄÄTÄNTÖ

Aloittaessamme rakentamaan mobiilisovellusta meille oli suureksi avuksi Jukka Selinin Mobiili- ja tietoliikenneohjelmointi-kurssilla tehdyt demot. Näiden demojen avulla saimme nopeahkosti tehtyä ensimmäisen toimivan sovelluksen. Ainoa täysin uusi asia oli kalenteri-luokan käyttö ja sen kanssa tulikin ilmi ainut ongelma sovelluksen toimivuudessa. Kalenteri luokasta löytyvä Calendar.Mont on niin sanotusti nolla-pohjainen eli kuukaudet ovat indeksoitu nollasta yhteentoista. Tämän ongelman ratkaisimme yksinkertaisen laskurin avulla kasvattamalla kuukauden arvoa yhdellä kokonaisluvulla.

Seuraavaksi paneuduimme tutkimaan Google Mapsia. Tämä sovellus oli meille molemmille tuttu, mutta kumpikaan ei ollut tutustunut Google Mapsin ohjelmointirajapintaan. Aloittaessamme tekemään omia sovelluksiamme, jotka hyödyntäisivät mobiilisovelluksen hakemia paikkatietoja, saimme paljon hyödyllisiä vinkkejä Antti Marttisen tekemästä opinnäytetyöstä Google Maps ja Google Maps API. Opinnäytetyömme tutkimusongelman ratkaisimme Google Mapsista löytyvillä sähköisillä nuppineuloilla. Nuppineulat näyttävät kartalla kohdan, missä paikkatiedot ovat otettu. Toisen opinnäytetyömme tavoitteemme ratkaisimme rakentamalla xml-pohjaisen tietovaraston paikkatietoja varten, josta php-sovellus hakee ja näyttää ne Internet-selaimella Google Mapsia hyväksikäyttäen yllä kuvatulla tavalla. Mielestämme onnistuimme ratkaisemaan tutkimusongelmamme hyvin ja olemmekin tyytyväisiä työn lopputulokseen.

Vaikka olemmekin tyytyväisiä työhömme, on siinä valtavasti jatkokehitysmahdollisuuksia. Mobiilisovelluksen ulkoasu on hyvin pelkistetty ja sitä voisi kehittää graafisesti näyttävämmäksi. Sovellukseen voisi myös lisätä uusia ominaisuuksia, kuten ku-

van ottamisen, videokuvauksen mahdollistamisen, puheen äänityksen ja näihin paikkatietojen liittämisen sekä näiden tiedostojen palvelimelle lähettämisen.

Selaimella toimivaan sovellukseen voisi lisätä käyttäjätilien luomisen ja tätä kautta henkilökohtaisien blogien ylläpitämisen. Käyttäjä voisi näin näyttää Google Mapsin avulla, missä kerätty data on tallennettu. Esimerkiksi kaupunkilomalla kuvatut nähtävyydet tai kuljetut reitit saisi liitettyä kartalle.

Mobiilisovellukseen voisi myös tehdä ajastimen, johon käyttäjä antaisi tiedon, kuinka tiheästi puhelin ottaisi paikkatiedot ja lähettäisi ne automaattisesti palvelimelle. Palvelinpään ohjelmaan voisi tehdä kuuntelijan Ajax-tekniikka hyväksikäyttäen, mikä päivittäisi kartan aina, kun xml-tiedostossa on tapahtunut muutoksia. Näin sovellukset toimisivat lähes reaaliajassa ilman, että käyttäjän tarvitsisi tehdä mitään lisätoimia.

LÄHTEET

Airos, Esa, Korhonen, Risto & Pulkkinen, Timo 2007. Satelliittipaikannusjärjestelmät. Helsinki: Edita Prima Oy.

Google Maps API 2009 Google. Www-dokumentti. http://www.google.com/apis/maps/documentation/. Luettu 7.12.2009.

Kontio, Mikko 2002. Inside Mobiili Java J2ME. Helsinki: Edita Prima Oy.

Miettinen, Samuli 2006. GPS Käsikirja. Porvoo: WS Bookwell Oy.

JSR-118 MIDP 2.0 2007 Nokia. WWW-dokumentti. http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-4AEC8DAF-DDCC-4A30-B820-23F2BA60EA52/overview-summary.html. Luettu 8.12.2009

JSR-179 Location API 2007 Nokia. WWW-dokumentti. http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-4AEC8DAF-DDCC-4A30-B820-23F2BA60EA52/overview-summary.html. Luettu 9.12.2009

Peltomäki, Juha 2004. J2ME-ohjelmointi. Porvoo: WS Bookwell Oy.

Veness, Chris 2007. Calculate distance, bearing and more between two latitude/longitude points. WWW-dokumentti. http://www.ig.utexas.edu/outreach/googleearth/latlong.html. Päivitetty 10.8.2007. Luettu 15.11.2009.

LIITE 1 (1).
Lähdekoodi

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class NewMain extends MIDlet implements CommandListener{
    private Display display = null;

    // Ensimmäisen näytön vaatimat elementit
    private Form t1 = null;
    private TextField kentta1=null;
    private TextField kentta2=null;
    private TextField kentta3=null;
    private TextField kentta4=null;

    //Tarvittavat komennot
    private Command poistu = null;
    private Command hae_tiedot = null;
    private Command laheta_tiedot = null;
    private Command tallenna_tiedot = null;
    private Command nayta_tiedot=null;


public NewMain() {
        display = Display.getDisplay(this);

        poistu = new Command("Lopeta",Command.EXIT,1);
        hae_tiedot = new Command("Hae tiedot",Command.ITEM,2);
        laheta_tiedot = new Command("Lähetä tiedot",Command.ITEM,2);
        tallenna_tiedot = new Command("Tallenna tiedot",Command.ITEM,2);
        nayta_tiedot = new Command("Näytä tiedot muistista",Command.ITEM,2);

                t1 = new Form("GPS-koe");
        kentta1 = new  TextField("Päivämäärä:","",200,TextField.ANY);
        kentta2 = new  TextField("Kellonaika:","",200,TextField.ANY);
        kentta3 = new  TextField("Latitudi:","",200,TextField.ANY);
        kentta4 = new  TextField("Longitudi:","",200,TextField.ANY);

    }

public void startApp() {
        t1.append(kentta1);
        t1.append(kentta2);
        t1.append(kentta3);
        t1.append(kentta4);
```

```
        t1.addCommand(poistu);
        t1.addCommand(hae_tiedot);
        t1.addCommand(laheta_tiedot);
        t1.addCommand(tallenna_tiedot);
        t1.addCommand(nayta_tiedot);
        t1.setCommandListener(this);
        display.setCurrent(t1);
    }
    public void pauseApp() {}
    public void destroyApp(boolean unconditional) {}
}
```

LIITE 2 (1).
hae_tiedot

```
public void hae_tiedot() {

        // Tutkitaan mikä versio Location API:sta
        String str=System.getProperty("microedition.location.version");
        if (str!=null && !str.equals("")){

        try {
        // Luodaan olio jolla määritellään paikannusehdot
        Criteria criteria = new Criteria();
        // Sallitaan maksullisten paikannuspalveluiden käyttö
        criteria.setCostAllowed(true);
        // Sallitaan korkea virrankulutus
        criteria.setPreferredPowerConsumption(Criteria.POWER_USAGE_HIGH);



        // Luodaan Paikkatietojen tarjoaja (provider) edellämääriteltyjen
        // kriteerien pohjalta
        LocationProvider provider =
                LocationProvider.getInstance(criteria);

        // Haetaan paikkatiedot. Maksimi viive asetetaan
        // 180s = 3 min
        Location location = provider.getLocation(180);

        // Otetaan paikkatiedot olioon josta ne
        // saadaan koordinaatteina
        Coordinates coordinates = location.getQualifiedCoordinates();

        if (coordinates  != null) {
            // haetaan
            double latitude = coordinates.getLatitude();
            double longitude = coordinates.getLongitude();
            kentta3.setString(""+latitude);
            kentta4.setString(""+longitude);
        } else {
            kentta3.setString("Ei koordinaatteja !!!");
            kentta4.setString("Ei koordinaatteja !!!");
        }
      } catch (Exception e1) {
         tb1.setString("Tietoja ei saada !!!");
      } // try
    } else {
```

```
tb1.setString("Puhelimellasi ei voi hakea paikkatietoja !!!");
    }

        Calendar kalenteri = Calendar.getInstance();
        String pvm = ""+kalenteri.get(Calendar.DATE);
        pvm = pvm + "-"+kalenteri.get(Calendar.MONTH);
        pvm = pvm + "-"+kalenteri.get(Calendar.YEAR);

        kentta1.setString(pvm);


        Date d = new Date();
        TimeZone tz = TimeZone.getTimeZone("GMT+02:00");
        kalenteri.setTimeZone(tz);
        kalenteri.setTime(d);
        String klo = kalenteri.get(Calendar.HOUR_OF_DAY) + ":" +
                kalenteri.get(Calendar.MINUTE) + ":" +
                kalenteri.get(Calendar.SECOND);

        kentta2.setString(klo);

  } // hae_tiedot
```

LIITE 3.
laheta_tiedot

```
public void laheta_tiedot(String url) {
      StringBuffer puskuri = new StringBuffer();
      HttpConnection yhteys =null;
      InputStream tietovirta =null;

      try {
         yhteys = (HttpConnection)Connector.open(url);
         yhteys.setRequestMethod(HttpConnection.GET);
         tietovirta = yhteys.openInputStream();

         // Luetaan saapuva data
         int merkki;
         while ((merkki = tietovirta.read()) != -1) {
            puskuri.append((char)merkki);
         } // while

         tb1.setString(puskuri.toString());

         // Suljetaan virrat
         tietovirta.close();

      }catch (Exception e1){
         tb1.setString("Yhteys ei onnistu !!");
      } // catch

   } // laheta_tiedot
```

LIITE 4.
nayta_tiedot

```
public void commandAction(Command c,Displayable s) {
                if (c==nayta_tiedot) {
        tb2.setString("");
        display.setCurrent(tb2);

        // Haetaan tiedot ja tulostetaan
        RecordStore varasto = null;
        RecordEnumeration re = null;
        try {
          // Avataan
          varasto = RecordStore.openRecordStore("koe1",false);
          re = varasto.enumerateRecords(null,null,false);

          while (re.hasNextElement()) {
              String jono = new String(re.nextRecord());
              tb2.setString(tb2.getString() + jono + "\n");
          }
        } catch (Exception e) { tb2.setString("Joku mättää !!!");
        } finally {
           try {
             varasto.closeRecordStore();
           } catch (Exception e) {}
        }

      } // if nayta_tiedot
}
```

LIITE 5.
Sql-lause

```
CREATE DATABASE IF NOT EXISTS mobiilioppari;
USE mobiilioppari;



DROP TABLE IF EXISTS `locations`;
CREATE TABLE `locations` (
`id` int(11) NOT NULL auto_increment,
`pvm` varchar(11) NOT NULL,
`klo` varchar(11) NOT NULL,
`lat` decimal(10,6) NOT NULL default '0.000000',
`lon` decimal(10,6) NOT NULL default '0.000000',
PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;


INSERT INTO `locations` (id,pvm,klo,lat,lon) VALUES (1,'26-10-
2009','17:30:31','61.680295','27.259440');
INSERT INTO `locations` (id,pvm,klo,lat,lon) VALUES (2,'26-10-
2009','17:31:30','61.687674','27.272787');
INSERT INTO `locations` (id,pvm,klo,lat,lon) VALUES (3,'26-10-
2009','17:32:30','61.686209','27.296348');
```

LIITE 6.
Php-skripti

```
<?php
   // Puretaan puhelimen lähettämät tiedot
   $pvm = $_GET["pvm"];
   $klo = $_GET["klo"];
   $lat = $_GET["lat"];
   $lon = $_GET["lon"];

   // Avataan tietokantayhteys
   $yhteys = mysql_connect( "127.0.0.1", "root", "kissakala");
   // Valitaan tietokanta
   mysql_select_db( "mobiilioppari", $yhteys );

   // Suoritetaan SQL-lause
   $sql = "INSERT INTO locations (pvm,klo,lat,lon) VALUES ('$pvm' ,'$klo' ,'$lat'
,'$lon');";
                $tulos=mysql_query($sql,$yhteys);
                //Jos lisäyksessä tapahtuu virhe, näytetään virheilmoitus.
                if (!$tulos)
                //Kyselyn suorittaminen epäonnistui.
                {
                $virhe=mysql_error();
                print($virhe);
                break;
                }
   //Suljetaan tietokantayhteys.
   mysql_close($yhteys);
   // Jos kaikki meni hyvin.
   print("Tiedot tallennettu kantaan!")

?>
```

LIITE 7.
Php-skripti

```
<?php
   // Puretaan puhelimen lähettämät tiedot
   $pvm = $_GET["pvm"];
   $klo = $_GET["klo"];
   $lat = $_GET["lat"];
   $lon = $_GET["lon"];


   $xmlName = "data.xml";
   $xml = simplexml_load_file($xmlName);


                $noteText        = "<lat>" . $lat . "</lat><lon>" . $lon .
"</lon>"."<pvm>" . $pvm . "</pvm><klo>" . $klo . "</klo>";
                $xmlNoteNode      = $xml->addChild("marker");
                $xmllatNode      = $xmlNoteNode->addChild("lat",$lat);
                $xmllonNode      = $xmlNoteNode->addChild("lon",$lon);
                $xmlpvmNode      = $xmlNoteNode->addChild("pvm",$pvm);
                $xmlkloNode      = $xmlNoteNode->addChild("klo",$klo);

                $xml->asXML("data.xml");


   print("Tiedot tallennettu kantaan!!");


?>
```

LIITE 8 (1).
Php-skripti

```
<head>
<title>Kartta</title>
<script
src="http://maps.google.com/maps?file=api&v=2&key=ABQIAAAAOYbP_3IzjVrY
UMAFw_YLphT2yXp_ZAY8_ufC3CFXhHIE1NvwkxQet819URywN4qbryF3H49i
KalnwQ" type="text/javascript"></script>
</head>
<body>
<p><strong>Pisteet haettu xml-tiedostosta</strong></p>
<div id="map" style="width: 800px; height: 600px"></div>
<script type="text/javascript">
//<![CDATA[

var map = new GMap2(document.getElementById("map"));
map.addControl(new GLargeMapControl());
map.addControl(new GMapTypeControl());
map.addControl(new GScaleControl());



// Luo nuppineulan jonka infoikkuna näyttää annetun numeron
function createMarker(point, number)
{
var marker = new GMarker(point);
// Näyttää tämän nuppineulan infoikkunan sisällön hiirtä klikattaessa
var html = number;
GEvent.addListener(marker, "click", function()
{marker.openInfoWindowHtml(html);});
return marker;
};



<?php
    $dom = new DomDocument("1.0");
    $dom->load("data.xml");


    $merkit = $dom->getElementsByTagName("marker");

    foreach($merkit as $merkki) {
        $lat = $merkki->GetElementsByTagName("lat")->item(0)->textContent;
        $lon = $merkki->GetElementsByTagName("lon")->item(0)->textContent;
        $pvm = $merkki->GetElementsByTagName("pvm")->item(0)->textContent;
```

LIITE 8 (2).
Php-skripti

```
        $klo = $merkki->GetElementsByTagName("klo")->item(0)->textContent;

                echo "map.setCenter(new GLatLng(". $lat . "," . $lon . "), 14,
G_NORMAL_MAP);";
                echo "var point = new GLatLng(" . $lat . "," . $lon . ");\n";
                                        echo "var marker = createMarker(point, '"
.$pvm ."<br>" .$klo . "');\n";
                                        echo "map.addOverlay(marker);\n";
                                        echo "\n";
}




?>
//]]>
</script>
</body>
</html>
```

LIITE 9 (1).
Php-skripti

```
<html>
<head>
<title>Etäisyyden laskeminen</title>
</head>
<body>
<p><strong>Kokeillaan laskea etäisyys...</strong></p>
<div id="etaisyys" style="width: 200px; height: 50px"></div>

<?php
$link = mysql_connect("localhost", "root") or die(" Ei pysty yhdistämään: " .
mysql_error());
mysql_selectdb("mobiilioppari",$link) or die ("Ei pysty käyttämään tietokantaa : " .
mysql_error());
$result = mysql_query("SELECT * FROM locations",$link);
if (!$result)
{
echo "Eipä löytyny mittään.... ";
}
$lat2 = mysql_result($result,1,'lat');
$lon2 = mysql_result($result,1,'lon');
$lat1 = mysql_result($result,0,'lat');
$lon1 = mysql_result($result,0,'lon');


$pi80 = M_PI / 180;
$lat1 *= $pi80;
$lon1 *= $pi80;
$lat2 *= $pi80;
$lon2 *= $pi80;
$r = 6371; // mean radius of Earth in km
$dlat = $lat2 - $lat1;
$dlng = $lon2 - $lon1;


$a = sin($dlat / 2) * sin($dlat / 2) + cos($lat1) * cos($lat2) * sin($dlng / 2) * sin($dlng
/ 2);
$c = 2 * atan2(sqrt($a), sqrt(1 - $a));
$km = $r * $c ;
echo round($km, 3)." kilometriä";
mysql_close($link);
?>
</body>
</html>
```