Tuomas Korjonen

MOBIILIOHJELMISTON TOTEUTTAMINEN PROJEKTINHALLINTAJÄRJESTELMÄÄN

Tietotekniikan koulutusohjelma
Tekniikka Rauma
2009

MOBIILIOHJELMISTON TOTEUTTAMINEN
PROJEKTINHALLINTAJÄRJESTELMÄÄN

Korjonen, Tuomas
Satakunnan ammattikorkeakoulu
Tekniikka Rauma
Tietotekniikan koulutusohjelma

Toukokuu 2009
Yritys: Bitec Oy
Valvoja: Janne Raitaniemi
Ohjaaja: Mikko Javanainen
UDK: 004.41
Sivumäärä: 54

Asiasanat: ohjelmistokehitys, sulautettu tietotekniikka, testaus, projektinhallinta, Java

---

Tämän opinnäytetyön aiheena oli toteuttaa projektinhallintajärjestelmään matkapuhelinohjelmisto. Tarkoituksena oli, että järjestelmään lisättäisiin etäkäyttömahdollisuus. Ohjelmiston rakenne määriteltiin niin, että se on helppo räätälöidä erikoisempiinkin tarpeisiin asiakkaan haluamalla tavalla.

Ohjelmiston tarve ilmeni, kun Bitec Oy alkoi toteuttaa projektinhallintajärjestelmää, jota käytettäisiin Internet-selaimen välityksellä. Asiakasyritysten työntekijöille piti myös luoda mahdollisuus etäkäyttöön, jossa voidaan tehdä kirjauksia järjestelmään.

Ohjelmiston toteuttamiseen käytettiin Javan mobiilialustaa Java ME:tä. Tämä tekniikka valittiin, koska Java ME on tuettu lähes kaikissa markkinoilla olevissa matkapuhelimissa. Tiedon siirto matkapuhelimesta järjestelmään toteutettiin lähettämällä XML-dokumentteja http-protokollan yli. Ohjelmiston arkkitehtuurina käytettiin MVC-arkkitehtuuria, jolloin ohjelmiston rakenne saatiin modulaariseksi.

Ohjelmisto toteutettiin käyttämällä testivetoista kehitysmenetelmää. Tässä menetelmässä luodaan testi ennen toteutusta. Menetelmän avulla jokainen ohjelmiston osa voitiin testata välittömästi.

Lopputuloksena syntyi matkapuhelimeen ohjelmisto, jota voitiin käyttää sellaisenaan tai se voitiin tarpeen mukaan räätälöidä eri tilanteisiin sopivaksi.

IMPLEMENTING MOBILE APPLICATION TO PROJECT MANAGEMENT SYSTEM

Korjonen, Tuomas
Satakunta University of Applied Sciences
Technology and Maritime Management Rauma
Information Technology
May 2009
Commissioned by Bitec Oy
Supervisor: Janne Raitaniemi
Tutor: Mikko Javanainen
UDC: 004.41
Number of pages: 54

Keywords: software development, testing, embedded information technology, project management, Java

---

The purpose of this thesis was to create a mobile phone application for a project management system. The purpose was to give the users possibility to remotely send and receive information from the system with their mobile phones. The Structural requirement was that the application should be easily modified and extended.

The need for this application came up when Bitec Oy started to implement a project management system. This system was designed to be used through a web-browser and there was a need for the possibility to use it with a mobile phone.

The technology used in this project was the Java Micro Edition. This was chosen because it is today supported in nearly every mobile phone in the market. Data transfer was implemented by sending XML formatted data over HTTP protocol. MVC-architecture was used to achieve modular structure in the application.

The application was created using test driven development. In this development method, the test is written before the implementation. By using this development method, every part of the application was tested immediately.

The result was a working application, which could be used as such or be easily modified if needed.

SISÄLLYS

LYHENTEET JA TERMIT

CLDC	Connected Limited Device Configuration
DOM	Document object model
Emulaattori	Laitteiden simulointiin tarkoitettu ohjelma
GPL	General Public License
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol secure
JDK	Java development kit
Java ME	Java Micro Edition
JMUnit	JME-yksikkötestausohjelmisto
KVM	Kilo virtual machine
MIDP	Mobile information device profile
MVC	Model-View-Controller arkkitehtuuri
MVP	Model-View-Presenter arkkitehtuuri
S40	Nokian peruspuhelimien käyttöjärjestelmä
S60	Nokian älypuhelimien käyttöjärjestelmä
SAMK	Satakunnan ammattikorkeakoulu
SAX	Simple Api for XML
SDK	Software development kit
SVG	Scalable Vector Graphics
SGML	Standard Generalized Markup Language
Tagi	XML-elementti
WLAN	Wireless local area network
WTK	Wireless toolkit
XML	Extensible markup language
XML-parserointi	XML dokumentin tulkitseminen
XMLPull	XML dokumentin tulkitsemismenetelmä

1 JOHDANTO

Bitec Oy tekee omaa projektinhallintajärjestelmää. Tähän järjestelmään on tarkoitus tehdä mobiiliohjelmisto, jonka avulla voidaan kirjata haluttuja tietoja järjestelmään. Ohjelmiston toimivuus osoitetaan jäljittelemällä erilaisia käyttötilanteita. Tämä testi toteutetaan yrityksen tiloissa yrityksen palvelimen avulla.

Insinöörityö aloitettiin kartoittamalla mahdolliset toteutustavat ja teknologiat. Toteutustavaksi valittiin Sun Microsystemsin mobiili alusta, nimeltään Java ME, koska Java ME on tuettu lähes kaikissa markkinoilla olevissa matkapuhelimissa. Java ME:n avulla sama sovellus toimii lähes jokaisessa matkapuhelimessa, merkistä ja mallista riippumatta. Ensisijainen laitetuki suunniteltiin kuitenkin Nokian matkapuhelimille niiden yleisyyden vuoksi. Ohjelmiston rakenne määriteltiin niin, että se olisi mahdollisimman modulaarinen eli siihen olisi helppo lisätä tai poistaa ominaisuuksia.

Ohjelmiston laatu varmistettiin käyttämällä ohjelmallista testausta, jolloin ohjelmiston jokainen komponentti voitiin testata erikseen ja näin varmistua myös siitä, että mikään komponentti ei ollut liian riippuvainen toisista komponenteista. Ohjelmallisella testauksella varmistuttiin myös siitä, että uusien toimintojen lisäys ohjelmistoon ei rikkonut jo olemassa olevien komponenttien toiminnallisuutta. Lopullisessa ohjelmistossa oli jokaista komponenttia kohden ainakin yksi testi, jolloin voitiin varmistua, että koko ohjelmisto toimi halutulla tavalla.

Insinöörityö toteutettiin yksilötyönä. Tuomas Korjonen vastasi ohjelmiston toteutuksesta, testauksesta ja suunnittelusta. Bitec Oy määritteli ohjelmiston vaatimukset ja rajapinnat palvelimeen.

2 JAVA

2.1 Mikä Java on?

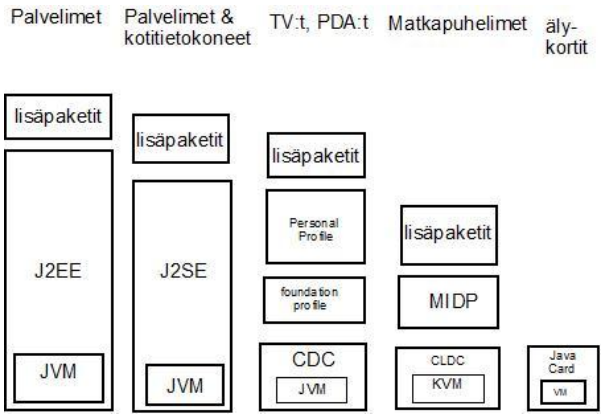
Java on ohjelmointikieli, jonka kehittivät Bill Joy ja James Gosling Sun Microsystems:llä 1990-luvun alussa. Ensimmäinen kehitystyökalu, nimeltään JDK 1.0, julkaistiin vuonna 1995. Java nousi suureen suosioon, koska sillä voitiin tehdä www-sivuille selaimessa ajettavia ohjelmia, niin kutsuttuja appletteja. Javan merkittäviä etuja oli myös se, että se muistutti, sen aikaisen suosituimman ohjelmointikielen C++:n syntaksia, jolloin ohjelmoijien oli helppo oppia se. Etuna oli sekin, että Java oli laitteistoriippumaton, mikä tarkoitti sitä, että java-ohjelmia voitiin ajaa erilaisissa käyttöjärjestelmissä ilman koodimuutoksia. Myös muistinhallintaa oli helpotettu siten, että ohjelmoijan ei tarvinnut itse vapauttaa muistia, kun sitä ei enää tarvittu, vaan Javan virtuaalikone pystyi automaattisesti siivoamaan muistista käyttämättömät tiedot.

Toisin kuin muut ohjelmointikielet, Javalla kirjoitettu koodi käännetään tavukoodiksi, joka suoritetaan Java-virtuaalikoneessa. Virtuaalikone on tavallaan yksi välikerros varsinaisen järjestelmän ja Java-ohjelman välissä. Virtuaalikoneella lisätään myös turvallisuutta, sillä kaikki käskyt järjestelmälle menevät virtuaalikoneen kautta, joka voi estää haitallisten toimenpiteiden suorittamisen.

Rakenteeltaan Java muistuttaa C++ -ohjelmointikieltä. Se on vahvasti tyypitetty, eli jokainen muuttuja on jotain tyyppiä, eikä se voi saada kuin sille tyypille ominaisia arvoja. Java on oliopohjainen, eli ohjelma koostuu erinäisistä olioista, joista kukin kuvaa jotain asiaa tai toimintoa. Ohjelma koostuu näiden olioiden yhteistoiminnasta. Näiden perusrakenteiden lisäksi Javassa on mukana runsas kirjasto, joka sisältää valmiina mm. käyttöliittymäkirjaston, säikeistyksen ja verkko-ominaisuudet. Nykyään Java on yksi yleisimmistä ohjelmointikielistä, ja sitä käytetään usein ohjelmoinnin opetuksessa. (Cadenhead & Lemay, 2007, 10-14.)

2.2 Java Micro Edition

Java Micro Edition, lyhyesti Java ME, on Java-sovellusympäristöstä karsittu versio, joka on tarkoitettu ominaisuuksiltaan rajoitettuihin laitteistoihin, kuten matkapuhelimiin, televisioihin ja kämmenmikroihin. Suurimmat erot tavalliseen Java-ympäristöön ovat huomattavasti rajoittuneempi luokkakirjasto ja Java-ohjelmien suoritustapa. JME:n virtuaalikone on nimetty KVM:ksi (Kilo Virtual Machine). Se on erittäin pieni, alle 100 kilotavun kokoinen, minkä vuoksi Sun Microsystems antoi virtuaalikoneen etuliitteeksi K-kirjaimen. JME jaetaan konfiguraatioihin, profiileihin ja valinnaisiin rajapintoihin. Konfiguraatiolla tarkoitetaan, että laite täyttää tietyt muisti- ja prosessorivaatimukset. Se määrittelee Java-virtuaalikoneen, joka on helppo muuntaa toimimaan vaatimukset täyttävässä laitteessa. Laitevalmistajat ovat vastuussa siitä, että tietty konfiguraatio saadaan laitteella toimimaan. Profiilit ovat yksityiskohtaisempia kuin konfiguraatiot. Ne perustuvat konfiguraatioihin, mutta määrittelevät lisäksi muita rajapintoja, kuten käyttöliittymän luonti ja multimedian käsittely. Valinnaiset rajapinnat sisältävät rajapintoja, joita laitevalmistaja voi halutessaan tukea. Näitä voivat esimerkiksi olla SVG-kuvien käsittely ja Bluetooth-yhteyden hallinta. Kuvassa 1. ovat kaikki eri Java-sovellusympäristöt. (Li & Knudsen 2005, 1.)



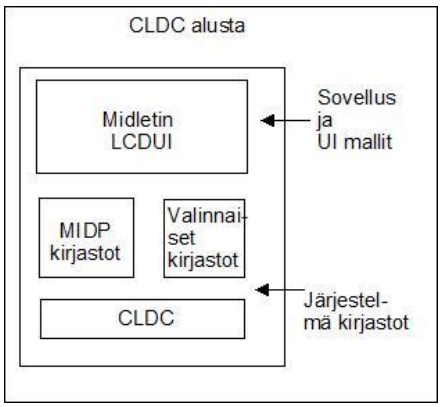
Kuva 1. Java-ympäristöjen rakenne.

2.2.1 MIDP-profiili

MIDP-profiili eli Mobile Information Device Profile, joka on tarkoitettu matkapuhelimille ja kämmenmikroille. MIDP-profiili tuo mukanaan kirjastoja mm. Käyttöliittymän ja grafiikan luontiin sekä multimedian käsittelyyn. MIDP-profiilin vaatimuksiin kuuluu, että laitteella on 256 kilotavua muistia MIDP:n toteutukselle, vähintään kahdeksan kilotavua häviämätöntä muistia, näytön koko vähintään 96 x 54 pikseliä, jokin syöttömahdollisuus, kuten näppäimistö tai kosketusnäyttö, ja kaksisuuntainen verkkoliikenne. MIDP-sovelluksia kutsutaan yleisesti Midleteiksi. (Li & Knudsen, 2005, 5.)

2.2.2 CLDC-konfiguraatio

CLDC:n vaatimuksiin kuuluu, että laitteessa on 16-bittinen prosessori ja 160 kilotavua muistia saatavilla Javan virtuaalikoneelle. CLDC:stä ovat nykyään versiot 1.1 ja 1.0. Suurin ero näiden versioiden välillä on siinä, että versio 1.1 tukee liukulukuaritmetiikkaa ja vaatii muistia 192 kilotavua. CLDC tuo muutamia peruskirjastoja ohjelmoijan käytettäväksi, kuten perus-input/output-toiminnot ja säikeistykseen vaadittavia luokkia. Kaikissa muutaman viime vuoden aikana valmistetuissa puhelimissa on CLDC 1.1 -versio käytettävissä. CLDC:n rakenne on kuvattu kuvassa 2. (Li & Knudsen, 2005, 3-4.)



Kuva 2. CLDC-alusta.

3 ARKKITEHTUURI

3.1 Arkkitehtuuri yleisesti

Arkkitehtuurilla tarkoitetaan ohjelmistokehityksessä sitä, että ohjelmisto rakennetaan tietyllä tavalla, joka on koko ohjelmistossa yhtenäinen. Hyvällä arkkitehtuurilla ohjelmistoa on helppo ylläpitää, laajentaa ja muokata tarpeen mukaan. Ohjelmistoarkkitehtuuri kuvaa sitä, miten ohjelmiston eri komponentit toimivat keskenään ja millainen suhde niillä on toisiinsa. Ohjelmistoarkkitehtuuri siis kuvaa abstraktilla tasolla, miten ohjelmisto on rakennettu ja mitä yhteyksiä eri ohjelmiston osilla on toisiinsa. Se ei kuvaa minkään osa-alueen toteutusta, mutta se kuvaa, miten mikäkin osa käyttäytyy. (Bass, Clements & Kazman, 2003, 19-22.)

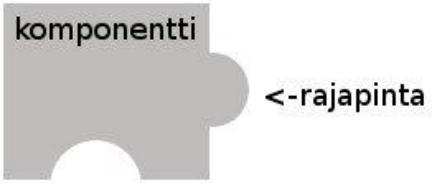
3.2 Modulaarinen rakenne

Modulaarinen rakenne tarkoittaa sitä, että ohjelmiston jokainen komponentti on mahdollisimman riippumaton toisistaan ja komponentit keskustelevat yhteisten sovittujen rajapintojen kautta. Mikään komponentti ei siis suoraan tunnista, millainen jokin toinen komponentti on. Komponentti tietää vain rajapinnan, jolla se saa toiselta komponentilta tietoa. Komponentti voi kuitenkin olla riippuvainen jostain rajapinnasta, muttei koskaan suoraan sen toteuttavasta komponentista.

Modulaarisen rakenteen edut ovat selvät. Ohjelmiston laajentaminen tai muokkaaminen tapahtuu komponentteja lisäämällä tai vaihtamalla. Tämä nopeuttaa erilaisten versioiden muodostamista ohjelmistosta. Komponentti voidaan yksinkertaisesti poistaa, jos jotain tiettyä ominaisuutta ei tarvita. Komponentteja voidaan helposti vaihtaa, kunhan huolehditaan siitä, että uusi komponentti toteuttaa saman rajapinnan.

Voidaan ajatella, että yksi komponentti on yksi palapelin pala. Palasta tiedetään, mitä se kuvaa ja minkä muotoinen se on. Kuvaus on se, mitä komponentti tekee ja muoto se, mikä on komponentin rajapinta. Palapeli kuvaa koko ohjelmistoa. Palan voi vaih-

taa, kunhan uusi pala sopii vanhan tilalle, eli komponentti toteuttaa saman rajapinnan. Perusidea on esitelty kuvassa 3.



Kuva 3. Komponentti palapelin palana kuvattuna.

3.3 MVC-arkkitehtuuri

MVC-arkkitehtuuri tulee sanoista Model-View-Controller eli malli-näkymä-ohjain. Sitä käytetään erityisesti sovelluksissa, joissa on käyttöliittymä. MVC:n ideana on, että sovellus jaetaan kolmelle eri tasolle: malliin, näkymään ja ohjaimeen. Malli on vastuussa sovelluksen tiedon tallentamisesta, järjestämisestä ja ylläpidosta. Näkymä on vastuussa tiedon esittämisestä, jonka se saa epäsuorasti mallilta. Yhtä mallia kohden voi olla useita näkymiä. Ohjain toimii niin sanotusti liimana mallin ja näkymän välillä. Ohjain toteuttaa ohjelmalogiikan ja tekee toiminnot käyttäjän antamien käskyjen mukaan. (Gamma, Helm, Johnson & Vlissides 1994, 14-16.)

MVC:n historia juontuu 1970-luvulle smalltalk-ohjelmointikieleen, jolla sitä ensin sovellettiin. MVC:stä on nykyään monia eri versioita ja toteutuksia riippuen siitä, millaisia sovelluksia ollaan tekemässä. Uudemmista versioista tunnetuin on Microsoftin kehittämä MVP (Model-View-Presenter), joka on erittäin käytetty arkkitehtuuri Microsoft-teknologioiden parissa. (Fowler, M. 2006.)

3.3.1 Edut

MVC:n etuna voidaan pitää sitä, ettei mikään osa-alue ole suuresti riippuvainen mistään toisesta osa-alueesta. Malli ei edes tiedä näkymästä tai ohjaimesta. Ohjain ei suoraan tiedä, tai tietää erittäin vähän, näkymästä tai mallista muuta kuin rajapintojen kautta. Tämä toimii hyvin silloin, kun sovelluksessa pyritään muutenkin käyttämään

modulaarista rakennetta. Sovelluksen komponenttien testaus on helppoa, sillä ne voidaan testata toisistaan erillään, koska komponentti ei ole suoraan riippuvainen mistään.

3.4 Ohjelmointi rajapintoihin

Ohjelmointi rajapintoihin on eräs olio-ohjelmoinnin perusohje, jota ei kuitenkaan usein käytetä. Käsite tarkoittaa sitä, että kun suunnitellaan ohjelmistoa, suunnitellaan tarvittavat rajapinnat, ei vain olioita, jotka sitten toteutetaan. Sopimalla rajapinnat ja toteuttamalla ne päästään siihen, ettei mikään luokka suoraan tiedä toisesta luokasta mitään muuta kuin rajapinnan, samalla tavalla kuin modulaarisessa rakenteessa. Ohjelmoimalla rajapintoihin edesauttaa sitä, että sovellusta pystytään nopeasti muuttamaan tarvittaessa eli sovelluksen rakenne on joustava. (Gamma yms 1994, 30.)

Rajapinnat estävät myös niin sanotun heikon yläluokan ongelman. Olio-ohjelmoinnissa on mahdollista periyttää toisen luokan ominaisuudet aliluokalle. Voidaan esimerkiksi olemassa olevan luokan toiminnallisuus periyttää toiselle luokalle ja tälle toiselle luokalle tehdä vielä omia ominaisuuksia. Heikon yläluokan ongelma tarkoittaa sitä, että yläluokan huonosti toteutetut ominaisuudet siirtyvät periytymisketjussa myös aliluokkiin, joihin nämä virheetkin sitten periytyvät. Toteuttamalla rajapinnan ei ole sellaista vaaraa, että yläluokassa on virhe, sillä rajapintojen toteutus on ohjelmoijan vastuulla. (Holub 2003.)

3.5 Suunnittelumallit

Suunnittelumalli on kuvaus ongelmasta, joka toistuu jossain tietyssä tilanteessa usein. Se kuvaa myös ratkaisua, jota voidaan käyttää samankaltaisessa tilanteessa uudestaan. Suunnittelumallit on tarkoitettu ratkaisemaan ongelmia, joita olio-ohjelmoinnissa tulee vastaan päivittäin. Suunnittelumalleja on todella paljon, joten seuraavaksi on esitelty vain ne, joita tässä projektissa on käytetty. (Gamma yms 1994, 12–13.)

3.5.1 Singleton-malli

Singleton-malli on ratkaisu tilanteeseen, jossa luokasta voi olla olemassa vain yksi olio ja siihen on yksi globaali yhteys. Tässä mallissa luokka on itse vastuussa siitä, että luokasta on olemassa vain yksi instanssi, ja luokka tarjoaa tähän instanssiin yhden yhteystavan, jolla mikä tahansa luokka voi kutsua tätä yhtä ja samaa instanssia. Singleton-mallin etuna on se, että tähän yhteen instanssiin on kontrolloitu yhteys. Singleton-luokka voidaan muuttaa sallimaan useamman kuin yhden olion luomisen. Singleton-luokasta voidaan periyttää aliluokkia, jolloin sen toimintaa voidaan vaihdella palauttamalla instanssi aliluokasta. (Gamma yms 1994, 144–152.)

3.5.2 Factory-malli

Factory-malli ratkaisee tilanteen, jossa luokka ei tiedä, millaisen olion sen täytyy luoda. Esimerkiksi tilanteessa, jossa tietokantayhteys voi vaihdella ja yhteys täytyy luoda tilanteen mukaan, voidaan delegoida tämä yhteyden luonti Factory-luokalle. Factory-luokka saa vain tietokannan tyypin ja palauttaa sen perusteella oikeanlaisen yhteyden. Factory-malli eliminoi konkreettisten luokkien toisiinsa sitomisen. Esimerkissä, jossa luokka pyytää erilaisia tietokantayhteyksiä, luokka ei tiedä yhteyksistä muuta kuin sovitun rajapinnan. (Holzner 2006, 50-61.)

3.5.3 Observer-malli

Observer-mallissa luodaan yleensä yhdestämoneen-riippuvuus luokkien välille siten, että yhden luokan tila muuttuessa muut luokat saavat siitä ilmoituksen. Malli koostuu kohteesta ja tarkkailijasta. Kun kohteen tila muuttuu, tarkkailija saa siitä ilmoituksen. Tarkkailijoita voi olla useita. Observer-malli on erittäin käyttökelpoinen, kun kohde ei halua tietää sen tarkkailijoista erityisemmin mitään vaan ainoastaan tietää rajapinnan, jonka kautta se voi tarkkailijalle ilmoittaa tilan muutoksesta. Observer-mallilla vältytään siltä, että yksi luokka tietäisi kaikista luokista enemmän kuin oikeastaan on tarpeellista. JME-kirjastossa on erinomainen esimerkki Observer-mallista. Siellä Form-luokalle voidaan asettaa CommandListener-rajapinnan toteuttava tarkkailija, jolle ilmoitetaan aina kun jokin Form-luokan komennoista on tapahtunut. (Gamma yms 1994, 326-330.)

4 XML-KIELI

4.1 XML yleisesti

XML-merkintäkieltä käytetään kuvaamaan ja järjestämään tietoa. XML on lyhenne sanoista ”Extensible Markup Language”. XML pohjautuu samantyyppiseen SGML-kieleen. SGML kieli kehitettiin, jotta tiedon välitys eri sovellusten välillä olisi helppoa. SGML on tekstimuotoinen kieli, jolla voidaan merkitä tietoa. SGML suunniteltiin standarditavaksi minkä tahansa tiedon merkintään. Eräs tunnetuimmista SGML-kieleen pohjautuvista kielistä on HTML eli ”Hypertext Markup Language”. HTML-kieltä käytetään kuvaamaan dokumentteja, joita voidaan katsella Internet-selaimella. Nämä dokumentit ovat siis Internet-sivuja, joita jokainen ihminen katselee päivittäin. HTML:n ideana oli, että mikä tahansa ohjelma eli Internet-selain, joka ymmärsi HTML-kieltä, pystyi näyttämään dokumentin.

XML-kieli luotiin, sillä SGML-kieli oli liian monimutkainen ja HTML-kieli oli käyttökelpoinen vain rajatulla alueella eli näyttämään dokumentteja Internet-selaimissa. XML-kielen tavoite on sama kuin SGML-kielessä, kuvata mitä tahansa tietoa, mutta yksinkertaisesti. XML ei oikeastaan ole kieli, vaan standardi, jolla voi luoda kieliä, jotka täyttävät XML:n kriteerit. XML on siis syntaksi, jolla luodaan omia kieliä. Esimerkiksi, jos henkilöiden nimiä halutaan jakaa muiden ohjelmien kanssa, voidaan luoda esimerkin mukainen XML-tiedosto sen sijaan, että kirjoitettaisiin vain lista nimiä.

Esimerkki:

```
<nimi>
<etu>Tuomas</etu>
<suku>Korjonen</suku>
</nimi>
```

Esimerkistä näemme, että XML on itsensä kuvaava kieli. Esimerkistä näkee helposti, että tämä tieto esittää nimiä. XML:ssä tagit eli esim. ”<nimi>” voidaan keksiä täysin itse. Tavalliseen nimilistaan verrattuna XML-tiedosto on selkeästi suurempi, mutta XML:ää ei suunniteltu tekemään pieniä tiedostoja vaan helpottamaan tiedon käsittelyä sovelluksissa. XML-kielen etu on se, että vaikka tieto olisi miten monimutkaista tahansa, pystytään se helposti tulkitsemaan missä tahansa sovelluksessa, joka ymmärtää XML:ää. (Hunter yms 2007, 4-12.)

4.2 XML-kielen tulkitseminen

XML-kielen tulkitseminen eli parserointi on helppoa, sillä sitä varten on tehty valmiita kirjastoja ja sovelluksia. Oman sovelluksen ei koskaan tarvitse nähdä XML-tiedostoa sellaisenaan, vaan se voidaan välittömästi antaa parseroitavaksi, jolloin XML-kielen tulkitseminen tehdään ohjelmoijan puolesta. Parserit eli sovellukset, jotka tulkitsevat XML-tiedoston, toimivat niin, että ne kertovat mitä tietoa XML-tagien välissä on. Parserit eivät ota kantaa siihen, mitä data on, ne vain palauttavat sen. Edellisen esimerkkitiedoston tulkitsemisessa parseri lukisi tiedostoa ja ilmoittaisi että ”<etu>”-tagin välissä oleva tieto on ”Tuomas”. XML:n parserointiin on kehitetty kolme eri tapaa: DOM, SAX ja vähemmän tunnettu XMLPull.

4.2.1 DOM-parserointi

DOM eli “Document Object Model” on XML-tiedoston parserointitapa, jossa tiedostosta rakennetaan puumainen kuvaus muistiin. Tämä puurakenne koostuu XML-elementeistä. Puun ylimpänä elementtinä on dokumentti-elementti. Tällä elementillä voi olla yksi tai useampi lapsielementti, joilla voi myös olla lapsielementtejä. Elementeillä voi olla myös tekstiosia. Nimilista-esimerkistä koostettu puurakenne olisi sellainen, että ylimpänä olisi dokumentti-elementti ja sillä olisi useita nimi-elementtejä lapsielementtinä. Näillä lapsielementeillä olisi taas lapsielementtinä etu-elementti ja suku-elementti. Etu- ja suku-elementtien tekstiosa sisältäisi varsinaisen nimen. DOM-parserointi on käytännöllinen silloin kun tiedetään, että XML-tiedosto

on pieni, sillä DOM-parseroinnissa koko XML-tiedosto otetaan muistiin käsittelyn ajaksi. Suurilla XML-tiedostoilla parserointi olisi tällöin erittäin tehotonta.
(Hunter ym. 2007, 444-449.)

4.2.2 SAX-parserointi

SAX eli “Simple API for XML” kehitettiin korjaamaan DOM-tyyppisen parseroinnin ongelmat ja standardoimaan XML-tiedostojen parserointitapa. SAX on tapahtuma-pohjainen parserointitapa. Kun SAX-parseri lukee XML-tiedostoa läpi, se ilmoittaa erilaisista tapahtumista kuten dokumentin alku, dokumentin loppu, alkutagi löydetty jne. Tällainen tapa vaatii vain vähän muistia, sillä XML-tiedoston ei tarvitse olla muistissa missään vaiheessa kokonaan. SAX-parseri toimii siten, että aloitettuaan ei sitä voida pysäyttää ennen, kuin dokumentti on tulkittu loppuun asti. Tässä on myös sen suurin heikkous, sillä missään vaiheessa ei voida pysähtyä ja palata takaisinpäin XML-tiedostossa. Tämä on periaatteessa suuri rajoite, mutta SAX-parserin nopeus perustuu siihen. SAX-parseria käytetään niin, että sille täytyy luoda niin sanottu kuuntelija, joka vastaanottaa parserin ilmoittamat tapahtumat ja toimii niiden perus-teella.(Hunter ym. 2007, 483-486.)

4.2.3 XMLPull-parserointi

SAX-parserointi on niin sanottua push-parserointia. Tämä tarkoittaa sitä, että parseri työntää kuuntelijalleen tapahtumia, jotka liittyvät XML-dokumenttiin. Sovelluksella, jossa käytetään SAX-parserointia, ei ole mitään kontrollia siitä, milloin seuraava par-serointitapahtuma tulee. SAX-parserointia on kritisoitu siitä, että se johtaa yleensä erittäin monimutkaiseen koodiin, jossa ohjelmoija joutuu usein käyttämään sisäisiä muuttujia ja tilakoneita.

Pull-parserointi kääntää SAX-parseroinnin päälaelleen. Siinä sovellus pyytää parse-rilta seuraavan tapahtuman. Tämä johtaa siihen, että tehty koodi käsittelee XML-tapahtumat samassa järjestyksessä kuin mikä XML-dokumentin rakenne on, mikä edesauttaa koodin ymmärtämistä. Pull-parserointi myös eliminoi täysin SAX-

parseroinnissa usein tarvittavia sisäisiä muuttujia ja tilakoneita. Pull-parseroinnista ei vielä ole olemassa virallista standardia, joten sen käyttö riippuu täysin implementaation kehittäjästä. (Slomiski. 2004.)

5 YKSIKKÖTESTAUS

5.1 Yksikkötestaus yleisesti

Yksikkötestaus tarkoittaa sitä, että jokin osa ohjelmakoodista testataan ohjelmallisesti. Yksikkö on pienin mahdollinen testattava osa ohjelmasta. Olio-ohjelmoinnissa tällainen yksikkö on esimerkiksi jonkin luokan metodi. Ideaalisin yksikkötesti on sellainen, että testattava yksikkö testataan täysin eristyksissä muista yksiköistä. Tällöin voidaan varmistua siitä, että yksikkötesti on validi, sillä mikään muu yksikkö ei ole vaikuttanut testitulokseen. Yksikkö on kuitenkin usein muista riippuvainen. Tällöin voidaan käyttää ns. vale-olioita, jotka vaihdetaan todellisten riippuvuuksien tilalle. Vale-oliot eivät toimi kuten todelliset oliot, vaan ne voivat, esimerkiksi palauttaa aina jonkin vakio-arvon.

Yksikkötestauksen tekee yleensä ohjelmoija itse, jolloin hän varmistaa, että tehty koodi toimii, kuten on tarkoitettu. Yksikkötestaukseen on nykyään erittäin hyvät työkalut, joilla voidaan luoda kokonaisia testijärjestelmiä, jotka suorittavat kaikki yksikkötestaukset peräkkäin ja näyttävät testitulokset ohjelmoijalle. Tunnetuin näistä työkaluista on xUnit, joka on itse asiassa joukko eri testausympäristöjä, joilla on samantapainen rakenne. Esimerkiksi Java-ohjelmoinnissa suosittu testiympäristö on JUnit. xUnit-ympäristöjen etuna on se, että ne ovat automaattisia. Ohjelmoijan tarvitsee vain kerran kirjoittaa testi ja liittää se osaksi testiympäristöä. Aina kun ohjelmoija suorittaa testiympäristön, järjestelmä varmistaa, että testissä tulevat arvot ovat halutut. Ohjelmoijan ei tarvitse muistaa, mikä arvo missäkin testivaiheessa täytyi olla.

5.2 Testivetoinen kehitys

Testivetoinen kehitys on sitä, että ohjelmiston suunnittelu ja toteutus tehdään tekemällä ensin testi ja sitten vasta toteuttamalla koodi, joka läpäisee kyseisen testin. Normaalisti ohjelmistokehityksessä järjestys on seuraava: suunnittelu, toteutus ja testaus. Ongelma tässä on se, että yleensä toteutuksen ollessa valmis, ei sitä testata kunnolla. Testivetoisessa kehityksessä ensin luodaan testi. Tämän jälkeen toteutetaan koodi, joka läpäisee testin. Viimeiseksi katsotaan koodia ja suunnitellaan mahdollisimman yksinkertainen ratkaisu. (Koskela 2008, 4-5.)

5.2.1 Etuja ohjelmoinnissa

Vaikka ohjelmistotuotanto on kehittynyt viime vuosina merkittävästi, luodaan vieläkin runsaasti huonoa koodia. Varsinkin, kun uusia teknologioita syntyy jatkuvasti lisää, ei ole ihme, että ohjelmistojen laatu on heikkoa. Testivetoisen kehityksen suurin etu on se, että ohjelmiston luomisen aikana voidaan varmistua, että koodi toimii oikein. Jos virhe löydetään vasta juuri ennen valmistumista, on virheen korjaaminen yleensä erittäin työlästä ja kalliimpaa kuin jos virhe olisi huomattu jo koodia kirjoittaessa. Testivetoisessa kehityksessä virhe huomataan heti, koska testiä ei läpäisty. Toinen tärkeä etu on, että kun testi on luotu ensin, on ohjelmoijan pakko luoda koodi niin, että se pystytään helposti testaamaan. Toisin sanoen on pakko tehdä koodi niin, että sitä on helppo käyttää. Tämä on suuri etu, sillä jos myöhemmin joku muu tarvitsee tehtyä komponentteja, hänen on myös niitä helppo käyttää.
(Koskela 2008, 5.)

5.2.2 Hyöty suunnittelussa

Kuten aiemmin todettiin, testivetoisessa kehityksessä suunnittelu tulee viimeisenä. Toimintoa ei oikeastaan kutsuta suunnitteluksi vaan muunteluksi. Muuntelulla tarkoitetaan sitä, että kun koodi on läpäissyt testin, voidaan sitä myöhemmin muuttaa paremmaksi tai rakenteellisesti erilaiseksi. Koodia on helppo muunnella, sillä se voidaan välittömästi testata, jolloin varmistetaan uuden koodin toimivuus.

Suunnittelu tapahtuu asteittain. Testivetoisessa ajatellaan vain niitä testejä, joita on etukäteen luotu ja joita ei läpäisty. Tehdään vain testin läpäisevä koodi. Kun koodia, joka on läpäissyt vaadittavat testit, on tarpeeksi, katsotaan koodia uudestaan ja tutkitaan sen suunnitelmaa ja mahdollisesti muokataan sitä paremmaksi. Etukäteen ei voida suunnitella ohjelmiston rakennetta niin, että se olisi täydellinen ja se ottaisi huomioon kaikki mahdolliset muutokset, joita kehityksen aikana voi tapahtua. Ohjelmiston suunnitelma siis muuttuu ohjelmiston elinkaaren aikana. Toteutetaan jokin toiminnallisuus ja sen perusteella muutetaan suunnitelmaa ja päinvastoin. Tärkeintä on, säädellä sitä, kuinka paljon etukäteen suunnitellaan. Liika etukäteen suunnittelu johtaa siihen, ettei se yleensä toteudu, jolloin täytyy enemmän panostaa toteutukseen.

On helpompaa ajatella muutamaa asiaa kerrallaan. Käytettäessä perinteistä ohjelmistotuotannon tapaa, ohjelmoijan täytyy pitää koko ajan suunnitelma mielessä. Testivetoisessa kehityksessä ohjelmoijalla on kerrallaan mielessään vain ne muutamat testit, joita ei ole vielä läpäisty. (Koskela 2008, 16-21.)

5.2.3 Ylläpidettävyys

Ohjelmisto annetaan käyttäjälle, kun se saadaan toteutettua vaadittavaan versioon. Lisättäessä ohjelmistoon myöhemmin uusia ominaisuuksia, on se huomattavan helppoa, sillä testijärjestelmällä voidaan varmistua siitä, ettei jo toimiva osuus ohjelmistosta rikkoudu. On myös helppo muokata olemassa olevaa koodia, kun koko ohjelmisto on suojattu kunnon testiympäristöllä.

5.3 Yksikkötestauksen rajat

Kaikkea ei voida testata etukäteen. Yksikkötestauksella voidaan testata yksittäiset komponentit ja niiden välinen toiminnallisuus. Sillä ei kuitenkaan voida testata sitä, tekeekö ohjelmisto sen mitä on vaadittu eli tekeekö se, mitä asiakas haluaa. Tämä täytyy testata itse erityyppisillä työkaluilla.

6 OHJELMISTO

6.1 Java ME-ympäristön kehitystyökalut

Java ME-ympäristön kehitykseen Sun Microsystems tarjoaa ilmaiseksi kaiken tarvittavan, joten ohjelmiston luominen alustalle on edullista. Matkapuhelinvalmistajat tarjoavat myös omista Java ME-toteutuksistaan työkalut, joilla Java ME-sovellus voidaan testata tietokoneella, kuten se olisi laitevalmistajan puhelimessa.

6.1.1 Netbeans kehitysympäristö

Netbeans on Sun Microsystems:n kehittämä täysin ilmainen ohjelmistokehitysympäristö, joka on ensisijaisesti tarkoitettu Java-ohjelmointiin. Tällä hetkellä uusin Netbeans ympäristö tukee myös C++, Ruby, PHP, Javascript, Python-kieliä ja monia muita suosittuja ohjelmointikieliä. Netbeans on tarkoitettu sitomaan monet eri kehitystyökalut yhden ohjelmiston alle, jolloin kehittäjän on helppo työskennellä, vaikkei välttämättä tiedä, miten kaikkia työkaluja yksittäin käytetään. Netbeans soveltuu hyvin Java ME-ohjelmointiin, sillä Java Wireless Toolkit on valmiina Netbeansin mukana. Netbeansin mukana on myös testaukseen tarvittavat työkalut, ja se tunnistaa automaattisesti mm. Nokian ja Ericssonin Java ME-työkalut. Netbeansin haittoihin voidaan lukea se, että se on melko raskas ohjelmisto ja vaatii tehokkaan työaseman, jotta sitä voisi kunnolla käyttää.

Toinen mahdollinen kehitysympäristö olisi ollut Eclipse. Eclipsen huonona puolena on, että sen Java ME-kehitystyökalut eivät ole samaa tasoa kuin Netbeansissa. Myös käytettävyydeltään Eclipse ei yllä samalle tasolle kuin Netbeans.

6.1.2 Nokian S40/S60 SDK

Nokia tarjoaa kehittäjille työkalut, joilla Java ME-sovelluksia voidaan testata ilman oikeaa matkapuhelinta. Nokia tarjoaa työkaluja kahdelle puhelinohjelmistolle. S40 on käyttöjärjestelmänä Nokian perusmatkapuhelimissa. S40 SDK on kehityspaketti,

jossa on mukana käyttöjärjestelmän emulaattori ja Nokian implementaatio Java ME-spesifikaatioon kuuluvista kirjastoista. S60 on Nokian älypuhelimissa oleva käyttöjärjestelmä, jota kutsutaan myös Symbian-käyttöjärjestelmäksi. S60 SDK tarjoaa samat työkalut ja kirjastot kuin S40 SDK, mutta ne on kohdennettu S60 alustalle.

Näissä kehityspaketeissa olevilla emulaattoreilla Java ME-sovellus voidaan testata ilman todellista puhelinta. Sovellusta voidaan myös tutkia kooditasolla ajon aikana, jolloin voidaan reaaliaikaisesti nähdä, mitä sovelluksessa tapahtuu. Emulaattorin käyttö nopeuttaa kehitystä, sillä sovellusta ei tarvitse jokaisen koodimuutoksen jälkeen siirtää aina uudestaan puhelimeen, vaan se voidaan käynnistää suoraan tietokoneella. Kuva 4 on kuvakaappaus Nokian S40 emulaattorista.

Kuva 4. Nokian emulaattori.

6.1.3 Sun Wireless Toolkit

Sun Microsystems tarjoaa ilmaiseksi Java ME-ohjelmointiin omaa työkalua nimeltään Sun Java Wireless Toolkit. Tämä ympäristö sisältää referenssi-implementaation JME-kirjastoista ja emulaattorin, jolla voi tietokoneella testata JME-sovelluksia.

6.1.4 JMunit yksikkötestausjärjestelmä

JMunit on Java ME-ympäristöön tarkoitettu testausjärjestelmä. Se pohjautuu Junit-testausjärjestelmään. JMunit luotiin, koska tavallinen Junit-testaus käyttää Javan reflection-rajapintaa, jota Java ME ympäristössä ei ole. JMunitissa jokaista luokkaa kohden voidaan tehdä testiluokka. Tämä testiluokka on itse asiassa pieni Midlet-sovellus, joka voidaan ajaa puhelimessa. JMunitissa on mahdollista suorittaa yksittäisiä tai testiryhmiä emulaattorissa ja todellisella laitteella. Testiryhmä tarkoittaa, sitä että tehdään ensin joukko testiluokkia, jotka sitten ajetaan yhdellä kertaa. Tämä mahdollistaa koko sovelluksen testaamisen yhdellä testiajolla.

Suurin ongelma testiryhmän ajamisessa on se, että ainakin Nokia ja Motorola eivät salli Midletin käynnistää toisia Midlettejä. Tämä estää testiryhmien ajon, sillä se rakentuu niin, että on yksi midlet, joka näyttää testien tulokset ja käynnistää yksittäiset testiluokat vuorotellen. Tämän vuoksi Nokian emulaattoria tai puhelinta ei voi helposti käyttää yksikkötestauksessa. Sun Microsystemsin tarjoamalla emulaattorilla tämä kuitenkin on mahdollista.

6.2 Suunnittelu

Tämä ohjelmisto päätettiin luoda käyttämällä testivetoista kehitystä. Etukäteen suunniteltiin vain ohjelmiston rakenne. Tähän vaikutti annetut vaatimukset, joihin kuului se, että ohjelmisto voitaisiin räätälöidä helposti eri asiakkaille tarpeen mukaan. Tämän vaatimuksen vuoksi päätettiin, että ohjelmisto toteutettaisiin modulaarisella rakenteella. Tällaisella rakenteella ohjelmistoon olisi helppo lisätä ja poistaa eri komponentteja tarvittaessa.

6.2.1 Vaatimukset

Ohjelmiston vaatimukset ovat seuraavat:

- Ohjelmisto helposti muokattavaksi tarpeen mukaan
- Ohjelmisto toimisi mahdollisimman monissa puhelimissa
- Käyttäjä voi tallentaa tuntikirjauksia puhelimeen
- Puhelin voi pyytää palvelimelta erinäisiä käyttäjäkohtaisia tietoja:
    - Työmääräykset
    - Projektit joissa käyttäjä on osallisena
    - Tuotteisiin liittyvää tietoa
- Puhelin voi lähettää palvelimelle:
    - Kuittauksen, että työmääräys on aloitettu tai lopetettu
    - Tallennetut tuntikirjaukset
- Tuntikirjaukseen liittyvät seuraavat tiedot:
    - Mihin projektiin kirjaus liittyy
    - Mikä työmuoto oli kyseessä
    - Päivämäärä
    - Tunnit
    - Mitä tuotteita käytettiin

6.2.2 Kohdelaitteisto

Kohdelaitteistoksi valittiin Nokian matkapuhelimet. Ohjelmiston tulisi toimia sekä peruspuhelimissa että älypuhelimissa. Matkapuhelimien vaatimuksiin kuului, että ne toteuttavat MIDP 2.0:n ja CLDC 1.1:n vaatimukset. Näitä laitteita ovat mm. Nokia 6110 Navigator, Nokia 3110 Classic ja Nokia E51.

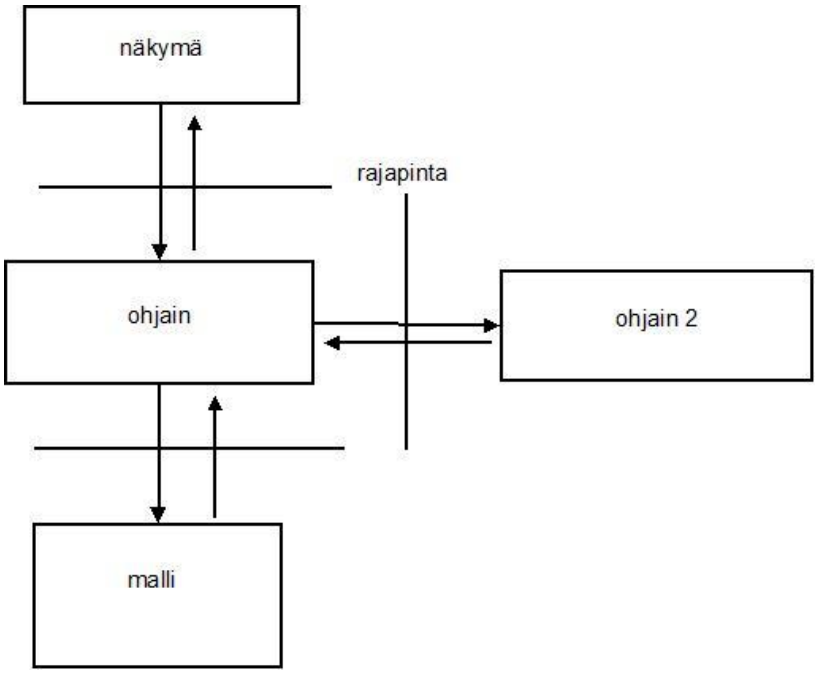
6.2.3 Rajapinnat palvelimeen

Rajapinta palvelimeen on XML-muotoinen. XML-tiedosto sisältää aina lähettävän käyttäjän tiedot ja operaation, mitä palvelimen pyydetään tekemään. Jos palvelimelle lähetetään dataa, on data muotoiltu XML-tiedostoon mukaan vaaditulla tavalla.

Pyyntöoperaatiot on nimetty niin, että niiden etuliitteenä on ”get”. Lähetysoperaatioissa etuliitteenä on ”set”. Tarkempi selvitys on liitteessä 1.

6.3 Arkkitehtuuri

Siitä huolimatta, että käyttöön oli valittu testivetoinen kehitystapa, jokin yleinen arkkitehtuuri ohjelmistolle täytyi päättää. Arkkitehtuurina käytetään MVC-arkkitehtuuria, sillä ohjelmistoon tehtävät komponentit ovat tähän arkkitehtuuriin soveltuvia. MVC:tä toteutetaan niin, että näkymä kutsuu kaikissa tarvitsemissaan toiminnoissa siihen näkymään liittyvää ohjainta. Vaikka toiminto saattaa mennä ristiin toisen ohjaimen kanssa, pyyntö tuohon toiseen ohjaimeen meni aina näkymän oman ohjaimen kautta. Toteutuksen idea on esitelty kuvassa 5.



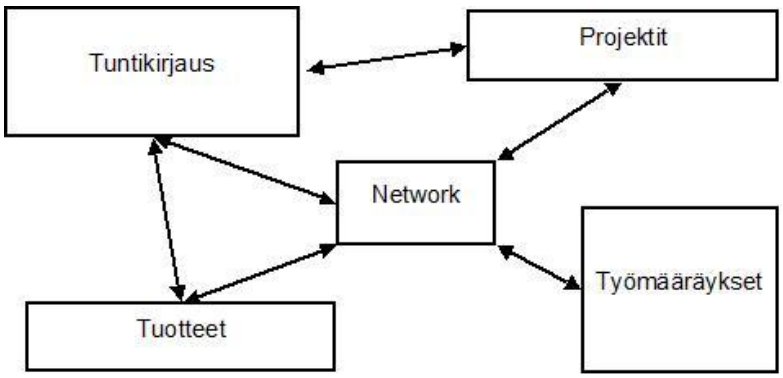
Kuva 5. Yleiskuva MVC:n toteuttamisesta.

Aivan kaikkea ei Java ME-ympäristössä ole järkevää siirtää pelkästään rajapintoihin. Java ME-ympäristössä jotkin luokat ovat sellaisia, että niiden piilottaminen rajapintojen taakse yleensä johtaa siihen, että syntyy pitkiä metodikutsuja, jotka sulautetuissa laitteissa ovat huomattavan hitaita. Tällaisia ovat esimerkiksi luokat, jotka periytyvät Displayable-luokasta. Nämä luokat ovat jonkin tasoisia näkymiä, joita voidaan näyttää matkapuhelimen näytöllä. Jos näkymästä tiedetään vain rajapinta, eikä tiede-

tä, mikä luokka se on, täytyy se aina tarvittaessa muuntaa vähintään Displayble-luokaksi, jotta se voidaan näyttää näytöllä. Sinänsä tämä ei ole ongelma, mutta se täytyy tehdä jokainen kerta käyttäjän vaihtaessa näkymästä toiseen. Tämä hidastaa ohjelman käyttöä. Käytännössä ongelma kierrettiin niin, että ohjain tietää näkymästään enemmän kuin rajapinnan. Ohjain tietää, minkä tyyppinen luokka näkymä on. Näkymä kuitenkin tietää vain ohjaimen rajapinnan.

6.4 Moduulit

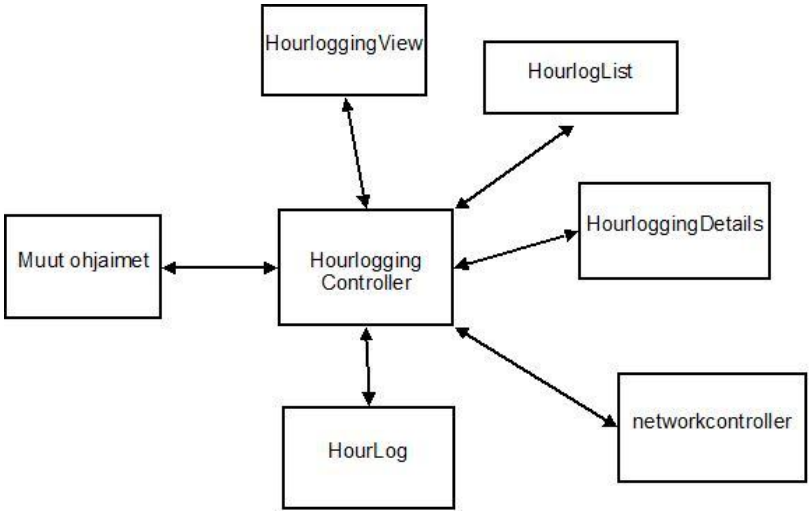
Ohjelmisto on jaettu eri moduuleihin. Moduulit voivat koostua vain muutamasta tai useasta luokasta. Kuvassa 6 näkyvät ohjelmistossa olevat moduulit. Moduulien koko kuvastaa luokkien määrää moduulissa.



Kuva 6. Ohjelmiston moduulit. Nuolet kertovat moduulien välisistä yhteyksistä.

6.4.1 Tuntikirjaus-moduuli

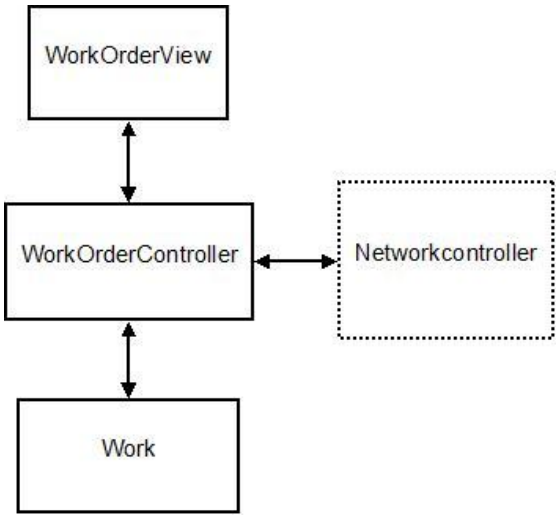
Tuntikirjaus-moduuli on yksi keskeisimmistä moduuleista tässä ohjelmistossa. Se koostuu useasta näkymästä, ohjaimesta ja mallista. Tuntikirjaus-moduulin tehtävänä on käsitellä kaikki toiminnot, jotka liittyvät tuntikirjausten tekemiseen. Käyttäjä voi luoda uusia tuntikirjauksia, tarkastella tallennettuja ja lähettää puhelimeen tallennetut tuntikirjaukset. HourLoggingView-luokka on päänäkymä, jossa käyttäjä voi asettaa uuden tuntikirjauksen tiedot. HourlogList-luokka näyttää puhelimeen tallennetut tuntikirjaukset. HourloggingDetails-luokka näyttää tallennetusta tuntikirjauksesta yksityiskohdat. Kuvassa 7 on esitelty kaikki moduuliin liittyvät luokat.



Kuva 7. Tuntikirjaus-moduulin osat ja yhteydet muihin moduuleihin.

6.4.2 Työmääräykset-moduuli

Työmääräykset-moduuli rakentuu yhdestä näkymästä, ohjaimesta ja mallista. Sillä on myös yhteys Network-moduuliin, koska käyttäjällä on mahdollisuus lähettää palvelimelle tieto, että työmääräyksen tila on muuttunut ja vastaanottaa uusia työmääräyksiä palvelimelta. Moduulin rakenne on esitelty kuvassa 8.



Kuva 8. Työmääräykset-moduulin rakenne.

6.4.3 Tuotteet- ja Projektit-moduuli

Nämä moduulit ovat hyvin samankaltaisia. Ne ovat hyvin yksinkertaisia, sillä ne ovat tarkoitettu vain tietojen tarkasteluun. Molemmissa rakenne on samanlainen kuin Työmääräykset-moduulissa. Tuotteet-moduulin tehtävän on näyttää käyttäjälle tietoa tuotteista, joita palvelimelta on lähetetty kuten tuotteen nimi, tuotekoodi ja mahdollisesti varastosaldo. Projektit-moduulin tehtävän on näyttää käyttäjälle tietoja projekteista, joissa käyttäjä on osallisena. Näitä tietoja voivat olla mm. projektin nimi, koodi ja lyhyt kuvaus. Näiden moduulien käytöstä ei ole mitään vaatimuksia, vaan ne toimivat valmiina pohjana, kun tulee tilanne, jossa halutaan joitakin ominaisuuksia lisätä ohjelmistoon.

6.4.4 Network-moduuli

Network-moduuli on suunniteltu erittäin yksinkertaiseksi. Tämän moduulin ainoa tehtävä on avata verkkoyhteys ja lähettää tai vastaanottaa dataa. Se ei itse prosessoi dataa vaan antaa sen eteenpäin sille moduulille, joka network-moduulia on käyttänyt. Etuna tässä on se, ettei network-moduulia tarvitse muokata, jos datan muotoon tulee jotain muutoksia. Ainoastaan se moduuli, joka on network-moduulia kutsunut, täytyy muuttaa.

7 TOTEUTUS

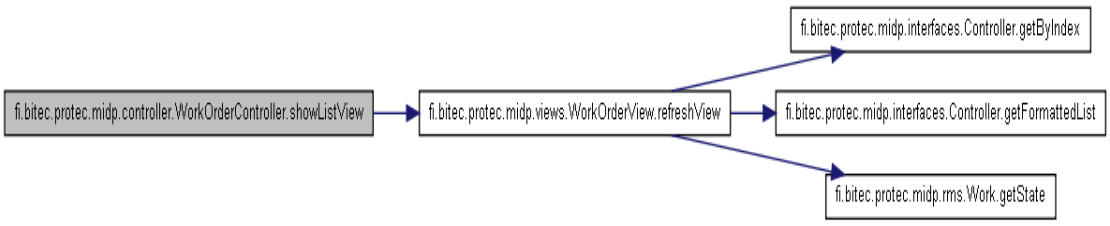
7.1 Toteutus yleisesti

Ohjelmiston toteutus aloitettiin päättämällä, mitä työkaluja käytettäisiin. Käyttöjärjestelmänä käytettiin Windows XP:tä, koska Nokian työkalut toimivat lähes ainoastaan Windows-käyttöjärjestelmissä. Sovelluskehitystyökaluksi valittiin Netbeans 6, sillä se oli kaikista vaihtoehdoista toimivin. Eclipse-ympäristöä harkittiin, mutta sitä ei valittu, koska sen työkalut olivat vaikeakäyttöisempiä. Matkapuhelimen emuloin-

tiin valittiin Nokian S40 SDK:n emulaattori, Nokian S60 SDK:n emulaattori ja Sun Microsystemsin oma referenssi-implementaatio Java ME-emulaattorista. Näin saatiin katettua mahdollisimman monien matkapuhelinten emulointi. Kehitystyökalujen asennus oli yksinkertaista. Netbeansin paketissa on mukana Sunin WTK (Wireless toolkit). Netbeans osaa hakea kaikkia yleisimpiä Java ME-työkaluja ja integroida ne osaksi kehitysympäristöä, joten Nokian työkalujen asennus oli erittäin helppoa.

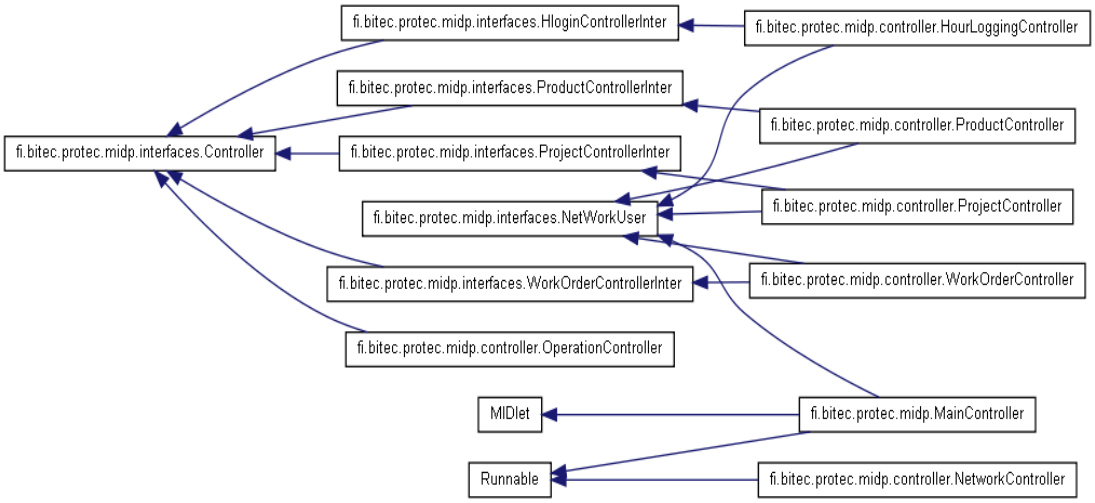
Yksikkötestauksessa käytettiin JMUnit-työkalua, sillä se on Java ME:n suosituimpia testaustyökaluja ja se on integroitu osaksi Netbeans-kehitysympäristöä. Testirunkojen luominen on Netbeansissa vain muutaman hiirenpainalluksen takana, joten se soveltui erinomaisesti testaustyökaluksi. Ohjelmisto tehtiin hyväksikäyttämällä testivetoista kehitystapaa. Bitec Oy:ssä on käytössä Subversion versionhallintajärjestelmä, jota käytettiin tässä projektissa. Versionhallinnan etuina on se, että niiden avulla voidaan jakaa tiedostoja muiden kanssa ja ohjelmiston eri versiot saadaan versionhallinnasta helposti palautettua.

Ohjelmiston dokumentointiin käytettiin Doxygen-työkalua. Doxygen pystyy generoimaan tietyllä tavalla kommentoidusta lähdekoodista html-dokumentin. Doxygen pystyy myös piirtämään funktiokutsujonoja, joista voitiin helposti nähdä jos jokin toiminto oli liian monen funktiokutsun takana. Kuva 9 on kuvattu mitä funktiokutsuja WorkOrderController-luokan showListView-metodi käyttää.



Kuva 9. WorkOrderController-luokan showListView-metodin funktiokutsut

Doxygen-ohjelma generoi myös luokkien välisistä suhteista kuvia, joista on helppo nähdä ohjelmiston eri osien rakenne. Kuva 10 on esitetty ohjaimiin liittyvien luokkien suhteita.



Kuva 10. Ohjainluokkien väliset suhteet.

7.2 Testivetoisen kehitysmenetelmän käyttö

Testivetoista kehitystä käytetään niin, että ensin kirjoitetaan luokan runko, jossa on määritelty metodit, jotka sillä hetkellä tiedetään tarvittavan. Tämän jälkeen käytetään Netbeansin testigeneraattoria, joka luo luokalle rungon tarkastelemalla sen metodeja ja generoimalla niiden perusteella jokaiselle metodille testin. Näihin testirunkoihin kirjoitetaan, miten metodin täytyy käyttäytyä. Suoritetaan testiympäristö, josta huomataan, että testit eivät ole onnistuneet, sillä toteutusta ei ole vielä tehty. Kirjoitetaan vain sen verran koodia, että testi läpäistään. Näin saadaan verrattain nopeasti tehtyä oikein toimiva luokka, jota voidaan heti alkaa käyttää muualla. Luokkaa voidaan parannella, kun muita luokkia alkaa ilmentyä ja toteutus vaatii muutoksia. Jokaisen muutoksen jälkeen ajetaan testiympäristö, jotta voidaan varmistua siitä, ettei mikään muutos riko toiminnallisuutta.

Näkymien kohdalla täytyy tehdä poikkeus, sillä käytettävä yksikkötestaustyökalu JMunit ei tue näkymien testausta. Näkymien omia metodeja voidaan kyllä testata, mutta ohjelmiston rakenteen vuoksi ainoat metodit, joita näkymillä on, ovat vain ne, joissa kuunnellaan komentoja ja joissa luodaan näkymän komponentit. Kaikki todellinen toiminnallisuus, joka täytyy testata, on ohjaimissa ja malleissa. Näkymät testataan käyttämällä emulaattoria ja katsomalla siitä, miten komponentit sijoittuvat näytölle.

7.3 Rajapintojen käyttö

Lähes jokaista luokkaa kohden on ainakin yksi rajapinta. Rakenteen pitää olla modulaarinen ja eri komponenttien vaihtamisen, muuntelun ja testauksen pitää olla helppoa. Rajapintojen etuna on se, että mikään luokka ei suoraan tiedä toisen luokan toteutusta. Muuntelun vuoksi pitää kaikki luokkien välisen kanssakäymisen tapahtua rajapintojen kautta, jotta toteutuksen muokkaaminen ei rikkoisi suoraan muiden luokkien toiminnallisuutta. Sama asia koskee komponenttien vaihtamista.

Rajapinnat helpottivat yksikkötestausta, sillä testit eivät ole periaatteessa kelvollisia jos testattava luokka vaatii suoraan toisen todellisen luokan. Valeluokkien käyttö on helppoa, koska voidaan vain toteuttaa vaadittava rajapinta valeluokalla.

Rajapintojen käyttöä on jonkin verran rajoitettu, koska kuitenkin kyseessä on vähäiset resurssit omaava laite, jolloin liiallinen abstraktio hidastaa ohjelmaa. Esimerkiksi, jokainen metodikutsu aiheuttaa sen, että virtuaalikoneen täytyy etsiä luokka, jolle metodi kuuluu, ja hypätä tähän metodiin suorittamaan sitä.

7.4 Käynnistys-tilanne

Java ME-alustalle tehtävän ohjelman yhden luokan pitää periytyä Midlet-luokasta. Midlet luokassa on kolme tärkeää funktiota. StartApp-funktiota kutsutaan, kun ohjelma käynnistetään. Tässä funktiossa pitää alustaa ohjelman tarvittavat luokat. PauseApp-funktiota kutsutaan, kun käyttäjä esimerkiksi saa puhelun ja vastaa siihen. Ohjelma asetetaan silloin taukotilaan. Tässä kohtaa Java ME-sovellus voi esimerkiksi vapauttaa joitain resursseja tai katkaista verkkoyhteyden. DestroyApp-funktiota kutsutaan kun ohjelma lopetetaan. Tässä sovellus voi esimerkiksi poistaa väliaikaisia tiedostoja tai keskeyttää verkkoliikenteen. Midlet-luokka on siis se, jota kutsutaan kun ohjelma käynnistetään. Tässä ohjelmistossa toteutettiin MainController-luokka, joka periytyy Midlet-luokasta. Tämän luokan tehtävä on luoda ja käynnistää muut luokat ja näyttää näytöllä päävalikko. Käynnistystilanteessa puhelimen näytölle laitetaan välittömästi ilmoitus siitä, että ohjelma on käynnistymässä. Kaikkien välttämät-

tömien luokkien alustus tehdään itse asiassa omassa säikeessä, joka valmistuttuaan asettaa ohjelman päävalikon näkyviin.

MainController on suunnittelumalliltaan Singleton-luokka (ks. 3.4.1). Tällä suunnittelumallilla estettiin se, ettei MainController-luokkaa yritetä vahingossa luoda uudestaan. Jos MainController-luokasta olisi mahdollista luoda useampi kuin yksi instanssi, kaatuisi ohjelmisto useimmissa puhelimissa.

7.5 Näkymät

Näkymien luontiin käytettiin vain Java ME-kirjastojen peruskomponentteja. Kaikki näkymät periytyvät joko List-luokasta tai Form-luokasta. List-luokka on listanäkymä johon jokaiseen listan kenttään voi valinnaisesti laittaa myös kuvan. Form-luokka on Java ME:n perusnäkymäluokka johon voidaan asettaa erilaisia Item-luokasta periytyviä komponentteja.Valmiita komponetteja on paljon, kuten teksti, kuva, valintalaatikko ja päivämäärä-kenttiä. Kaikissa näkymissä käytettiin näitä valmiita komponentteja, jotta saavutettaisiin mahdollisimman hyvä yhteensopivuus. S60-alustaa varten jouduttiin käyttämään muutamaa lisäkomponenttia. Joissakin S60-alustan puhelimissa näyttö on niin leveä, että komponentit näkyvät vierekkäin, eivätkä allekkain. Tämä ongelma ratkaistiin niin, että käytettiin Spacer-komponenttia. Spacer-komponentti on tarkoitettu sijoitettavaksi komponenttien väliin luomaan joko ylimääräinen tyhjä tila tai pakottamaan seuraava komponentti seuraavalle riville. Ongelma ratkaistiin määrittämällä komponentti pakottamaan toinen komponentti toisen alapuolelle.

Jokainen näkymä kuuntelee itse käyttäjän painalluksia. Näiden painallusten perusteella kutsutaan ohjainta. Mikäli ohjain olisi ollut kuuntelijana komennoille, olisi joidenkin ohjaimien koodi paisunut vaikeasti ylläpidettäväksi. Näkymille ei tehty omia rajapintoja. Koska näkymän näyttämistä varten olisi tarvittu ylimääräistä koodia ja tiedettiin että näkymiä vaihdeltaisiin paljon, jolloin ohjelman suoritusnopeus olisi kärsinyt (ks.6.3). Komentojen kuuntelu toteutetaan niin, että näkymä toteuttaa CommandListener-rajapinnan. Tämä rajapinta vaatii CommandAction-metodin toteuttamisen. Java ME:n virtuaalikone kutsuu tätä metodia, kun käyttäjä valitsee jonkin komennon. Metodi saa parametrina komennon, jota käytettiin, ja näkymän, jolle komento kuuluu.

7.6 Mallit

Malli koostuu kahdesta luokasta: luokasta, joka kuvaa haluttua tietoa ja niin sanottua Data Access Object:a (DAO), joka huolehtii tietojen tallentamisesta puhelimeen. Dao:t käyttävät sisäiseltä toiminnoltaan Java ME:n RecordStore-luokkaa. Sen etuna on siinä, ettei se tarvitse tiedostonkäsittely-oikeuksia, joita tavallinen tiedostoon kirjoittaminen tarvitsee. RecordStore on erittäin pieni ja yksinkertainen tietokanta. Siinä voidaan luoda erilaisia tauluja, kuten normaalissa tietokannassa, mutta tieto tallennetaan aina bitteinä. Jokainen tietue koostuu biteistä ja uniikista numerosta. tämän uniikin numeron avulla RecordStoresta voidaan hakea aina halutut bitit. Kaiken ollessa bitteinä, pitää kaikki tallennettava tieto muuttaa bittimuotoon ennen tallentamista. DAO:t tehdään niin, että sille voidaan antaa luokka, ja se muuttaa sen automaattisesti biteiksi ja tallentaa tietokantaan. Myös tiedon haku tietokannasta toimii niin, että DAO:lle annetaan id ja se hakee bitit tietokannasta ja muuntaa sen automaattisesti olioksi.

Dao:t käyttävät erilaisia suodatus- ja komparaattori-luokkia. Komparaattorilla voidaan tietokannasta palautettava tieto järjestää halutulla tavalla vaikka tieto olisi tallennettu bitteinä. Esimerkiksi projektit palautetaan niin, että ne ovat projekti-id:n mukaisesti suurimmasta pienempää järjestetty. Tämä id on sama kuin palvelimen tietokannassa on. Suodatusluokalla voidaan hakea tietokannasta esim. tietyn päivämäärän mukaan haluttu tieto. Tämä helpottaa palvelimen ja puhelimen välistä kanssakäymistä kun molemmat järjestelmät käyttävät samoja tunnisteita. Tietojen järjestäminen ja suodatus on etukäteen aina päätetty, missä sitä käytetään, sillä näiden toimintojen käyttö on hitaampaa kuin normaali tietokanta-palautus.

DAO-luokille määriteltiin yleinen rajapinta, joka on esitelty kuvassa 11. Siinä on määritelty metodit, joita kaikissa näissä luokissa täytyy olla. Näitä ovat id:llä haettava tieto, kaikkien tietokannassa olevien tietojen palautus, tietokannasta poisto id:llä ja tietokannassa olevan tietueen päivitys id:llä.

Dao-luokat luodaan ns. Factory-luokalla (ks.3.4.2). Tällä pyrittiin siihen, että kun ohjelmistoa halutaan räätälöidä, niin uusien Dao-luokkien lisääminen ja vaihtaminen, olisi mahdollisimman vaivatonta.

```
public interface DAOInter {
    /**
     * Valitsee rms-kannasta rsid:n perusteella tietueen.
     */
    public Object select(int rsid);
    /**
     * Valitsee kaikki tietueet kannasta
     */
    public Vector selectAll();
    /**
     * Asettaa olion kantaan.
     */
    public boolean insert(Object obj);
    /**
     * Päivittää olion tiedot kantaan.
     */
    public boolean update(Object obj);
    /**
     * Poistaa tietueen kannasta rsid:n perusteella
     */
    public boolean delete(int rsid);
    /**
     * Kaikki poistetaan.
     */
    public boolean deleteAll();
    /**
     * Valitse daosta jonkin toisen id:n mukaan tämä id on implementaatiokohtainen
     */
    public Object selectById(int pid);
}
```

Kuva 11. DAO-rajapinta.

7.7 Ohjaimet

Ohjaimia varten on luotu yleinen Controller-rajapinta, joka on esitelty kuvassa 12. Tämän lisäksi Tuntikirjaus-ohjainta varten on oma rajapintansa, joka laajentaa yleistä rajapintaa. Jokainen ohjain lataa muistiin mallista saadut tiedot. Tämä siksi, että nykypuhelimissa alkaa olla melko paljon muistia, ja koska muut ohjaimet tarvitsevat toisilta ohjaimilta usein tietoa, on se huomattavasti nopeampaa ottaa suoraan muistista kuin hakea se joka kerta mallista. Tämä ominaisuus on mahdollista vain, kun tiedetään, että palvelimelta lähetetään melko vähän tietoa puhelimeen. Ohjaimet toimivat myös niin, että kun esimerkiksi tuntikirjauksia lähetetään palvelimelle ja palvelin vastaanottaa ne onnistuneesti, ne poistetaan puhelimesta välittömästi. Tämä ominaisuus on toteutettu niin, että ohjaimet käyttävät Vector-luokkaa, johon laitetaan mallista saadut oliot. Vector-luokka on tavallaan taulu, jonka kokoa ei tarvitse etukäteen määritellä vaan sinne voidaan lisätä vaihtuva määrä olioita.

```
public interface Controller {
    /**
     * Tällä metodilla controllerilta voi pyytää listaindeksiä vastaava olio
     */
    public Object getByIndex(int index);
    /**
     * Controlleri palauttaa hallitsemiensa datan String tauluun muotoiltuna
     */
    public String[] getFormattedList();
    /**
     * Controlleri päivittää sisäiset muuttujansa kuvaamaan nykyistä tilannetta
     */
    public void refresh();
    /**
     * Asettaa controllerille midletin display -olion, jota voidaan käyttää esim. viewien vaihtamiseen
     */
    public void setDisplay(Display d);
    /**
     * Pyytää controllerilta jonkin id:n perusteella oliota.
     */
    public Object getById(int id);
    /**
     * Controlleri asettaa modeliin vectorissa olevat arvot
     */
    public void setVector(Vector vec);
    /**
     * Lähettää pyynnön palvelimelle
     */
    public void sendGet();
}
```

Kuva 12. Controller-rajapinta.

7.7.1 XMLController

Palvelimelle lähetettävän datan pitää olla XML-muotoiltu. Koska ohjaimet ovat itse vastuussa lähetettävän datan muodosta ja vastaanotettavan datan tulkitsemisesta, täytyy niissä jotenkin käsitellä XML-muotoista tietoa. Vastaanotetun XML-dokumentin käsittely on delegoitu XMLController-luokalle, jonka rajapinta on esitelty kuvassa 13. Se käyttää kXML-kirjastoa XML:n parserointiin. kXML on suunniteltu rajoitettuihin ympäristöihin kuten Java ME-ympäristö ja Se pohjautuu XMLPull-rajapintaan (ks. 4.1.3). XMLController on tehty niin, että se pystyy käsittelemään kaikki palvelinrajapinnassa (ks. Liite 1) olevat XML:t. Kun XMLController saa XML-dokumentit parseroitavaksi, se rakentaa sisäisiin muuttujiin talteen kaiken datan, jota se parseroi dokumentista. Esimerkiksi jos XML-tiedosto sisältää tietoa projekteista ja tuotteista, niin XMLController rakentaa kaikista projekti- ja tuotetiedoista omat olionsa ja asettaa ne listaan. Nämä listat voidaan pyytää XMLControllerilta, kun parserointi on valmis. Jos XML-dokumentissa ei ollut yhtään tuotetietoa, on luokan sisäi-

nen tuotelista tyhjä. Tällaista rajapintaa on helppo käyttää, sillä varsinaista XML-dokumenttia ei tarvitse koskaan nähdä sellaisenaan, vaan se parseroidaan heti ja XMLController palauttaa vain erilaisia olioita.

```
public interface XMLControllerInter {
    /**
     * Parseroi annetun datan valmiiksi.
     */
    public int parse(byte[] xmldata);
    /**
     * Palauttaa parseroidusta xmldatasta työmääräykset
     */
    public Vector getWorks();
    /**
     * Palauttaa parseroidusta xmldatasta operaatiot
     */
    public Vector getOperations();
    /**
     * Palauttaa parseroidusta xmldatasta projektit
     */
    public Vector getProjects();
    /**
     * Palauttaa parseroidusta xmldatasta tuotteet
     */
    public Vector getProducts();
    /**
     * Palauttaa xml-tiedostossa olevan errorcodetagin sisällön.*/
    public int getErrorcode();
}
```

Kuva 13. XMLControllerin rajapinta.

7.7.2 HourLoggingController

Tämä ohjain on vastuussa tuntikirjaukseen liittyvistä toiminnoista. Sen rajapinta laajentää yleistä Controller-rajapintaa muutamilla lisätoiminnoilla, kuten Save-toiminto eli tallennustoiminto ja näkymien käsittelyyn tarvittavia metodeja. Kuvassa 14 on näytetty tärkeimmät metodit rajapinnasta.

```
public interface HloginControllerInter extends Controller{
    public void setListChoice(int type,int index);
    public void save(Date date, Date hours);
    public void setSelectedProducts(boolean [ ] selections);
    public void sendHourlogs();
    }
```

Kuva 14. Laajennetun rajapinnan tärkeimmät metodit.

Tuntikirjaukseen liitetään projekti ja operaatio. SetListChoice-metodia käytetään lisäämään nämä tiedot uuteen tuntikirjaukseen. Se ottaa parametrina tyypin ja indek-

sin. Tyyppi on joko projekti tai operaatio. Indeksi on listanäkymistä tuleva sillä hetkellä valitun komponentin indeksi. Näiden tietojen avulla ohjain kysyy muilta ohjaimilta oikeat arvot, jotka liitetään tuntikirjaukseen. Save-metodi nimensä mukaisesti tallentaa tuntikirjauksen puhelimen tietokantaan. Se myös tarkistaa, että käyttäjä on antanut tunnit ja päivämäärän. Muussa tapauksessa ilmoitetaan käyttäjälle siitä. Tuntikirjaukseen voi lisätä tuotteita, joita työssä on käytetty. SetSelectedProducts-metodi ottaa vastaan käyttäjän valitseman tuotteet. Metodin parametrina annetaan taulu, jossa taulun indeksi vastaa listanäkymässä valitun tuotteen indeksiä. Jos taulun kentän arvo on tosi niin, tuote on valittu. SendHourlogs-metodi lähettää palvelimelle kaikki tuntikirjaukset, jotka ovat tietokannassa.

7.7.3 WorkOrderController

Tämä ohjain toimii niin, että kun käyttäjän valitessa jonkin työmääräyksen ja painaa esimerkiksi ”Aloita työ”-komentoa otetaan palvelimeen välittömästi yhteys ja lähetetään tieto, että työ on aloitettu. Samoin, kun käytetään komentoa ”Työ valmis”, lähetetään tästä palvelimelle tieto. Jos palvelimelle ei saada yhteyttä, tallennetaan tieto puhelimeen ja lähetetään se seuraavalla yrityskerralla uudestaan. Tieto tallennetaan suoraan samaan tietokantaan, johon työmääräykset ovat tallennettu ja työmääräyksestä laitetaan sen tunniste ja muuttunut tila toiseen tietokantaan odottamaan kun yhteys palvelimeen on mahdollista. Edellä mainittu toiminnallisuus on toteutettu sendStatusChange-metodilla. Metodin parametreina ovat tila, johon muutetaan ja valitun työmääräyksen indeksi.

7.8 NetworkController

Tämä ohjain ei toteuta yleistä Controller-rajapintaa, koska sen toiminta eroaa muista ohjaimista. Ohjaimen vastuulla on datan lähetys ja vastaanotto palvelimelta. Luokan käyttö vaatii, että sitä käyttävä luokka toteuttaa NetworkUser-rajapinnan, joka on esitelty kuvassa 15. Tämä rajapinta koostuu kahdesta metodista, getRequestData ja requestComplete-metodista. Rajapinnan toteuttava luokka annetaan NetworkController-luokalle, jolloin se tilanteen mukaan osaa kutsua oikeaa luokkaa. NetworkController käynnistää uuden säikeen, joka ottaa yhteyttä palvelimeen ja lähettää datan,

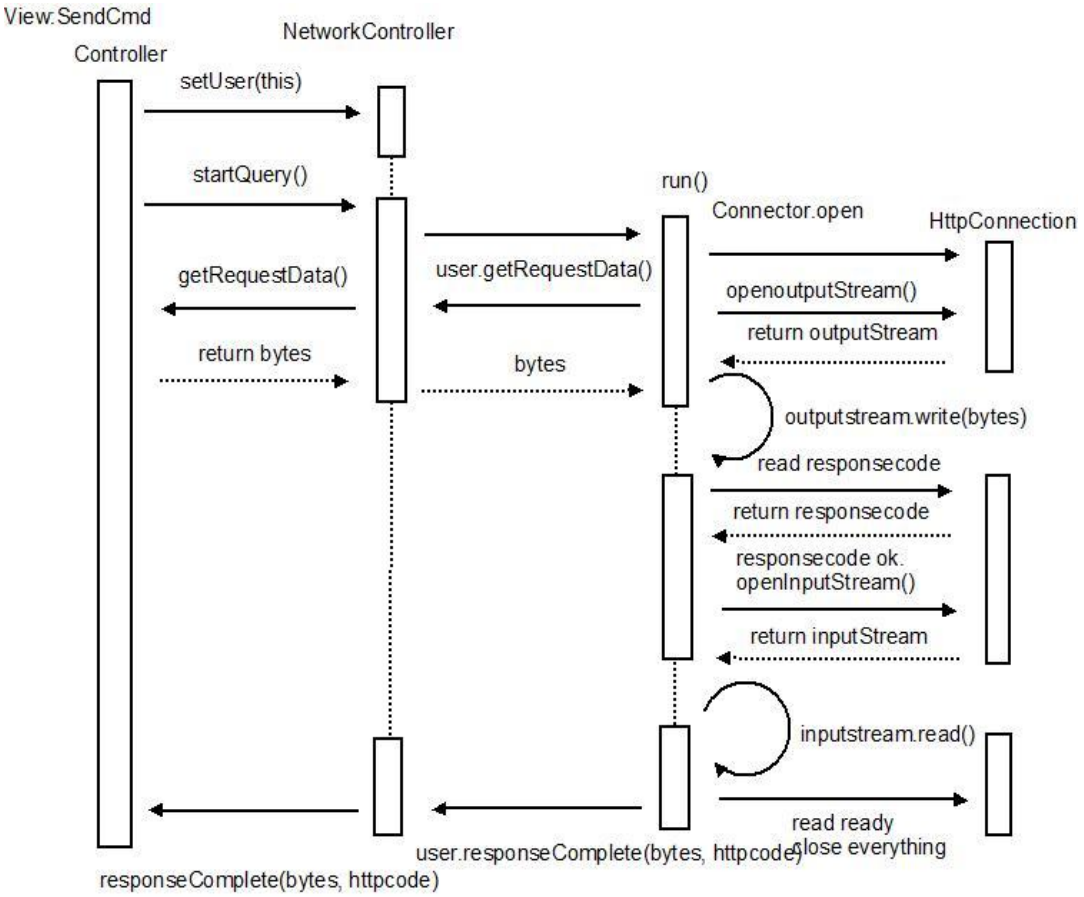
sekä vastaanottaa palvelimelta dataa. NetworkController kutsuu sitä käyttävän luokan getRequestData-metodia, jolloin koostetaan lähetettävä data sopivaan muotoon. Kun NetworkController on vastaanottanut palvelimelta datan, se välittää sen edelleen kutsumalla sitä käyttävän luokan requestComplete-metodia, johon se antaa parametrina datan ja HTTP-kerroksen statuskoodin. Käyttämällä säikeistystä vältytään siltä, että ohjelman käyttöliittymä jumiutuisi palvelinpyynnön ajaksi. Palvelinrajapinnan vaatima salaus (ks. Liite 1) on toteutettu käyttämällä https-protokollaa tiedonsiirrossa. Tämän käyttö on helppoa, sillä ainoa ero yhteyden luontiin on käyttää HttpsConnection-luokkaa HttpConnection-luokan sijasta.

NetworkController-luokan suunnittelumallina on käytetty observer-mallia (ks.3.4.3). Luokka, joka implementoi NetWorkUser-rajapinnan, toimii NetworkController-luokan tarkailijana. Observer-mallista poiketaan hieman, sillä NetworkController-luokalla voi olla vain yksi tarkkailija, sillä operaatiot, jotka se suorittaa koskevat vain sen tarkkailijaa. Koska verkkoliikenne on riippuvainen monesta eri tekijästä, kuten verkon nopeudesta ja liikennemäärästä, soveltuu observer-malli erinomaisesti tähän tilanteeseen. NetworkController-luokan käyttäjän ei tarvitse kysellä luokalta tilatietoja vaan se voi odottaa kunnes NetworkController ilmoittaa olevansa valmis.

```
public interface NetWorkUser {
    /**
     * Kun networkcontrollerin pyyntö on valmis niin se ilmoittaa tämän metodin implementoijalle
     */
    public void requestComplete(int errorcode, byte[] data);
    /**
     * Implementoija palauttaa tällä funktiolla datan jonka haluaa requestiin laittaa
     */
    public byte[] getRequestData();
}
```

Kuva 15. NetworkUser-rajapinta.

NetworkController-luokka itse koostuu vain muutamasta metodista. setUser-metodista, jolla asetetaan NetworkUser-rajapinnan toteuttava luokka, säikeen käynnistävästä startQuery-metodista ja run-metodista joka suoritetaan, kun säie käynnistetään. Kuvassa 16 on esitetty NetworkController-luokan toiminta.



Kuva 16.NetworkControllerin toiminto.

8 TESTAUS

8.1 Testaus yleisesti

Testaus oli kolmivaiheinen. Ensimmäiseksi ohjelmiston toteutuksen aikana käytettiin yksikkötestausta, jolla varmistuttiin siitä, että jokainen yksittäinen osa toimi tarkoitetulla tavalla. Kun ohjelmisto oli saatu siihen vaiheeseen, että voitiin testata palvelinyhteyttä, piti tätä varten luoda testipalvelin. Palvelimelle luotiin yksinkertainen palvelinsovellus, joka vastasi spesifikaation määrittelemällä tavalla. Näin voitiin simuloida palvelimen toimintaa ohjelmistolle. Toisessa vaiheessa kyse olikin ohjelmis-

ton testaamisesta emulaattorilla testipalvelinta vasten. Kolmas vaihe oli testata ohjelmistoa oikealla matkapuhelimella testipalvelinta vasten.

8.2 Testiympäristö

Testiympäristönä käytettiin työn toteutukseen käytettyä työasemaa, johon oli asennettu Apache Tomcat 6 palvelinohjelmisto. Tähän palvelimeen luotiin Java-kielellä palvelinsovellus, joka toteutti spesifikaation (Liite 1.) määrittelemän palvelinrajapinnan. Testipuhelimena käytettiin Nokian E51 puhelinta, tällöin tiedonsiirrossa voitiin käyttää WLAN-verkkoa, joka on täysin ilmaista verrattuna 3G-verkon käyttämiseen.

8.3 JMunitin käyttö

JMunitin käyttö on Netbeans-ympäristössä helppoa. Netbeansilla voidaan olemassa olevasta luokasta generoida suoraan testiluokka, joka liitetään automaattisesti osaksi testausjärjestelmää. Testivetoista kehitystä käytettäessä luodaan luokan runko, jossa on kaikki metodit valmiina ilman toteutusta. Tästä luokasta voidaan Netbeansissa luoda testiluokka, joka tekee jokaista metodia varten testimetodit valmiiksi. Näihin testimetodeihin kirjoitetaan jokaista oikeaa metodia varten testit. JMunitissa on valmiina suuri määrä erilaisia assert-metodeja. Assert-metodilla voidaan testata, että jokin arvo on oikea. Esimerkiksi AssertEquals-metodilla voi testata kahden muuttujan arvoa. Jos ne ovat samat, testi onnistui, jos ei, niistä ilmoitetaan testijärjestelmälle.

Testit voidaan suorittaa käynnistämällä testijärjestelmä emulaattorissa. Testijärjestelmä on itse asiassa Midlet-luokka, joka voidaan suorittaa kuten mikä tahansa muu Java ME-sovellus. Testijärjestelmä koostuu näkymästä, jossa testauksen aikana näkyy testien määrä, läpäisyjen määrä, epäonnistuneiden määrä ja testaukseen kulunut aika. Kun testit on suoritettu, ilmoittaa testijärjestelmä epäonnistuneet testit ja testiin liittyvän virheen.

8.4 Testaustulokset

Testipalvelimen avulla ohjelmisto testattiin emulaattorilla ja varsinaisella puhelimella. Testeissä keskityttiin siihen, että ohjelmisto oli helppo käyttää ja se pystyi käsittelemään mahdolliset virhetilanteet. Testeissä huomatut ongelmat kirjattiin ylös ja korjattiin välittömästi. Koska ohjelmistolla ei vielä varsinaisesti ollut käyttäjiä, jäivät käyttäjäkokemukset selvittämättä.

Ohjelmiston käyttäminen todettiin vaatimusten mukaiseksi. Käyttöliittymä oli helppokäyttöinen ja selkeä. Testauksessa testattiin seuraavia virheitä: palvelin palauttaa HTTP-kerroksen statuskoodiksi jonkin muun kuin 200, palvelin palauttaa XML-dokumentissa virhekoodiksi muun kuin -1, palvelin ei vastaa ja palvelinosoite puuttuu. Ohjelmisto selviytyi virhetilanteista ongelmitta.

9 JATKOKEHITYS

Ohjelmiston jatkokehitys on riippuvainen mahdollisten asiakkaiden tarpeista. Esimerkiksi projekti- ja tuotenäkymä ovat erittäin yksinkertaisia, eivätkä sisällä mitään erikoista toiminnallisuutta. Ne on luotu valmiiksi pohjiksi, kunnes tiedetään millaisia ominaisuuksia näissä näkymissä tarvitaan. Ohjelmistoa kehittäessä syntyi uusia toteutukseen liittyviä ideoita, joihin projektin aikataulu ei riittänyt. Alla on lueteltu oleellisimmat.

Ohjaimiin olisi voitu liittää RecordListener-rajapinnan toteutus. Tämän rajapinnan toteuttava luokka voidaan liittää kuuntelijaksi tietokantaan. Kaikki muutokset tietokantaan aiheuttavat ilmoituksen kuuntelijalle. Tällä vältyttäisiin siltä, että aina tietokantaan päivittäessä täytyy kutsua ohjaimien refresh-metodia, jotta ne saavat myös kaikki päivitykset.

Näkymät eivät näytä kaikissa puhelimissa samanlaisilta. Parhaiten eron huomaa, kun ohjelmistoa vertaa S40- ja S60-puhelimien välillä. S60-puhelimissa, esimerkiksi tuntikirjaus-näkymä näyttää erittäin karulta, sillä Symbian-käyttöjärjestelmä piirtää pai-

nike-komponentit yksinkertaisilla viivoilla. Ongelma voitaisiin kiertää rakentamalla näkymät SVG-kuvista tehdyillä komponenteilla, joita Netbeans-ympäristössä on valmiina. SVG-kuvien käyttö taas rajoittaisi pois joitakin puhelinmalleja, jotka eivät tue Java ME:n SVG-rajapintaa.

Käynnistystilanteessa käyttäjälle näytetään vain yksinkertainen teksti, jossa lukee, että ohjelmaa käynnistetään. Tämän tilalla olisi hyvä näyttää jokin logo ja käynnistyksen edistymispalkki.

Ohjelmiston toteuttamista J2MEPolish-työkaluilla voisi tutkia. J2MEPolish on kokoelma eri työkaluja, jotka on tarkoitettu helpottamaan Java ME-sovellusten tekoa. J2MEPolish sisältää muun muassa käyttöliittymän tekemiseen tarkoitetun Lush-kirjaston, jolla on erittäin helppo suunnitella ja toteuttaa tyylikkäitä näkymiä. J2MEPolish on ilmainen, kun tehdään GPL-lisenssin alaista ohjelmistoa, mutta maksullinen, jos tehdään kaupallisia sovelluksia.

10 YHTEENVETO

Projektin tavoitteena oli luoda ohjelmisto matkapuhelimeen. Ohjelmisto piti suunnitella rakenteeltaan niin, että sitä olisi helppo muokata ja laajentaa. Ohjelmiston piti toimia osana projektinhallintajärjestelmää. Ohjelmiston vaatimuksiin kuului, että se pystyy lähettämään ja vastaanottamaan tuntikirjauksiin, työmääräyksiin, projekteihin ja tuotteisiin liittyvää dataa verkon yli.

Ohjelmiston täytyi toimia mahdollisimman monessa matkapuhelimessa ilman muutoksia. Tämän vuoksi ohjelmiston toteuttamiseen valittiin Java ME-teknologia, sillä se on tuettuna lähes kaikissa markkinoilla olevissa matkapuhelimissa.

Ohjelmiston toteutuksessa sovellettiin testivetoista kehitysmenetelmää. Tässä menetelmässä testit luodaan ensin ja vasta sitten tehdään toteutus. Testivetoinen kehitysmenetelmä kasvattaa koko ajan suosiotaan ja sitä pidetään hyvänä ohjelmistokehitysmenetelmänä. Testivetoinen kehitys pakotti rakentamaan ohjelmiston modulaariseksi, jotta se olisi helppo testata. Modulaarisesta rakenteesta oli myös ohjelmiston vaatimusten kannalta hyötyä, sillä näin ohjelmistosta tuli helposti muokattava ja laajennettava. Modulaarisen rakenteen toteuttamisessa onnistuttiin kohtalaisesti, muutamia Java ME-ympäristön rajoittamia tilanteita lukuun ottamatta.

Toteutuksessa käytettiin suunnittelumalleja, joilla ohjelmisto saatiin rakenteeltaan helposti ylläpidettäväksi. Arkkitehtuurina käytettiin MVC-arkkitehtuuria, jossa on eroteltu näkymät, ohjelmalogiikka ja tiedon käsittely toisistaan.

Datan siirtoon verkon yli käytettiin XML:ää. Palvelin vastasi lähettämällä XML-muotoisen vastausviestin. Tämä vastausviesti tulkittiin käyttämällä XMLPull-tyyppistä parserointimenetelmää. Tämän tyyppisellä menetelmällä viestin tulkitsemisen toteuttaminen oli yksinkertaisempaa ja se toimi nopeasti.

Projektissa onnistuttiin toteuttamaan ohjelmisto, jota on helppo muokata ja laajentaa ja se toimii vaatimusten määrittelemällä tavalla.

LÄHTEET

Cadenhead, R. & Lemay, L. (2007). Sams Teach Yourself Java 6. USA: Sams Publishing.

Li, S. & Knudsen, J. (2005). Beginning J2ME 3rd Edition. USA: Apress.

Bass, L. Clementes, P. Kazman, R. (2003). Software architecture in practice 2nd edition. USA: Addison Wesley.

Koskela, L. (2008). Test Driven Practical TDD and acceptance TDD for Java developers. USA: Manning Publications Co.

Hunter, D. Rafter, J. Fawcett, J. Van der Vlist, E. Ayers, D. Duckett, J. Watt, A. McKinnon, L.(2007). Beginning XML 4th Edition. USA: Wiley Publishing Inc.

Holzner, S. (2006). Design patterns for dummies. USA: Wiley Publishing Inc.

Gamma, E. Helm, R. Johnson, R. Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. USA: Addisson-Wesley Professional Computing Series.

Holub, A. 2003. Why extends is evil [verkkodokumentti]. Javaworld[Viitattu 4.3.2009]. Saatavissa: http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-toolbox.html?page=1.

Fowler, M. 2006. Gui architectures [verkkodokumentti]. Martinfowler.com[Viitattu 27.2.2009]. Saatavissa: http://www.martinfowler.com/eaaDev/uiArchs.html.

Slomiski, A. 2004. On using XML pull parsing java APIs [verkkodokumentti]. xmlpull.org [Viitattu 5.4.2009]. Saatavissa:

http://www.xmlpull.org/history/index.html

LIITTEET

Liite 1 Palvelinrajapinta

LIITE 1, Palvelinrajapinta

Bitec Protec
Mobiilirajapinnat

Versio	0.2
Tila	Vedos
Päivämäärä	24.02.2009
Omistaja	Janne Raitaniemi
Alkuperäinen tekijä	Janne Raitaniemi
Tyyppi	Suunnitelma
Dokumentin nimi	Protec_Interface_04.doc
Talletussijainti	Palvelin

1 Päivitys historia

Versio 0.1	**24.2.2009**	**Janne Raitaniemi**	**Vedos**
Syy:	Ensimmäinen versio		
Kuvaus:	-		
Versio 0.2	**24.2.2009**	**Tuomas Korjonen**	**Vedos**
Syy:	Toinen versio		
Kuvaus:	-		
Versio 0.3	**03.04.2009**	**Tuomas Korjonen**	**Vedos**
Syy:	Kolmas versio		
Kuvaus:	-		
Versio 0.4	**07.04.2009**	**Tuomas Korjonen**	**Vedos**
Syy:	Tarkennus		
Kuvaus:	Lisätty työmääräysten lähetykseen timestamp-arvo		

2 Sisällysluettelo

3 Lyhenteet ja yleiset ohjeet

1.1 Lyhenteet

Lyhenne	**Kuvaus**
XML	Extended Markup Language
GSM	Global System for Mobile Communications

1.2 Yleiset ohjeet

Mitään erityistä merkitsemiskäytäntöä ei ole käytetty

4 Esittely

Dokumentti kuvaa Protec-järjestelmän rajapinnat. Pääpiirteittäin palvelimelle lähetetään XML-dokumentti. Vastaukset tulevat XML-muodossa.

5 Rajapinta

5.1 Mobiilikutsu ja parametrit

Matkapuhelin kutsuu palvelinta kutsulla: http://domain:port/protec/mobile/

Matkapuhelin lähettää http post-metodilla XML tiedoston alla olevan esimerkin mukaisesti.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<request>
        <username>matti</username>
        <password>salasana</password>
        <action>getprojects</action>
        <data>
                …
        </data>
</request>
```

Jos XML:ssä on data kenttä, niin silloin matkapuhelin lähettää tietoa palvelimelle. Viesti lähetetään joko https:n yli tai tärkeä tieto salataan ennen lähetystä.

5.2 Getprojects tapahtuma

Kun pyyntö xml:ssä on action getprojects, palauttaa palvelin matkapuhelimeen projektien tietoja esimerkin mukaisena xml:nä.

5.2.1 Pyyntösanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<request>
        <username>matti</username>
        <password>salasana</password>
        <action>getprojects</action>
</request>
```

5.2.2 Vastesanoma

```
<?xml version="1.0" encoding="UTF-8"?>
<projects>
        <project><id>123</id><code>123</code><name>Projekti123</name></project>
        <project><id>124</id><code>124</code><name>Projekti124</name></project>
        <project><id>125</id><code>125</code><name>Projekti125</name></project>
</projects>
```

Vastauksena tulee joko vain henkilön omat projektit oletuksena tai sitten voidaan toimittaa kaikki.

5.3 Getproducts tapahtuma

Kun action getproducts, palauttaa palvelin matkapuhelimelle tuotteiden tietoja esimerkin mukaisena xml:nä.

5.3.1 Pyyntösanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<request>
        <username>matti</username>
        <password>salasana</password>
        <action>getproducts</action>
</request>
```

5.3.2 Vastesanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<products>
<product>
        <id>123</id>
        <code>123</code>
        <name>Tuote1</name>
        <amount>5</amount>
</product>
<product>
        <id>124</id>
        <code>124</code>
        <name>Tuote2</name>
        <amount>2</amount>
</product>
</products>
```

Vastauksena voidaan palauttaa vain henkilön projektien tuotteet tai kaikki tuotteet.

5.4 Getworks tapahtuma

Kun action getworks, palauttaa palvelin matkapuhelimelle työmääräykset esimerkin mukaisena xml:nä

5.4.1 Pyyntösanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<request>
        <username>matti</username>
        <password>salasana</password>
        <action>getworks</action>
</request>
```

5.4.2 Vastesanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<works>
  <work>
        <id>123</id>
```



```
        <startdate>12.12.2009</startdate>
        <starttime>10:00:00</starttime>
        <endtime>12:00:00</endtime>
        <enddate>13.12.2009</endddate>
        <projectid>22</projectid>
        <operationid>33</operationid>
        <state>3</state>
        <description>kuvaus työmääräyksestä</description>
 </work>
 <work>
        <id>124</id>
        <startdate>12.11.2009</startdate>
        <starttime>11:00:00</starttime>
        <endtime>16:00:00</endtime>
        <enddate>13.11.2009</endddate>
        <projectid>22</projectid>
        <operationid>32</operationid>
        <state>2</state>
        <description>kuvaus työmääräyksestä 2</description>
  </work>

</works>
```

Vastauksessa on aina kyseisen henkilön työmääräykset tai vaihtoehtoisesti koko projektin työmääräykset.

Palvelimen täytyy myös palauttaa projekti- ja operaatiotiedot, sillä työmääräyksissä voi olla sellaisia projekti id:tä ja operation id:tä, joita puhelimessa ei välttämättä ole.

5.5 Getoperations tapahtuma

Kun action getoperations, palvelin palauttaa listan operaatioista esimerkin mukaisena xml:nä

5.5.1 Pyyntösanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<request>
        <username>matti</username>
        <password>salasana</password>
        <action>getoperations</action>
</request>
```

5.5.2 Vastesanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<operations>
 <operation>
        <id>342</id>
        <code>AEI</code>
```



```
        <name>Suunnittelu</name>
  </operation>
  <operation>
        <id>33</id>
        <code>AER</code>
        <name>Dokumentointi</name>
  </operation>
</operations>
```

Palvelin palauttaa projektiin liittyvät operaatiot

5.6 getall tapahtuma

Kun action getall, niin palvelin hakee kaikki tiedot ja palauttaa ne yhdessä xml:ssä, kuten esimerkissä.

5.6.1 Pyyntösanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<request>
        <username>matti</username>
        <password>salasana</password>
        <action>getall</action>
</request>
```

5.6.2 Vastesanoma

```
<?xml version=”1.0” encoding=”UTF-8”?>
<projects>
        <project><id>123</id><code>123</code><name>Projekti123</name></project>
        <project><id>124</id><code>124</code><name>Projekti124</name></project>
        <project><id>125</id><code>125</code><name>Projekti125</name></project>
</projects>
<products>
<product>
        <id>123</id>
        <code>123</code>
        <name>Tuote1</name>
        <amount>5</amount>
</product>
<product>
        <id>124</id>
        <code>124</code>
        <name>Tuote2</name>
        <amount>2</amount>
</product>
</products>
<works>
  <work>
        <id>123</id>
        <startdate>12.12.2009</startdate>
        <starttime>10:00:00</starttime>
```



```
        <endtime>12:00:00</endtime>
        <enddate>13.12.2009</endddate>
        <projectid>22</projectid>
        <operationid>33</operationid>
        <state>3</state>
        <description>kuvaus työmääräyksestä</description>
  </work>
<works>
<operations>
  <operation>
        <id>342</id>
        <code>AEI</code>
        <name>Suunnittelu</name>
  </operation>
</operations>
```

5.7 Mobiilipyynnön datakenttä

Kun XML:n action kenttä on setwork voi XML-tiedostolla olla datakenttä, joka sisältää tuntikirjauksia, ja työmääräysten aloitus/lopetustietoja. Esimerkki 1. lähetetään työmääräyksestä tilan muutos, esimerkiksi työ aloitettu tai työ lopetettu. Timestamp-tagi on puhelimesta sillä hetkellä otettu aikaleima, kun työtä on muutettu.

5.7.1 Pyyntösanoma

Esim 1.

```
…
<action>setwork</action>
<data>
<work>
        <id>22</id>
        <state>2</state>
        <timestamp>123456778</timestamp>
</work>
</data>
…
```

Esim 2.

```
…
<action>setwork</action>
<data>
        <work>
                <date>12.12.2009</date>
                <hours>5.5</hours>
                <projectid>34</projectid>
                <operationid>345</operationid>
                <productid>23</productid>
                <productid>34</productid>
                <productid>12</productid>
        </work>
```


</data>
…

Productid-kenttiä voi olla useampia, sillä tuotteita voi olla tuntikirjaukseen liitetty useampia.

5.7.2 Vastesanoma

Palvelin vastaa esimerkin mukaisen xml:n jos lähetys onnistui.

<?xml version=”1.0” encoding=”UTF-8”?>
<error>
<errorcode>-1</errorcode>
</error>

5.8 Virhetilanteet

Palvelimen virhetilanteessa lähetetään esimerkin mukainen xml.

<?xml version=”1.0” encoding=”UTF-8”?>
<error>
<errorcode>34</errorcode>
</error>

