STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN ALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

MOBIILISOVELLUSKEHYS

Työn tekijä: Anssi Luomaranta
Työn valvoja: Simo Silander
Työn ohjaaja: Timo Salmi

Työ hyväksytty: __. __. 2007

Simo Silander
lehtori

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ALKULAUSE

Tämä insinöörityö tehtiin A4SP Technologies Oy:lle keväällä 2007. Haluan kiittää A4SP:tä mielenkiintoisesta insinöörityön aiheesta ja projektista sekä Helsingin ammattikorkeakoulun opettajakuntaa, erityisesti tietotekniikan opettajia.

Helsingissä 11.4.2007

Anssi Luomaranta

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Anssi Luomaranta	
Työn nimi: Mobiilisovelluskehys	
Päivämäärä: 11.4.2007	Sivumäärä: 46 s. + 2 liitettä
Koulutusohjelma: Tietotekniikka	Suuntautumisvaihtoehto: Ohjelmistotekniikka
Työn valvoja: lehtori Simo Silander Työn ohjaaja: TJ Timo Salmi	
Tämän insinöörityön tarkoituksena oli kehittää mahdollisimman helposti laajennettava sovelluskehys A4SP Technologies Oy:n käyttöön osana ohjelmointiprojektia, johon liittyy useasta osasta koostuvan mobiilisovelluskokonaisuuden kehittäminen. Tarkoituksena oli saada aikaan sovelluskehys, jota kaikki sovellukset käyttäisivät runkonaan ja joka olisi riittävän monipuolinen, jotta sille voitaisiin perustaa myös mahdolliset tulevat sovellukset. Työssä käydään läpi projektin suunnittelulähtökohtia, lopputuloksena aikaansaatu sovelluskehys siinä käytettyine suunnitelumalleineen sekä käytännön kokemuksia oikeista mobiililaitteista. Erityisesti työhön liittyvät Bluetooth API (JSR 82), Mobile Media API (JSR 135), verkkoyhteydet mobiililaitteista, käyttöliittymän suunnittelu sekä RMS. Lopputuloksena saatiin aikaan määrittelyt täyttävä sovelluskehys, joka soveltuu Bluetooth-yhteyttä käyttäviin sovelluksiin joiden käyttöliittymän voidaan katsoa koostuvat erillisistä komponenteista. Kehys tukee sovelluksia, jotka lähettävät dataa käyttäen HTTP-protokollaa sekä sovelluksia joiden tarvitsee lukea ja tallentaa erityyppistä dataa käyttäen RMS:ää. Lisäksi kehys tarjoaa MMAPIn käyttöön pohjautuvan äänentoistoratkaisun.	
Avainsanat: sovelluskehys, Java, J2ME, Bluetooth, JSR 82, MMAPI, JSR 135, suunnittelumallit	

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Anssi Luomaranta

Title: Application Framework for Mobile Software

Date: April 11th 2007 | Number of pages: 46 + 2 attachments

Department: IT | Study Programme: Software Engineering

Instructor: Simo Silander, M.Sc.

Supervisor: Timo Salmi, CEO

The purpose of this graduate study was to create an extensible programming framework as a part of a mobile software project. The framework would form the base of the application under development as well as a base for possible future applications. The framework design process would use object oriented design patterns where applicable. The graduate study describes the initial requirements that are placed on the framework. The requirements were formed based on the design goals of the first application using the framework. The key elements of the implementation are explained along with usage examples and class diagrams. Areas particularly covered are Bluetooth API, network connections from mobile devices, graphic user interface design and rms.

The resulting framework meets the requirements placed on it. It supports applications using Bluetooth to search and connect to nearby devices. It also provides applications with means to use RMS to read and write data of different types and means to use HTTP to read and write data using network connection. In addition, it also provides interfaces and abstract classes for a user interface as well as several concrete user interface component classes and an audio playback system based on MMAPI.

Keywords: Application Framework, Java, J2ME, Bluetooth, JSR 82, MMAPI, JSR 135, design patterns

SISÄLLYS

SYMBOLILUETTELO

API	*Application Programming Interface.* Ohjelmointirajapinta.
CLDC	*Connected Limited Device Configuration.* J2ME-konfiguraatio laitteille joilla on hyvin rajalliset resurssit.
HTTP	*Hypertext Transfer Protocol.* Kommunikointiprotokolla.
J2ME	*Java 2 Micro Edition.* Javan pienille langattomille laitteille suunnattu versio. Myös Java ME.
JSR	*Java Specifications Request.*
MIDP	*Mobile Information Device Profile*. Joukko ohjelmointirajapintoja mobiililaitteille.
MMAPI	*Mobile Media Api.* Laajentaa J2ME alustaa tarjoamalla audio-, video- yms. multimediatuen.
RMS	*Record Management System.* MIDP:n osa joka tarjoaa tavan tallentaa sovelluskohtaista dataa.

1 JOHDANTO

Ohjelmistokehityksessä ilmenee usein, että ratkaistakseen jokin tietty sovelluskohtainen ongelma, on ohjelmoijan kirjoitettava suuri määrä koodia, joka ei kuitenkaan poikkea kahden eri ohjelman välillä paljoakaan. Sovellus voi kommunikoida esimerkiksi palvelimen kanssa hyvinkin erilaisia asioita, mutta ainoastaan datan sisältö ja sen tulkitseminen vaihtelevat. Sovelluskehyksellä tarkoitetaan luokka- tai kirjastokokoelmia, jotka muodostavat ohjelman rungon sisältäen yleisesti käytettyjä toimintoja. Sellaisenaan sovelluskehys ei yleensä tee mitään, vaan se vaatii toimiakseen sitä käyttävän sovelluksen. Sovelluskehys säästää aikaa sisältämällä paljon uudelleenkäytettävää koodia, jolloin ohjelmoijan ei tarvitse toteuttaa samoja asioita uudelleen eri sovelluksiin.

Myös suunnittelutyössä ilmenee samantyyppisiä ongelmia, joihin on kehitetty yleisiä ja ajan kuluessa hyväksi havaittuja ratkaisuja. Näitä suunnitteluratkaisuja kutsutaan suunnittelumalleiksi. Olio-ohjelmoinnin suunnittelumallit soveltuvat kukin erilaisiin tilanteisiin. Niiden käytössä hankalinta on usein valita oikea malli oikeaan tilanteeseen.

Tässä työssä pyrittiin muodostamaan sovelluskehys samaan aikaan mobiiliohjelmointiprojektin yhteydessä. Kehys tulisi tukemaan ohjelmoitavan sovelluksen toimintoja ja siitä olisi hyötyä myös mahdollisissa tulevissa sovelluksissa. Kehys tulisi tarjoamaan palvelut Bluetooth-laitteiden hakemiseen ja yhteyden ylläpitoon, valmiita ja laajennettavia käyttöliittymärakenteita, menetelmät vapaavalintaisen datan tallennukseen RMS-järjestelmän avulla datan tyypillä eroteltuna sekä valmiin ratkaisun datan lukemiseksi ja lähettämiseksi internetissä sijaitsevalle palvelimelle. Lisäksi tarjottaisiin järjestelmä äänitiedostojen toistamiseen. Suunnittelussa pyrittiin käyttämään avuksi suunnittelumalleja ratkaisemaan ongelmia.

2 A4SP-SOVELLUSKEHYKSEN MÄÄRITTELY

2.1 Lähtökohdat

Suunnittelun lähtökohtana oli että A4SP Technologies Oy:llä on kehitteillä anturi-instrumentti, joka kommunikoi mobiililaitteen kanssa Bluetooth-yhteyden kautta. Mobiililaitteessa olevat sovellukset tulkitsevat instrumentista tulevaa dataa ja reagoivat sovelluksesta riippuen eri tavoin. Itse instrumentti on yhä kehitteillä, joten kehitystyötä tuli pystyä tekemään ilman toimivaa anturia, joten osaksi projektia piti hankkia tai kehittää sovellus, jolla voitiin lähettää dataa toisesta mobiililaitteesta samassa muodossa kuin valmis anturi tulisi lähettämään. Koska laite on suunnattu kuluttajille, katsottiin, että mobiililaitteen ominaisuuksien tulisi asettaa mahdollisimman vähän rajoituksia hankinnalle. Tässä yhteydessä suosittiin J2ME-alustaa Symbian C++ -ohjelman sijaan sille tarjolla olevan laajan laitevalikoiman vuoksi. Kaikilla sovelluksilla tulee olemaan toteutukseltaan samanlaista toiminnallisuutta. Nämä yhteiset toiminnot tulisivat muodostamaan pohjan sovelluksien käyttämälle kehykselle.

J2ME-alustan vaatimukset

Koska kommunikointi instrumentin kanssa tapahtuu Bluetooth-yhteydellä, oli selvää, että mobiililaitteessa tulee olla Bluetooth API (JSR 82). Lisäksi selvää oli, että ohjelmien tuli voida antaa palautetta käyttäjälle äänien avulla, joiden tulisi olla kehittyneempiä kuin yksittäiset äänet. Tämä vaatii mobiililaitteelta MMAPIa eli Mobile Media APIa (JSR 135). Tämän vaatimuksen olisi hyvä olla optionaalinen, jotta sovellus toimisi myös laitteissa, joissa MMAPIa ei ole. Muita sovelluksissa olevia ominaisuuksia olisivat mahdollisuus kommunikoida internetissä olevan palvelimen kanssa sekä tietojen tallentaminen mobiililaitteeseen. Näiden pohjalta mobiililaitteen J2ME-alustalta vaadittavat ominaisuudet olisivat MIDP2 / CLDC 1.1, Bluetooth API sekä MMAPI (äänille).

2.2 Työkalut

Työkaluissa pyrittiin hyödyntämään mahdollisimman paljon ilmaistyökaluja. Ohjelmistokehitystä tehtiin Eclipse-kehittimen versiolla 3.2, johon oli lisäksi asennettu EclipseME-plugin versio 1.6.2. EclipseME on Eclipsen mobiiliohjelmointiin suunnattu laajennus. Se tarjoaa joukon mobiiliohjelmien

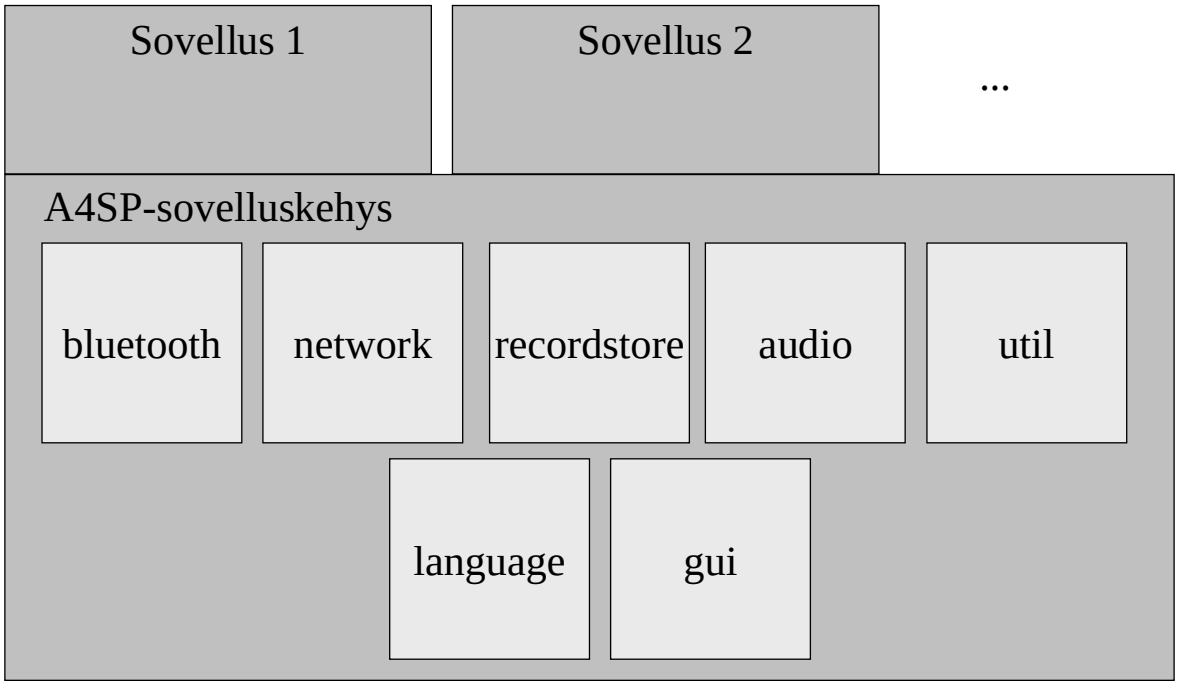
ohjelmointia ja testausta helpottavia työkaluja. Ohjelman testaukseen emulaattorilla käytettiin Sun Wireless Toolkit -versiota 2.2. Lisäksi viimeistelyssä käytettiin Proguard-ohjelmiston versiota 3.5. Työn dokumentoinnissa käytettiin luokka- ja sekvenssikaavioiden tekoon Visual Paradigm -mallinnusohjelmistoa.

Lisäksi oli tärkeää, että ohjelmistokehityksen aikana olisi käytössä myös oikea mobiililaite. Tähän tarkoitukseen valittiin Nokia 5500 -matkapuhelin, jossa on ohjelmiston vaatimat ominaisuudet (MIDP2 / CLDC 1.1). Matkapuhelimen katsottiin myös vastaavan ulkoasultaan yleisiä puhelinmalleja ja sisältävän monipuolisesti ominaisuuksia myös jatkokehityksen kannalta.

2.3 Ohjelmiston arkkitehtuuri

Laadittujen määrittelyjen pohjalta jaoteltiin sovellus osiin ja määriteltiin osien toiminnallisuus. Näiden osien pohjalta rakennettaisiin sovelluskehys, joka sisältää uudelleenkäytettäviä luokkia ja rajapintoja, joita voitaisiin käyttää myös sovellusta laajennettaessa tai kokonaan uusia sovelluksia kehitettäessä.

Kuvassa 1 on esitetty sovellusarkkitehtuuri yleisellä tasolla. Sovelluskehys sisältää pakkaukset bluetooth, network, recordstore, audio, util, language ja gui. Näistä bluetooth-pakkaus sisältää Bluetooth-toiminnallisuuden. Network sisältää luokkia verkkoyhteyden käyttöön, recordstore sisältää rms-toiminnallisuuden, audio äänentoiston ja gui käyttöliittymäkomponentit. Util sisältää tarpeellisia yleishyödyllisiä metodeja mm. matematiikkaan liittyen. Language sisältää lokalisoitaessa käännettäviä merkkijonoja. Aluksi kehystä hyödyntäviä sovelluksia olisi vain yksi, joskin se käyttää kehyksen tarjoamia toimintoja useissa eri tilanteissa. Lisäksi tässä työssä esitetyt sovelluskehyksen käyttöesimerkit muodostavat yhdistettynä yksinkertaisen esimerkkisovelluksen (luku 3).



Kuva 1: Arkkitehtuuri

Seuraavassa on eritelty kehyksen toiminnot:

Bluetooth-laitehaku

Laitehaun tehtävänä on hakea ympäristöstä Bluetooth-laitteet, jotka toimivat yhdessä sovelluksen kanssa. Lisäksi laitehaun on tarjottava käyttöliittymä, jossa käyttäjä valitsee laitteen, johon yhteys muodostetaan. Laitehaku on osa bluetooth-pakkausta.

Bluetooth-yhteys

Bluetooth-yhteyden muodostaminen sekä datan lähetys ja vastaanotto ovat toinen osa bluetooth-pakkaukselta vaadittavaa toiminnallisuutta. Lisäksi osan tulisi myös tarjota tapoja virhetilanteiden käsittelyyn mm. yhteyden automaattinen muodostaminen uudelleen. Komponentin ei tule hoitaa itse datan käsittelyä, sillä se saattaa vaihdella.

Äänentoisto

Äänentoistokomponentin tehtävänä on hoitaa MMAPIn avulla toteutettua äänentoistoa. Tarkennettuna sovelluksen pitää voida toistaa erillisistä äänitiedostoista koostettua puhepalautetta sekä lisäksi ääniefektejä. Lisäksi

äänentoiston tulisi toimia siten, että MMAPIn puuttuminen tai rajoitettu tuki äänitiedostotyypeille aiheuttaisi mahdollisimman vähän ongelmia. Äänentoiston toteutus tapahtuu audio-paketin luokkien ja rajapintojen avulla.

RMS-käsittely

Sovelluksien on voitava tallentaa tietojaan laitteen muistiin. RMS-käsittelyn tehtävänä on hoitaa datan lukeminen ja kirjoittaminen RMS-järjestelmään. Lisäksi järjestelmän olisi hyvä tarjota mahdollisuus käsitellä dataa sen tyypin mukaan siten, että esimerkiksi merkkijonojen ja kuvien lukeminen RMS:stä onnistuu helposti. Nämä toiminnot sijoitettiin osaksi recordstore-pakkausta.

Verkkoyhteys

Verkkoyhteysosion tehtävänä on hoitaa datan lähettäminen ja vastaanottaminen verkon yli palvelimelle käyttäen HTTP GET- ja POST-pyyntöjä. Komponentti tarjoaa tavan datan lähettämiseen ja lukemiseen ottamatta kantaa siihen, miten dataa käsitellään. Verkkoyhteyden käyttöön liittyvät luokat muodostavat network-pakkauksen.

Käyttöliittymä

Määrittelyjen perusteella ohjelman käyttöliittymään tulisi kuulumaan seuraavanlaisia erityyppisiä näkymiä:

- graafinen listanäkymä
- useista sivuista koostuva ohjenäkymä
- ns. latausruutu
- kullekin sovellusosiolle räätälöity oma näkymä.

Käyttöliittymässä tulisi olemaan erilaisia tapoja antaa informaatiota käyttäjälle mm. erilaisien popup-ikkunoiden ja mittareiden muodossa. Näitä komponentteja käytettäisiin useissa yhteyksissä, joten kehys tarjoaisi niille valmiit ratkaisut. Käyttöliittymän toimintoihin liittyvät luokat sijoittuvat gui-pakkaukseen.

2.4 Muut ohjelmiston ominaisuudet

Lokalisointi

Toiminnallisuuden lisäksi ohjelmiston suunnittelussa oli otettava huomioon lokalisointi. Ohjelman toteutus on sellainen, että käytössä on kulloinkin vain yksi kieli. Ohjelma on kuitenkin voitava kääntää muille kielille mahdollisimman pienellä vaivalla.

Palvelin

Projektiin liittyvä mobiiliohjelmiston kanssa toimiva palvelinohjelmisto jää aluksi toiminnallisuudeltaan yksinkertaiseksi, mutta sen pitää kuitenkin kyetä vastaamaan ohjelmiston lähettämiin HTTP-pyyntöihin, tallentamaan tietoa tietokantaan ja tulostamaan tietoja www-sivulla. Palvelinratkaisun on hyvä perustua ilmaisohjelmistoihin, joten käytettäviksi teknologioiksi valittiin Tomcat Servlet-alustan versio 5.5 ja mySQL-tietokanta. Itse palvelimen rakennetta ei käsitellä tämän yksityiskohtaisemmin tässä insinöörityössä.

3 ESIMERKKISOVELLUS

Sovelluskehyksen toimintoja havainnollistetaan tässä työssä yksinkertaisen esimerkkisovelluksen avulla, joka mittaa käyttäjän reaktionopeutta Bluetooth-yhteydellä lähetettyyn viestiin. Käyttäjä yrittää painaa nappia mahdollisimman nopeasti. Paras reaktioaika tallentuu, ja se myös toistetaan puheena. Esimerkkisovelluksen käyttöliittymä on esitetty kuvassa 2. Se koostuu menurakenteesta ja kolmesta eri mittarista, joista vasemmassa näkyy reaktioaika, oikeanpuoleisessa ennätys ja neulamittarissa, kuinka kaukana reaktioaika oli ennätyksestä. Lisäksi käyttöliittymän alareunassa on tekstikehys, jossa voidaan näyttää viestejä käyttäjälle. Sovellusta käynnistettäessä näytetään latauskuva. Käyttäjälle tarjotaan myös mahdollisuus lähettää ennätyksensä palvelimelle valitsemalla lähetystoiminto valikosta. Esimerkkisovelluksen luokkien koodi on esitetty kokonaisuudessaan liitteessä 1.

Kuva 2: Esimerkkisovelluksen käyttöliittymä

4 A4SP-SOVELLUSKEHYKSEN SUUNNITTELU, TOTEUTUS JA KÄYTTÖ

4.1 Suunnittelumallit

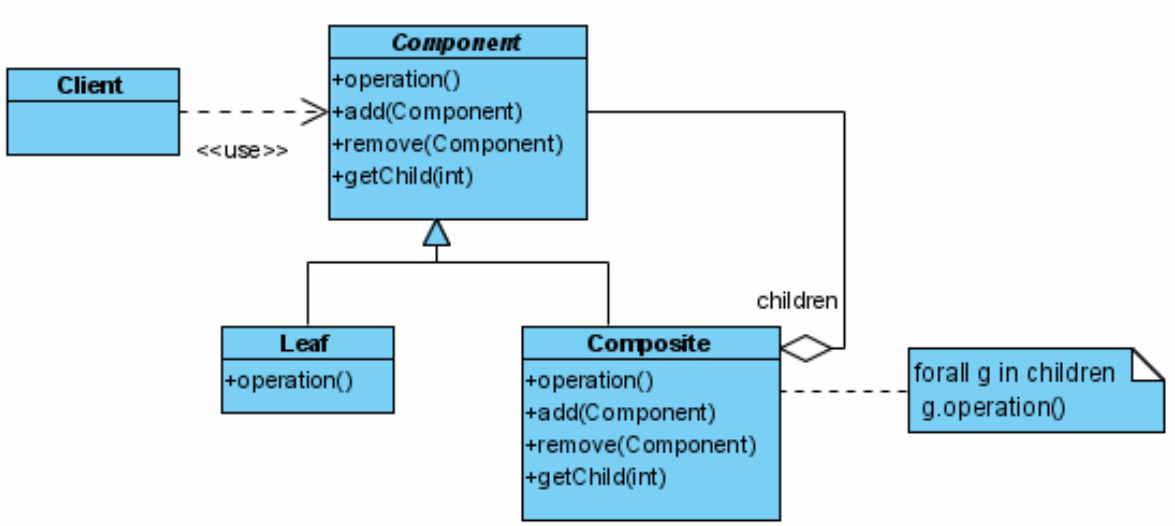
Suunnittelumallit tarjoavat valmiita ja toimiviksi havaittuja ratkaisuja erilaisiin ongelmiin. Teoksessa Olio-ohjelmointi suunnittelumallit (Design Patterns) [1] on esitetty 23 erilaisiin tilanteisiin soveltuvaa suunnittelumallia. Käyttötavasta riippuen suunnittelumallit voidaan erotella luontimalleihin, rakennemalleihin ja käyttäytymismalleihin. Luontimalleja käytetään rakentamaan järjestelmiä, jotka ovat riippumattomia olioiden luonti-, koostamis- tai esitystavasta. Rakennemalleissa yhdistelemällä luokkia ja olioita muodostetaan suurempia rakenteita. Käyttäytymismallit kuvaavat rakenteen lisäksi myös olioiden vuorovaikutussuhteita. Luvussa on esitelty lyhyesti sovelluskehyksen eri osioissa hyödynnetyt suunnittelumallit.

4.1.1 Rekursiokooste- eli Composite-malli

Composite-mallin ideana on, että yksittäisiä olioita ja oliokoosteita voidaan käsitellä samalla tavalla. Se soveltuu erityisesti grafiikan piirtämiseen tilanteissa, joissa isompi kokonaisuus koostuu useista osista.

Composite-mallissa määritellään abstrakti luokka, joka esittää sekä primitiiviolioita että koosteolioita. Tämä määrittelee primitiiviolioihin kohdistuvat toiminnot. Sovellus käsittelee koosterakenteita luokan määrittelemän rajapinnan kautta. Pyynnön mennessä primitiivioliolle se käsitellään suoraan. Koosteolio tyypillisesti delegoi pyynnön lapsilleen.

Composite-mallin rakenne on esitetty kuvassa 3. Rajapinta Component määrittelee yhteiset toiminnot. Komposiittiolio Composite koostuu Component-rajapintaa toteuttavista komponenteista, jotka voivat siis myös itse olla koosteolioita. Asiakassovellus käyttää olioita rajapinnan kautta.

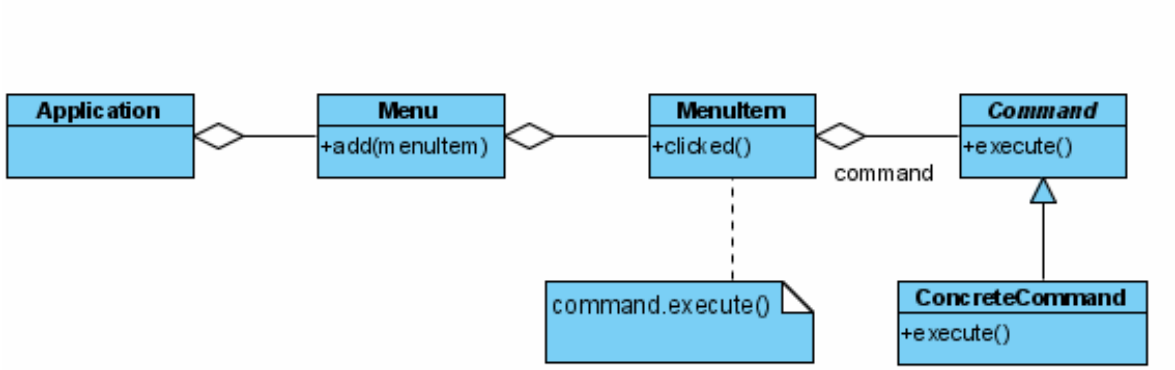


Kuva 3: Rekursiokoosteen luokkakaavio [1. s. 164]

4.1.2 Komento- eli Command-malli

Command-suunnittelumalli on käyttäytymismalli, jossa kapseloidaan pyyntö olioksi. Ideana on erottaa operaation käynnistävä olio siitä, miten operaatio suoritetaan. Tämä mahdollistaa käyttöliittymän joustavan suunnittelun. Esimerkiksi menurakenteessa se, mitä tapahtuu valittaessa jokin menun alkio riippuu sovelluksesta. Command-malli mahdollistaa myös komentojen ketjuttamisen ja eteenpäin välittämisen.

Kuvassa 4 on esitetty Command-mallin luokkakaavio. *Menu*-luokka on valikko, johon voi lisätä *MenuItem*-luokan jäseniä. Jokainen *MenuItem* sen lisäksi sisältää *Command*-rajapintaa toteuttavan käskyn, joka ajetaan valittaessa *MenuItem*. *MenuItem* ei tiedä, mikä *Command*-aliluokka tulee ajetuksi.



Kuva 4: Command-mallin luokkakaavio [1. s. 233]

4.1.3 Tehdasmetodi- eli Factory method -malli

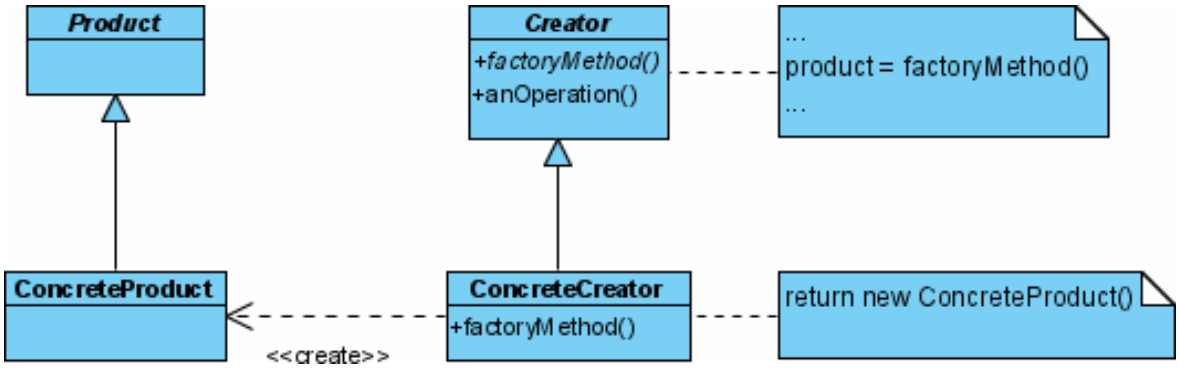
Tehdasmetodi on luontimalli, jossa määritellään rajapinta olion luomiseksi, mutta aliluokka voi päättää, mikä luokka luodaan.

Tehdasmetodin käyttö soveltuu seuraaviin tilanteisiin:

- Luokka ei tiedä, mistä luokasta sen on luotava ilmentymiä.
- Luokka haluaa aliluokkiensa määrittelevän luotavat oliot.
- Luokat delegoivat vastuun jollekin useista avustavista aliluokista ja tieto siitä, mille aliluokalle delegoidaan, halutaan paikallistaa tiettyyn kohtaan.

[1, s. 108]

Kuvassa 5 on esitetty tehdasmetodin luokkakaavio. Abstrakti Creator-luokka määrittelee FactoryMethodin, joka luo Product-rajapinnan mukaisen tuotteen. Variaationa Creator-luokka voi olla myös konkreettinen ja tarjota oletustoteutuksen.

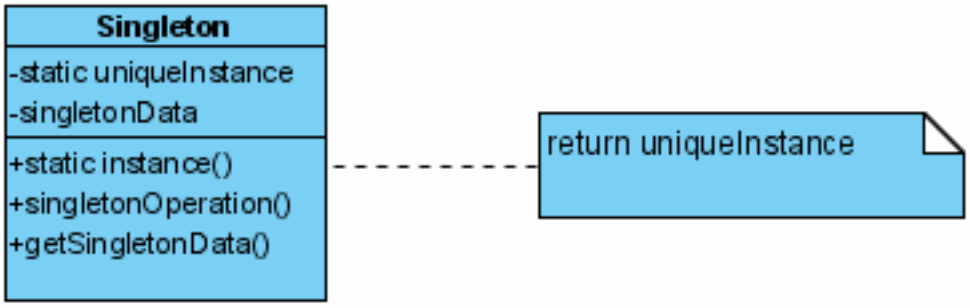


Kuva 5: Tehdasmetodin luokkakaavio [1. s. 107]

4.1.4 Ainokainen- eli Singleton-malli

Singleton-luontimalli varmistaa, että luokasta on aina olemassa vain yksi ilmentymä. Malli tarjoaa tavan päästä käsiksi tähän ilmentymään mistä tahansa. Singleton määrittelee Instance-operaation, jolla ilmentymään pääsee käsiksi. Singleton voi myös olla itse vastuussa ilmentymänsä luomisesta. Tällöin ilmentymän luonti tapahtuu Instance-operaatiossa, jos ilmentymää ei vielä ole olemassa. Tekemällä singletonin konstruktorista privaatti voidaan estää luokan ilmentymien luonti luokan ulkopuolella.

Kuvassa 6 on esitetty Singletonin luokkakaavio. Instance-operaatio on staattinen samoin kuin uniqueInstance-muuttuja. SingletonOperation ja getSingletonData ovat metodeja singletonin käsittelemiseksi.



Kuva 6: Singleton-mallin luokkakaavio. [1. s. 127]

4.2 Bluetooth-pakkaus

4.2.1 Bluetooth-pakkauksen toteutus

Bluetooth-yhteyskokonaisuus koostuu kahdesta erillisestä osasta: Bluetooth-laitehausta, jonka tehtävänä on hakea lähistöllä olevista Bluetooth-laitteista ne, joiden on tarkoitus toimia ohjelman kanssa, sekä itse yhteyskomponentista, joka hoitaa yhteyttä yhteen Bluetooth-laitteeseen. Bluetooth-yhteyden toteutus tapahtuu Bluetooth APIn (JSR 82) avulla. Bluetooth API on standardoitu laitteistoriippumaton tapa käyttää Bluetooth-laitteita Java-kielen avulla.

Laitehaku ja palveluhaku

Ennen Bluetooth-yhteyden muodostamista on löydettävä lähistöllä olevat Bluetooth-laitteet, jotka lisäksi pystyvät kommunikoimaan kyseessä olevan ohjelman kanssa. Sekä laite- että palveluhaku suoritetaan sovelluskehyksessä olevan *ServiceDiscovery*–luokan avulla. Luokka toteuttaa Bluetooth APIssa määriteltyä *DiscoveryListener*-rajapintaa. Rajapinta määrittelee *servicesDiscovered*-, *serviceSearchCompleted*-, *deviceDiscovered*- ja *inquiryCompleted*-metodit. Luokka on toteutettu säikeenä, joka käynnistyessään aloittaa laitehaun, etsii löydetyistä laitteista ne, joissa on haluttu palvelu ja näyttää lopuksi listan löydetyistä laitteista ja tarjoaa mahdollisuuden muodostaa yhteyden tai useita sekä mahdollisuuden uusia haku.

ServiceDiscovery käyttää haussa apuna kahta sisäistä tietorakennetta. Löydetyt laitteet lisätään Vector-rakenteeseen *deviceList deviceDiscovered*-metodin yhteydessä ja haettua palvelua vastaavat *ServiceRecord*-oliot tallennetaan *HashTable*en. Kun laitehaku päättyy *inquiryCompleted*-metodiin, tarkastetaan ensin, tapahtuiko haussa virheitä. Jos ongelmia ei ollut ja ainakin yksi laite löytyi, käynnistyy palveluhaku löydetyille laitteille. *ServiceDiscovery* tarkastaa arvon bluetooth.sd.trans.max avulla, mikä on laitteen sallima palveluhakujen maksimimäärä ja käynnistää uusia hakuja ainoastaan tämän määrän. Jos myös palveluhaussa löytyy vastaava *ServiceRecord*, haetaan laitteen nimi. Vaihtoehtoisesti käytetään osoitetta, jos nimen haussa oli ongelmia. *HashTable*en tallennetaan *ServiceRecord* yhdistettynä nimeen. Jos haku kestää yli sallitun maksimiajan se keskeytetään ja tilanne käsitellään virheenä. Lopuksi *ServiceDiscovery*n sisäluokkana toteutettu *javax.microedition.lcdui.Form*-luokasta periytetty *DiscoveryScreen* näyttää *HashTable*n sisällön.

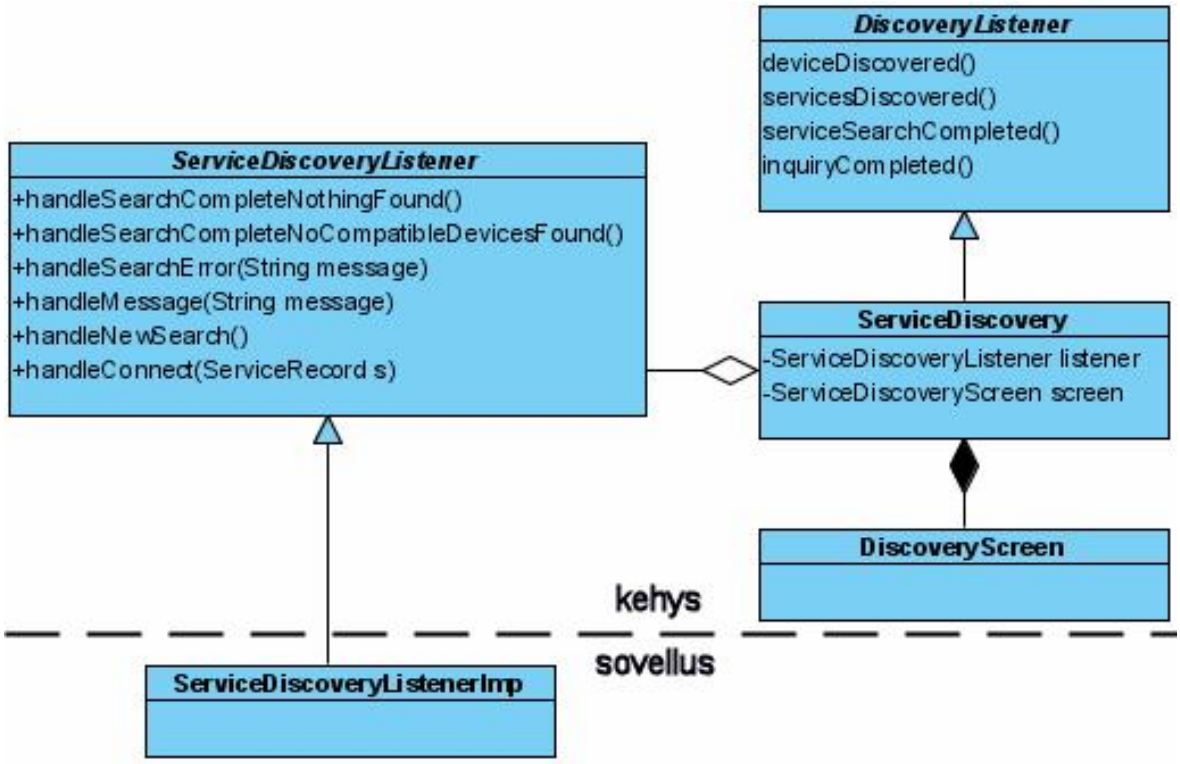
ServiceDiscovery-luokka kommunikoi sovellukselle bluetooth-paketissa määritellyn *ServiceDiscoveryListener*-rajapinnan avulla. Rajapinta määrittelee metodit:

- handleSearchCompleteNothingFound()
- handleSearchCompleteNoCompatibleDevicesFound()
- handleSearchError(String message)
- handleMessage(String message)

- handleNewSearch()
- handleConnect(ServiceRecord s).

Metodeja käytetään erilaisiin tilanteisiin reagoimiseksi. Toteutus jättää yhteyden muodostamisen sovellukselle *handleConnect*-metodin kautta, joka tarjoaa viitteen *ServiceRecord*–ilmentymään, jonka avulla yhteyden saa muodostettua. *ServiceDiscovery* lähettää lisäksi tietoa haun etenemisestä sovellukselle *handleMessage*-metodin avulla. Sovellus voi halutessaan näyttää tiedot käyttäjälle.

Kuvassa 7 on esitetty Bluetooth-hakujärjestelmän luokkakaavio. *ServiceDiscoveryListener*-rajapintaa toteuttavana sovelluskohtaisena luokkana on esitetty *ServiceDiscoveryListenerImp*-luokka.



Kuva 7: Bluetooth-hakujärjestelmän luokkakaavio

Listauksissa 1, 2 ja 3 on esitetty *ServiceDiscovery*-luokan keskeiset osat. Sisältöä on korvattu osin pseudokoodilla kohdissa, jotka eivät ole toiminnan kannalta keskeisiä. Listauksessa 1 on esitetty luokan määrittelyä ja konstruktori.

```
public class ServiceDiscovery extends Thread implements
                                        DiscoveryListener{

  private int maxSdCount;  //hakujen maksimimäärä.
  private DiscoveryScreen discovery;
  private Vector deviceList;  //laitelista
  //Löydetyt palvelut:
  private Hashtable matchingServiceRecords;
  private int pendingTransactions; //hakuja käynnissä.
  ...
  public ServiceDiscovery(ServiceDiscoveryListener l,
                          int minConnections,
                          int maxConnections,
                          UUID[] uuids,
                          ScreenDisplayer d)
                          throws BluetoothStateException{
    ...
    String property = LocalDevice.getProperty(
                     "bluetooth.sd.trans.max");
    maxSdCount = Integer.parseInt(property);
    ...
```

Listaus 1: ServiceDiscovery luokan attribuutit ja konstruktori

Listauksessa 2 on esitetty Bluetooth APIn *DiscoveryListener*-rajapinnan metodien toteutukset. Näissä siis *deviceDiscovered* lisää laitteen listalle myöhempää palveluhakua varten. Metodi *servicesDiscovered* hakee laitteen nimen ja lisää laitteen hakua vastaavien *ServiceRecord*ien tauluun yhdistettynä nimeensä. *InquiryCompleted*- ja *ServiceSearchCompleted*–metodit ohjaavat haun etenemistä *pendingTransactions*-, *searchInProgress*- ja *inquiryInProgress*-muuttujilla ja tarpeen vaatiessa keskeyttävät haun.

```
public void deviceDiscovered(RemoteDevice dev,
                             DeviceClass devClass) {
  deviceList.addElement(dev);
}

public void servicesDiscovered(int transId,
                               ServiceRecord[] rec) {
  String name;
  try {
     name = device.getFriendlyName(false); //yritetään
                                           //hakea nimi
  } catch (IOException e) {
    name = device.getBluetoothAddress();
  }
   ...
  matchingServiceRecords.put(name,record[0]);
}
```

```
public void serviceSearchCompleted(int transId,
                                        int respCode) {
  pendingTransactions--; //yksi käynnistetyistä hauista
                         //päättyi
  if (pendingTransactions == 0 &&
      deviceList.size() == 0) {
    discovery.populate();
    searchInProgress = false;
    näytä Form discovery;
  }
  else {
    listener.handleSearchComplete
            NoCompatibleDevicesFound();
  }
}

public void inquiryCompleted(int discType) {
  if(deviceList.size() == 0) {
    listener.handleSearchCompleteNothingFound();
    keskeyta haku;
  }
  else {
    inquiryInProgress = false; //tämä sallii palveluhaun
  }
}
```

Listaus 2: DiscoveryListener-toteutus

Listauksessa 3 on haun etenemistä ohjaava säikeen *run*-metodin toiminnan kannalta olennaiset osat sekä *DiscoveryScreen*-luokan keskeiset osat. Säie käynnistää haun, aloittaa palveluhaun laitehaun päätyttyä sekä tarpeen vaatiessa keskeyttää haun aikarajan umpeuduttua.

```
public void run() {
  if(!kaynnistaHaku()) {
    listener.handleSearchError(”Haku ei käynnisty”);
    running = false;
  }

  while(running) {
    listener.handleMessage(“Haku käynnissä -teksti.”)
    if (!inquiryInProgress) doServiceSearch();
    if (hakuaika umpeutuu) {
      peruuta kaikki haut;
      running = false;
    }
  }


private class DiscoveryScreen extends Form implements
                                        CommandListener{
...
public DiscoveryScreen() {
  luo ja alusta komponentit;
  populate();
}
```

```
public int populate() {
  tayta lista kayttaen HashTablea;
}

public void commandAction(Command cmd,
                          Displayable disp) {
  if(cmd==muodostaYhteysCommand) {
    listener.handleConnect(
           (ServiceRecord)matchingServiceRecords.get(
           choice.getString(i)));
  }
  else if(cmd==kaynnistaUusiHakuCommand) {
    listener.handleNewSearch();
  }
}
```

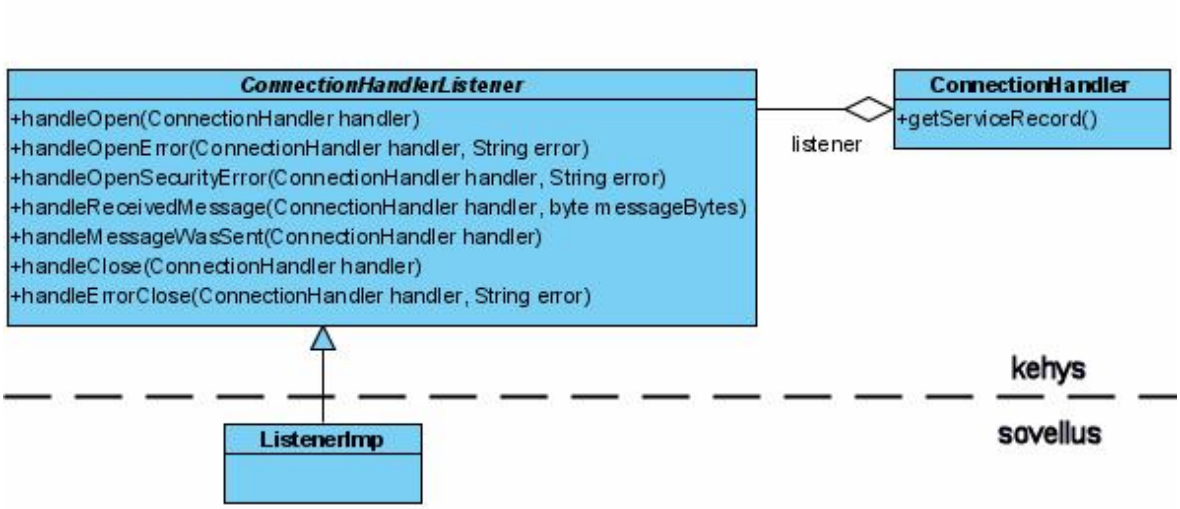
Listaus 3. ServiceDiscovery-luokan toteutusta.

Bluetooth-yhteys

Bluetooth-yhteys on toteutettu *ConnectionHandler*-luokan ja siihen liittyvän *ConnectionHandlerListener*-rajapinnan avulla. *ConnectionHandler* hoitaa itse yhteyden käsittelyn ja välittää olennaiset tapahtumat sovellukselle rajapinnan metodien avulla. Rajapinta määrittelee metodit:

- handleOpen – ilmoitus yhteyden avaamisesta
- handleOpenError – ilmoitus yhteyden avaamisen epäonnistumisesta
- handleReceivedMessage – vastaanotetun viestin ohjaus käsiteltäväksi
- handleMessageWasSent – viesti lähetetty
- handleClose – ilmoitus yhteyden sulkemisesta
- handleErrorClose – yhteys suljettiin virheen vuoksi.

Bluetooth-yhteydessä käytettävät luokat on esitetty kuvassa 8. Rajapinnan toteuttavana sovelluskohtaisena luokkana on *ListenerImp*.



Kuva 8: Bluetooth-yhteyden luokat

ConnectionHandler-luokka on toteutettu säikeenä, joka käynnistyessään pyrkii ensin avaamaan yhteyden. Yhteyden muodostumisen jälkeen se lukee sisään tulevaa dataa pakettipohjaisesti siten, että paketin pituuden oletetaan tulevan ensin. Tämän jälkeen luetaan pituuden ilmoittama määrä dataa ottamatta kantaa sen sisältöön. Data ohjataan sovellukselle *ConnectionHandlerListener*-rajapinnan määrittelemän *handleReceived-Message*-metodin avulla tavutaulukkona. Sovellus voi tällöin käsitellä dataa haluamallaan tavalla. Esimerkiksi yhteyden katketessa sovelluksessa toteutettu *ConnectionHandlerListener* saa siitä tiedon *handleErrorClose (ConnectionHandler handler, String errorMessage)* –metodin avulla, jolloin sovellus voi reagoida siihen, vaikka sulkemalla sovelluksen tai palaamalla alkutilaan ja näyttämällä käyttäjälle metodissa välittyvän virheilmoituksen.

ConnectionHandler-luokan määrittelyt ja konstruktori on esitetty listauksessa 4. Konstruktori ottaa parametreiksi *ConnectionHandlerListener*-toteutuksen, *ServiceRecord* luokan yhteyden muodostusta varten sekä yhteyteen halutun tietoturvatason *ServiceRecord*-luokan määrittelemänä kokonaislukuna.

```
public class ConnectionHandler extends Thread{

  private int requiredSecurity;
  private ServiceRecord serviceRecord;
  private ConnectionHandlerListener listener;
  private StreamConnection connection;
  private OutputStream out;
  private InputStream in;



  public ConnectionHandler(ConnectionHandlerListener
                           listener, ServiceRecord
                           serviceRecord,
                           int requiredSecurity) {
          this.requiredSecurity = requiredSecurity;
          this.serviceRecord = serviceRecord;
          this.listener = listener;
}
```

Listaus 4: ConnectionHandlerin määrittely ja konstruktori

ConnectionHandler-luokan *run*-metodin toteutuksen keskeiset osat on esitetty listauksessa 5. Alussa avataan yhteys sekä *input*- että *outputstream*, jonka jälkeen luetaan datan pituus sekä vastaava määrä dataa taulukkoon alustaen se tarpeen mukaan uudelleen. *Inputstream*in *read*-metodin palauttaessa –1 suljetaan yhteys ja kutsutaan *ConnectionHandlerListener*in sulkemismetodia.

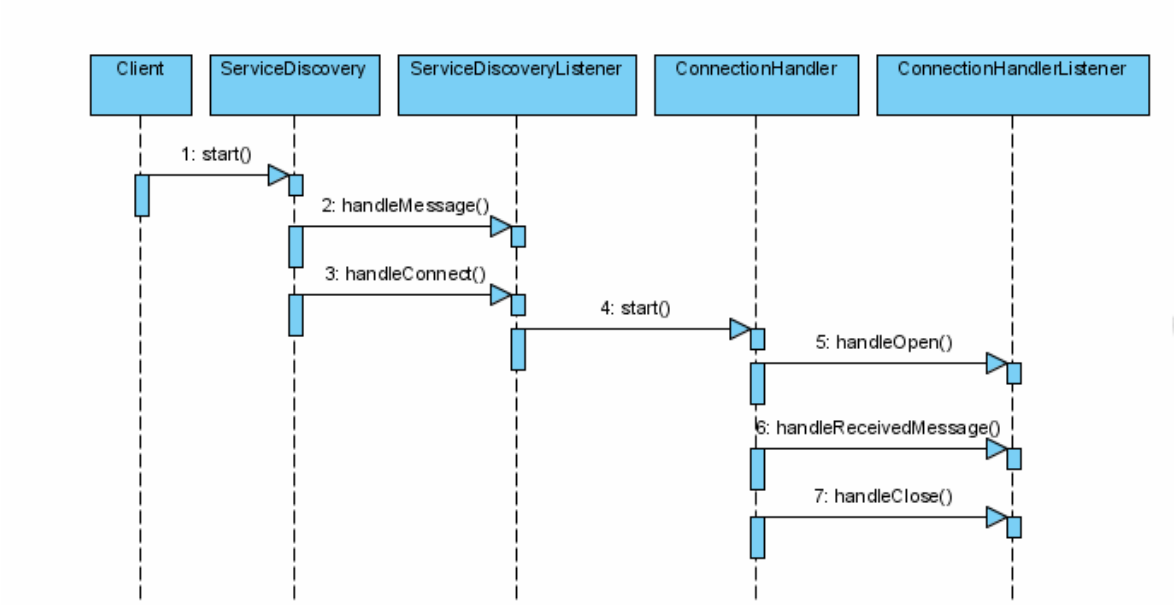
```
public void run() {
  boolean run = true;
  String url = null;
  int messageLengthBytes = 8;
  byte[] message = new byte[messageLengthBytes];
  try {
   url = serviceRecord.getConnectionURL(
                       requiredSecurity,
                       false);

   connection = (StreamConnection) Connector.open(url);
   in = connection.openInputStream();
   out = connection.openOutputStream();
   listener.handleOpen(this);

 }
 catch (IOException e) {
   listener.handleOpenError(this,e.toString());
   return;
 }
 catch (SecurityException e) {
   listener.handleOpenError(this,e.toString());
   return;
 }
```

```
 while (run) {
    messageLengthBytes = in.read();

    if (messageLengthBytes == -1) {
      listener.handleErrorClose(this,"Inputstream
                                      terminated.");
      closeConnection();
      return;
    }
    if (messageLengthBytes != message.length) {
      message = new byte[messageLengthBytes];
    }

    in.read(message,0,messageLengthBytes);
    listener.handleReceivedMessage(this,message);
  }

  listener.handleClose(this);
  closeConnection();
}
```

Listaus 5: ConnectionHandler run()-metodi

Kommunikointi

Kuvassa 9 on esitetty edellä esiteltyjen luokkien ja rajapintojen välinen kommunikointi tapauksessa, jossa *ServiceDiscoveryListener*-rajapinnan toteutuksesta käynnistetään myös *ConnectionHandler*illä käytettävä yhteys. Sovellus käynnistää laitehaun, joka lähettää viestejä haun etenemisestä *ServiceDiscoveryListener*-rajapinnan toteutukselle. Toteutus käynnistää *ConnectionHandler*in *handleConnect*-metodissa. Yhteyden auettua saadaan yksi viesti, jonka jälkeen yhteys sulkeutuu.



Kuva 9: Bluetooth-pakkauksen tapahtumasekvenssikaavio

4.2.2 Bluetooth-pakkauksen käyttö

Esimerkkisovelluksessa käytetään *ConnectionManager*-luokkaa toteuttamassa *ServiceDiscoveryListener*-rajapintaa. Laitehaku käynnistetään suoraan *MIDlet*-luokkaan tehdyllä *startBluetooth*-metodilla luomalla *ConnectionManager* ja käynnistämällä se listauksessa 6 esitetyllä tavalla. *ServiceDiscovery*n konstruktorille annetaan parametrina *ConnectionManager*-luokka, muodostettavien yhteyksien minimi- ja maksimimäärät, haettavat Bluetooth-UUID:t sekä *MIDlet*-luokka näkymän vaihtamista varten.

```
try {
  UUID uuid = new
       UUID("AAAAA5EE9F9146109085C2055CBBBB17",false);
  UUID[] uuids = {uuid};
  ServiceDiscovery disc = new ServiceDiscovery(
           getConnectionManager(), 1, 1, uuids, this);
  disc.start();
} catch (BluetoothStateException e) {
  screen.setMessage(e.toString());
}
```

Listaus 6: Esimerkkisovelluksen Bluetooth-haun käynnistys.

Esimerkkisovelluksessa *ConnectionHandlerListener*-rajapintaa toteuttaa edellisessä kohdassa käsitelty *ConnectionManager*-luokka. Bluetooth yhteys käynnistetään Bluetooth-haun yhteydessä käsitellyssä metodissa *handleConnect*. Toteutus on esitetty listauksessa 7.

```
ConnectionHandler conn;

public void handleConnect(ServiceRecord s) {
          OmaMidlet.getInstance().getScreen().show();
          conn = new ConnectionHandler(this, s,
                    ServiceRecord.NOAUTHENTICATE_NOENCRYPT);
          conn.start();
}
```

Listaus 7: Yhteyden käynnistäminen ConnectionHandler-luokan avulla

Esimerkkisovelluksen ei tarvitse ryhtyä erityisiin toimenpiteisiin virhetilanteissa, joten riittää, että *ConnectionManager*iin lisätään rajapinnan tyhjät toteutukset. Viestipaketin saapuessa tallennetaan saapumisaika ja päivitetään mittareita. Tämä tehdään sitä varten kirjoitetussa metodissa *MIDlet*-luokassa. Rajapinnan määrittelemän *handleReceivedMessage*-metodin toteutus esimerkkisovelluksessa on esitetty listauksessa 8. Jos sovelluksen olisi tarpeellista lukea datan sisältöä, sen voisi tehdä tässä metodissa.

```
public void handleReceivedMessage(ConnectionHandler
       handler, byte[] messageBytes) {
          OmaMidlet.getInstance().messageIn();
}
```

Listaus 8: Saapuneen viestin käsittely esimerkkisovelluksessa

4.2.3 Bluetooth-pakkauksen soveltuvuus ja resurssivaatimukset

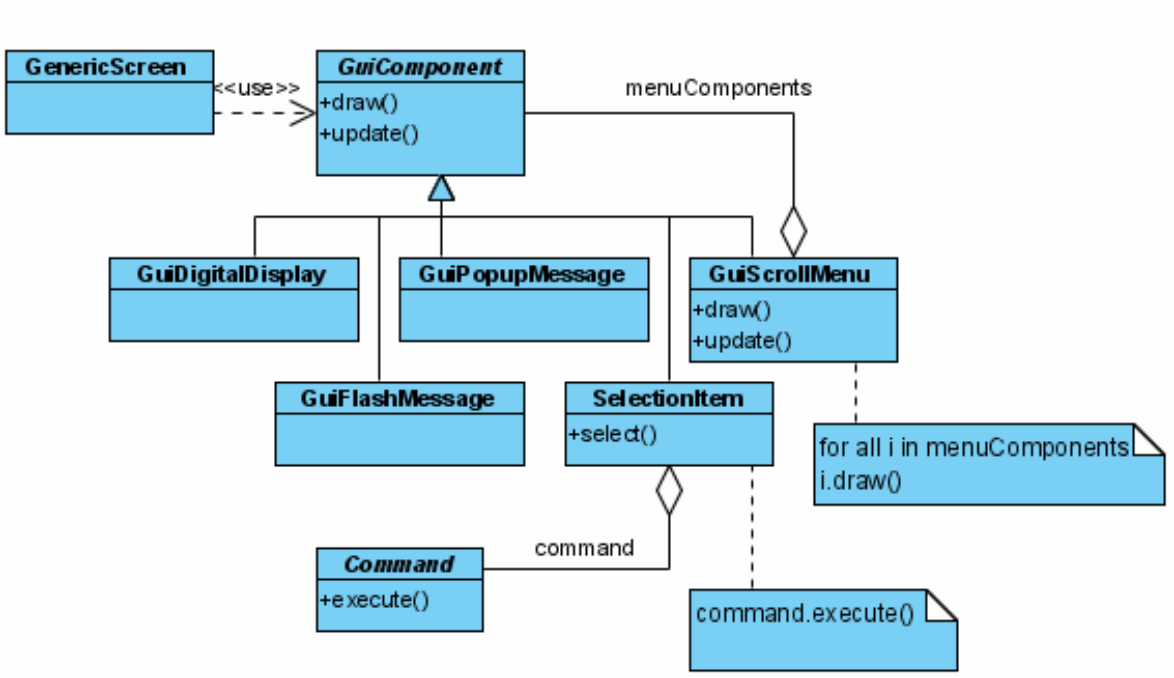
Laitehakutoteutus soveltuu Bluetooth-sovelluksiin, jossa käyttäjän halutaan valitsevan löydetyistä laitteista yksi tai useampia ja näihin halutaan tämän jälkeen muodostaa yhteys. Yhteystoteutus soveltuu vapaasti valittavan binääridatan lähettämiseen ja vastaanottamiseen, mutta ei tarjoa valmiita ratkaisuja datan käsittelemiseksi. Yhteystoteutus kuitenkin edellyttää, että laite, johon yhteys muodostetaan, odottaa yhteyttä. Lisäksi vaatimuksena pakkauksen käytölle on, että kohdelaitteessa on Bluetooth API.

4.3 Gui-pakkaus

4.3.1 Gui-pakkauksen toteutus

Käyttöliittymän visuaalinen ulkoasu haluttiin näyttävämmäksi ja monipuolisemmaksi kuin J2ME-peruskomponenteilla (esim. *Form*- ja *TextField*-luokat) on mahdollista toteuttaa. Tätä varten toteutettiin rakenne, jolla saatiin lisättyä erilaisia omia käyttöliittymäkomponentteja.

Käyttöliittymän luokkakaavio on esitetty kuvassa 10. Käyttöliittymän toteutuksessa on käytetty hyväksi Composite-mallia. Sovelluskehys määrittelee *GuiComponent*–rajapinnan, jota kaikki käyttöliittymässä käytettävät visuaaliset komponentit toteuttavat. Rajapinnassa on määritelty metodit *draw* – komponentin piirtämiseen – sekä *update* – komponentin tilan päivittämiseen.



Kuva 10: Gui-pakkauksen luokkakaavio

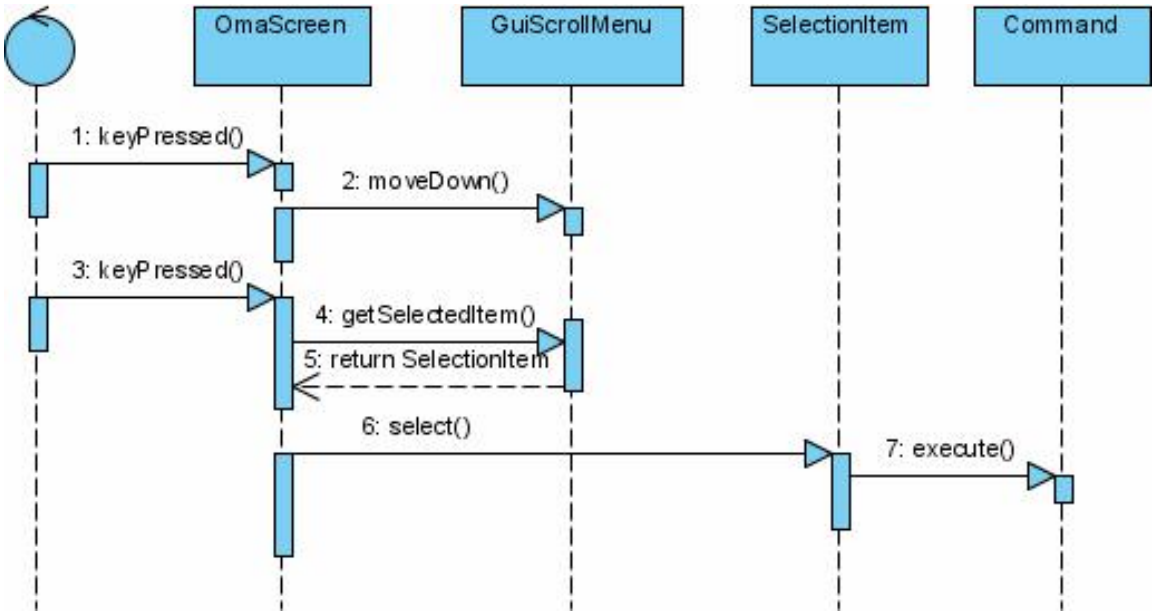
Erilaisia näkymiä voidaan koostaa luomalla erilaisia *GuiComponent*-toteutuksia ja piirtämällä ne halutussa järjestyksessä näytölle. Kaikki komponentit piirretään sisällöstään riippumatta samalla tavalla kutsumalla rajapinnan *draw*–metodia. Lisäksi sovelluskehys tarjoaa abstraktin luokan *GenericScreen*, jossa on määritelty sisäinen tietorakenne komponenttien

säilyttämiseksi ja metodit komponenttien lisäämiseksi ja poistamiseksi. Oletustoteutuksen piirtometodi piirtää lisäämisjärjestyksessä kaikki *GuiComponent*-jäsenet näytölle. Lisäksi luokka tarjoaa mahdollisuuden päivittää kaikki jäsenet kerralla. *GenericScreen*istä periytettyjä näkymiä voi myös helposti vaihtaa näytölle sen tarjoamaa *show*–metodia käyttäen.

GuiScrollMenu

Luvussa käsitellään tarkemmin sovelluskehyksessä olevaa valmista *GuiScrollMenu*–toteutusta. Luokan tehtävänä on tarjota valikkorakenne, jossa valikon komponentit voi sijoitella näytölle varsin vapaasti. Valikko pystyy myös vierittämään näkymää valitun elementin kohdalle sekä päivittämään elementin tilaa sen ollessa valittuna siten, että elementti voi esimerkiksi näyttää jonkun animaation tai sen ulkonäkö muuten muuttua. *GuiScrollMenu*-luokka sisältää sisäluokkana *SelectionItem*-luokan, joka toteuttaa myös *GuiComponent*-rajapintaa. Piirrettäessä *GuiScrollMenu* piirtää sisältämänsä *SelectionItem*-oliot ja toimii näin rekursiokoostemallin mukaisesti komposiittioliona. *SelectionItem* sisältää yhden valikkoelementin toteutuksen. Valikkoelementti sisältää *javax.microedition.lcdui.game.Sprite*-luokan, jota käytetään hyväksi animaation toteutukseen sekä sijainnin asettamiseen. Lisäksi valikkoelementti sisältää yhden *Command*-rajapinnan määrittelemän käskyolion, jossa on valikkoelementtiä valittaessa ajettava koodi. Valikkoelementtien toiminnallisuuden kapselointi käskyoloihin vastaa näin luvussa 4.1.2 käsiteltyä komento-suunnittelumallia ja mahdollistaa ulkoasun erottamisen toiminnoista.

Kuvassa 11 on esitetty *GuiScrollMenu*-luokkaa käyttävän sovelluksen tapahtumasekvenssikaavio tapauksessa, jossa näppäinpainalluksilla ohjataan menun toimintaa. *OmaScree*n on menun sisältävä *GenericScreen*-luokan aliluokka. Ensimmäinen painallus ohjataan siirtämään menuvalintaa yksi alaspäin. Seuraava painallus aktivoi *SelectionItem*-olion, joka haetaan menusta *getSelectedItem*-metodilla. *SelectionItem* kutsuu valittaessa sisältämäänsä *Command*-oliota.



Kuva 11: GuiScrollMenun tapahtumasekvenssikaavio

4.3.2 Gui-pakkauksen käyttö

Esimerkkisovellukseen halutaan käynnistettäessä näkyvä latauskuva. Sovelluskehys tarjoaa valmiin *GenericScreen*istä periytetyn toteutuksen *SplashScreen*. Sille annetaan konstruktorissa parametrina seuraavaksi näytettävä *GenericScreen*-luokasta periytetty näkymä, joka näytetään halutun viiveen jälkeen. Lisäksi annetaan kuva ja haluttaessa myös kuvan alla näytettävä teksti. *SplashScreen* käyttää omaa säiettä viiveen toteuttamiseen ja se käynnistetään *execute*-metodilla. Listauksessa 9 on esimerkkisovelluksen *MIDlet*-luokan *startApp*-metodissa käytetty latausruudun käyttöesimerkki. Kuvassa 12 on listauksella aikaansaatava latauskuva, joka oletusarvoisesti näkyy reilun sekunnin ajan.

```
...
screen = new OmaScreen(false,this);
SplashScreen loadScreen = new SplashScreen(
                                    screen, "/challenge.png",
                                   "ladataan...", this);
loadScreen.execute();
...
```

Listaus 9: SplashScreen-käyttöesimerkki

Kuva 12: Latauskuva

Esimerkkisovelluksessa tarvitaan komento sovelluksen sulkemiseksi. Listauksessa 10 on komento-olion toteutus, joka kutsuu *OmaMidlet*-luokan metodia sovelluksen sulkemiseksi.

```
public class ExitCommand implements GuiCommand{
  public void execute() {
      OmaMidlet.getInstance().closeApp();
  }
}
```

Listaus 10: Esimerkkisovelluksen exit-komento.

Listauksessa 11 on esimerkkisovelluksen käyttöliittymän *GenericScreen*-luokkaa laajentava luokka *OmaScreen*, joka luo konstruktorissaan *GuiScrollMenu*n ja lisää siihen *Sprite*n avulla animoidun menuvalinnan. Menu lisätään *addComponent*-metodilla. Listauksen avulla syntynyt käyttöliittymä on esitetty kuvassa 13.

```
public class OmaScreen extends GenericScreen implements
                        CommandListener, GuiScreen{
private GuiScrollMenu menu;

public MainMenuScreen(boolean arg0) {
  menu = new GuiScrollMenu(
             this.getWidth()*3/4, this.getHeight(), 0);
  try {
    GuiCommand exit = new ExitCommand();
    GuiCommand start = new StartCommand();
    Sprite s = new Sprite(
              Image.createImage("/nappi.png"),69,33);
    s.setPosition(getWidth()-69,0);
    Sprite s2 = new Sprite(
               Image.createImage("/nappi.png"),69,33);
    s2.setPosition(getWidth()-69-7,33);
    menu.addSelection("Lopeta", s, exit);
    menu.addSelection("Aloita", s2, start);
  } catch (IOException e) {}
 this.addComponent(menu);
}
```

*Listaus 11: Esimerkki GenericScreen*iä *laajentavasta valikosta.*

Kuva 13: Menun lisäys

Itse *GuiScrollMenu* sisältää sisäisessä *Vector*-tietorakenteessa listan *SelectionItem*-luokan jäseniä sekä tiedot menun koosta ja sijainnista. Lisäksi se tarjoaa mm. metodit *moveUp*- ja *moveDown*-menussa liikkumiseksi. Piirrettäessä *GuiScrollMenu* rajapinnan määrittelemällä *draw*-metodilla se kutsuu järjestyksessä kaikkien sisältämiensä *SelectionItem*-luokkien piirtometodia. Sovelluksen tehtäväksi jää *draw*-metodin käyttö näytön päivittämiseen tarvittaessa. Yksi mahdollinen toteutus on säie, joka kutsuu metodia jatkuvasti.

Muut GuiComponent-toteutukset

GuiScrollMenu-toteutuksen lisäksi sovelluskehys tarjoaa valmiit toteutukset *GuiPopupMessage* ja *GuiFlashMessage*, erilaisten viestien näyttämiseksi käyttäjälle. Lisäksi tarjotaan *GuiGauge* mittarien toteuttamiseksi sekä *GuiDigitalDisplay* numeronäyttöjen piirtämiseen.

Esimerkkisovellukseen tarvitaan muutama mittari edellä mainitun valikon lisäksi. Näiden luominen ja lisääminen esimerkkisovelluksen käyttöliittymäruutuun on esitetty listauksessa 12 ja lisäysten seurauksena syntyvä käyttöliittymä kuvassa 14. Mittarit tarjoavat erilaisia toimintatiloja ja asetuksia, joita voidaan säädellä set-metodeilla. *GuiDigitalDisplay*-luokan konstruktorissa annetaan parametreiksi numeroiden lukumäärä ja mittarin sijainti. Lopuksi mittarit lisätään näkymään *addComponent*-metodilla. *GuiGauge*-luokalle annetaan vastaavasti sijainti sekä asetetaan pehmeästi liikkuvan neulan toiminta pois päältä *setSmoothMode*-metodilla. *GuiPopupMessage*-vaatii konstruktorissa sijainnin lisäksi myös koon. Esimerkkisovelluksessa sitä käytetään viestien näyttämiseen käyttäjälle.

```
...
display1 = new GuiDigitalDisplay(3,this.getWidth()/4,
                                 this.getHeight()/4+90);
display2 = new GuiDigitalDisplay(3,this.getWidth()*3/4,
                                 this.getHeight()/4+90);
display2.setTitle("Paras");
gauge1 = new GuiGauge(this.getWidth()/4,
                      this.getHeight()/4);
gauge1.setSmoothMode(false);
message = new GuiPopupMessage(false,getWidth()/2,
                       getHeight()-15,getWidth()-20,30);

this.addComponent(display1);
this.addComponent(display2);
this.addComponent(gauge1);
this.addComponent(message);
...
```

Listaus 12: Muiden käyttöliittymäkomponenttien lisäys.

Kuva 14: Käyttöliittymä

4.3.3 Gui-pakkauksen soveltuvuus ja resurssivaatimukset

Sovelluskehyksen käyttöliittymätoteutus soveltuu parhaiten sovelluksille, joiden käyttöliittymä koostuu yksittäisistä piirrettävistä komponenteista. Kehyksen valmiit toteutukset tukevat parhaiten erilaisista mittareista, tekstikehyksistä tai valikoista koostuvia käyttöliittymiä, sillä näitä tarjotaan valmiina ja niiden ulkoasua voi muutella tarjottujen metodien avulla. Kehys ei kuitenkaan sisällä valmista tekstin muokkaamiseen soveltuvaa komponenttia. Komponenteissa tai valikoissa voi olla animaatioita, mutta sovelluksen on huolehdittava niiden päivittämisestä. Tällöin päivittäminen kannattaa tehdä säikeessä. Vaihtoehtona sovellus voi päivittää komponentteja vain silloin, kun ne muuttuvat. Myös valmiiden toteutusten laajentaminen tai uusien luominen on mahdollista. Osa valmiista komponenteista käyttää png-muotoisia kuvaresurssitiedostoja osana

ulkoasuaan. Esimerkiksi *GuiDigitalDisplay* käyttää kuvaresurssia, josta muodostetaan *Sprite*-luokka numeron näyttämiseen.

4.4 Audio-pakkaus

4.4.1 Audio-pakkauksen toteutus

Ääniformaatit

Mobile Media APIa käyttävät mobiililaitteet tukevat vaihtelevaa joukkoa audioformaatteja. MMAPI ei kuitenkaan määrittele audioformaatteja, joita kaikkien MMAPI-laitteiden pitää tukea [3]. Yleisesti tuettuja äänitiedostoformaatteja ovat mm. wav ja amr. Myös mp3-tuki löytyy monista uusista laitteista, mutta se ei ole niin yleinen kuin edellä mainitut. Pakkaamattomien wav-äänten huono puoli on niiden suuri koko; pienelläkin pakkaussuhteella vastaava amr-tiedosto mahtuu noin kymmenesosaan wav-äänen viemästä tilasta. Toisaalta äänen laatu hieman kärsii. Lisäksi amr toimii paremmin puheelle kuin muille äänille. Koska tilansäästö on erityisen tärkeää juuri mobiilisovelluksissa, ja kehitettävä sovellus sisälsi lukuisia puheäänitiedostoja, amr-tiedostojen käyttö oli toimiva ratkaisu.

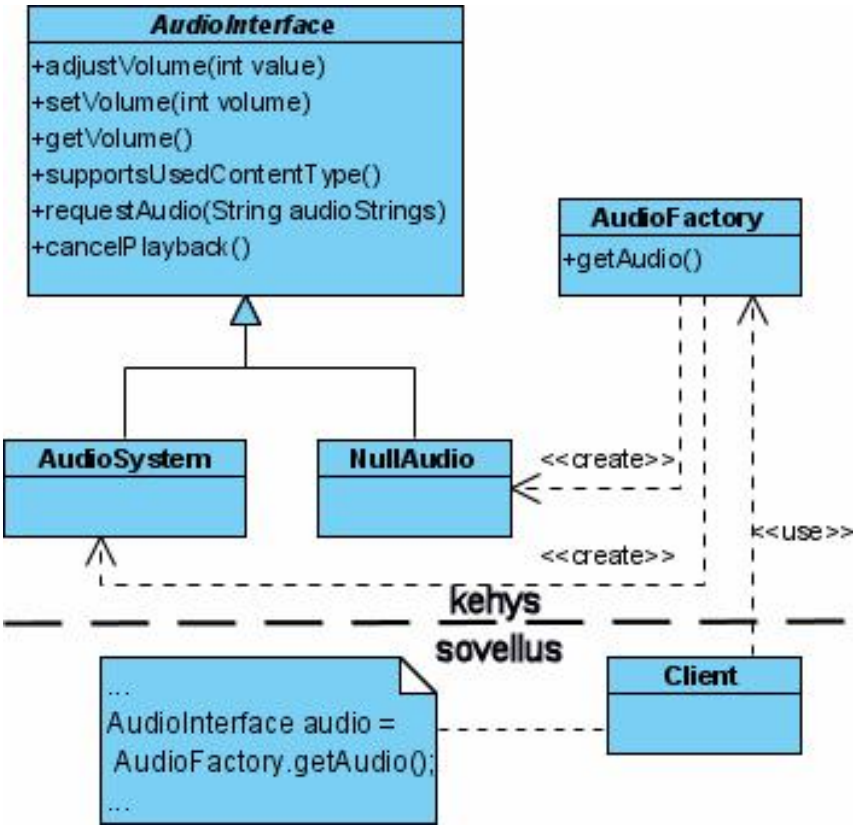
Äänentoistojärjestelmä

Äänentoisto koostuu rajapinnasta *AudioInterface*, joka määrittelee metodit, joiden kautta äänentoistoa käytetään sekä sitä toteuttavista konkreettisista luokista *AudioSystem* ja *NullAudio*. Näistä *NullAudio* on tyhjä toteutus, joka on tarkoitettu tilanteisiin, joissa laitteesta ei löydy MMAPIa tai sen käytössä on ongelmia. *AudioSystem* tarjoaa toiminnallisuuden yksittäisten ääniefektien toistamiseen sekä useista erillisistä äänistä koostuvien kokonaisuuksien toistamiseen. Jälkimmäistä käytetään erityisesti puhepalautteen antamiseen käyttäjälle. Puhepalautetoiminnallisuuteen kuuluu metodi kokonaislukujen muuttamiseksi puheeksi. Metodi yhdistää luvun numerot ennalta määriteltyihin äänitiedostoihin ja palauttaa *String*-taulukon toistettavista äänitiedostoista.

Äänentoistojärjestelmään kuuluu lisäksi luokka *AudioFactory*, joka tarjoaa tehdasmetodin *getAudio* äänentoiston saamiseksi käyttöön. Metodi tarkastaa laitteen kyvyn toistaa ääniä testaamalla ensin MMAPIn olemassaolon *Class.forName(…)* –metodilla, jonka jälkeen se käyttää *AudioInterface*n määrittelemää *supportsUsedContentType*-metodia

tarkistamaan, onnistuuko halutuntyyppisten äänien toistaminen. Jos molemmat tarkistukset menevät läpi, *getAudio* palauttaa *AudioInterface*na luokan *AudioSystem*. Muussa tapauksessa palautetaan *NullAudio*. Sovellus voi haluttaessa käyttää metodia *supportUsedContentType* tarkistamaan, onko äänentoisto mahdollista.

Kuvassa 15 on esitetty äänentoistoon liittyvien luokkien luokkakaavio. Asiakassovellus käyttää äänentoistoa aina *AudioInterface*n kautta ja ottaa äänentoiston käyttöön *AudioFactory*n avulla.



Kuva 15: Äänentoistojärjestelmän luokkakaavio

4.4.2 Audio-pakkauksen käyttö

Listauksessa 13 on esimerkki äänentoistojärjestelmän käytöstä. Esimerkkisovelluksessa otetaan *MIDlet*-luokassa ohjelman käynnistyessä äänentoisto käyttöön *AudioFactory*n avulla. Tämän jälkeen testataan onnistuuko äänentoisto *supportUsedContentType*-metodilla ja näytetään tarvittaessa käyttäjälle virheilmoitus.

```
...
AudioInterface audio;
audio = AudioFactory.getAudio();
if(!audio.supportsUsedContentType() {
  näytäViesti(“Laite ei tue äänien käyttöä.”);
}
...
```

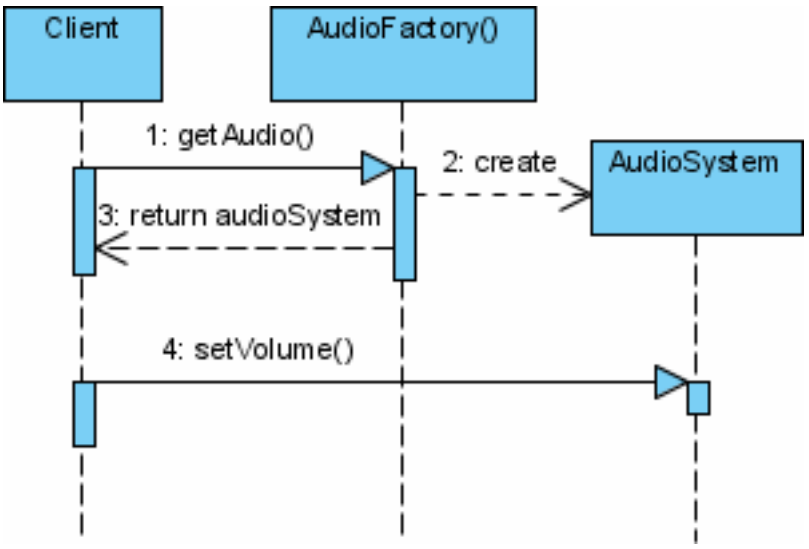
Listaus 13: Äänentoiston käyttöesimerkki

Esimerkkisovelluksen halutaan toistavan puheena ennätyksen reaktioajan. *AudioInterface* määrittelee metodin *requestAudio(String[] audioString)*, jolle annetaan parametriksi taulukko toistettavista äänitiedostoista. Numerot voidaan muuttaa toistettaviksi tiedostoiksi helposti *AudioSystem*-luokan tarjoaman *intToStrings*-apumetodin avulla. Näin esimerkkisovelluksen asianmukaiseen kohtaan lisätään listauksen 14 osoittamalla tavalla.

```
int time = getReactionTime();


if (time < record) {
        ...
    audio.requestAudio(AudioSystem.intToStrings(time));
        ...
}
```

Listaus 14: Äänentoiston käyttö

Kuvassa 16 on esitetty tapahtumasekvenssikaavio äänentoiston käytöstä. Asiakassovellus *Client* kutsuu *AudioFactory*n *getAudio*-metodia joka luo äänentoistojärjestelmän. Tämän jälkeen asiakassovellus säätää äänenvoimakkuutta *setVolume*-metodilla.



Kuva 16: Tapahtumasekvenssikaavio Audio-pakkauksen luokkien käytöstä

4.4.3 Audio-pakkauksen soveltuvuus ja resurssivaatimukset

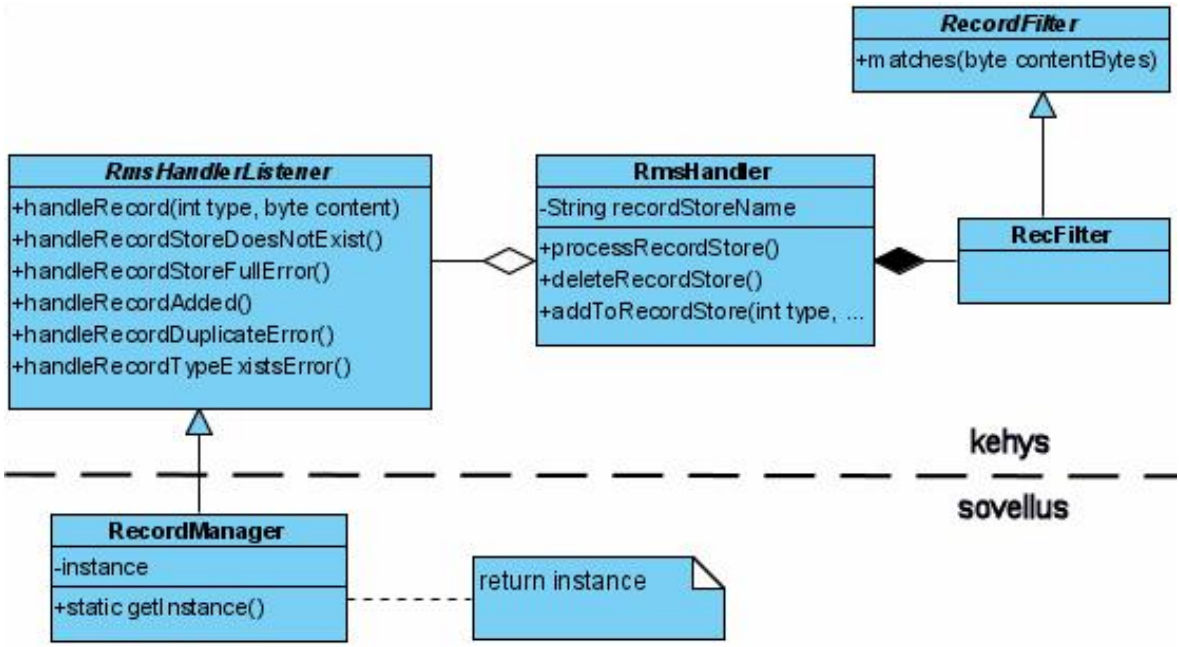
Audio-pakkauksen äänentoisto soveltuu sekä yksittäisten ääniefektien että useista eri äänistä koostuvien kokonaisuuksien toistamiseen. Rajoituksena äänien on oltava sellaisia, että ne muodostetaan tiedostoista sekä niiden on oltava samaa tyyppiä. Jos halutaan käyttää äänentoistoa numeroiden toistamiseen, on numerot muodostavat ääniresurssit liitettävä mukaan ja nimettävä odotetulla tavalla. Jotta äänet saa kuuluviin, on laitteessa oltava MMAPI, mutta sovelluskehyksen audio-pakkausta voi käyttää myös laitteissa joista MMAPI puuttuu.

4.5 Recordstore-pakkaus

4.5.1 Recordstore-pakkauksen toteutus

Sovelluskohtaisen datan tallennus recordstore-pakkauksessa tapahtuu kahdessa tasossa. Pakkauksessa oleva *RmsHandler*-luokka hoitaa datan lukemisen ja tallentamisen tavutasolla ja välittää tietoa eteenpäin sovellukselle *RmsHandlerListener*-rajapintaa käyttäen. *RmsHandlerListener*-rajapintaa toteuttava luokka hoitaa itse datan sisällön tulkitsemisen ja kirjoitettavan datan muotoilemisen. *RmsHandler* käyttää apunaan sisäluokkaa *RecFilter*, joka on *javax.microedition.rms.RecordFilter*-rajapinnan toteutus. *RecFilter*iä käytetään datasisällön tai tyypin vertailuun.

Kuvassa 17 on esitetty luokkakaavio recordstore-pakkauksen sisältämän järjestelmän rakenteesta. *RmsHandlerListener*-rajapintaa toteuttaa tässä konkreettinen asiakassovelluksen luokka *RecordManager*, joka on rakenteeltaan singleton.



Kuva 17: Recordstore-pakkauksen luokkakaavio

RmsHandler tallentaa dataa pakettimuotoisesti siten, että paketin sisältöä edeltää paketin datan koko ja tyyppi. *RmsHandler* ei itse välitä datan tyypistä vaan välittää sen eteenpäin *RmsHandlerListener*-rajapinnan määrittelemän metodin *handleRecord(int type,byte[] content)* kautta. Näin *RmsHandler* erottelee datan lukemisen ja kirjoittamisen sen sisällön käsittelystä. Yksi *RmsHandler* voi käyttää vain yhtä recordstorea, mutta haluttaessa voidaan luoda useita *RmsHandlereita,* joista kukin käyttää eri recordstorea. Dataa voidaan kirjoittaa metodin *addToRecordStore(int type,byte[] data, boolean noDuplicates,boolean noMatchingTypes,boolean overWrite)* avulla. Metodi antaa mahdollisuuden kontrolloida datan kirjoitustapaa muuttujilla *noDuplicates*, *noMatchingTypes* ja *overWrite*. Näistä *noDuplicates* tarkistaa, onko recordstoressa jo täsmälleen samansisältöinen merkintä eikä anna lisätä vastaavaa. Muuttuja *noMatchingTypes* taas estää enemmän kuin yhden samaa tyyppiä olevan merkinnän lisäämisen. Muuttuja *overWrite* puolestaan ylikirjoittaa vanhan merkinnän, jos sellainen on. Virhetilanteissa käytetään rajapinnan metodeita

- handleRecordStoreDoesNotExist, kun recordstorea ei ole olemassa.
- handleRecordStoreFullError, kun recordstore on täynnä.
- handleRecordDuplicateError, kun samanlainen merkintä löytyy eikä niitä sallita.

- handleRecordTypeExistsError, kun sama tyyppi löytyy eikä useita samaa tyyppiä olevia tai ylikirjoitusta sallita.

Etsintä kirjoituksen yhteydessä on toteutettu *javax.microedition.rms. RecordFilter*–rajapinnan avulla *RmsHandler*in sisäluokassa *RecFilter*. *RecordFilter* rajapinta määrittelee vertailumetodin *matches(byte[] candidate)*, jossa *candidate* sisältää testattavat tavut.

4.5.2 Recordstore-pakkauksen käyttö

Koska on usein sovelluskohtaista miten ja mitä dataa halutaan lukea ja kirjoittaa, ei sovelluskehys tarjoa valmista *RmsHandlerListener*-toteutusta. Listauksissa 15, 16 ja 17 on esitetty keskeisiltä osiltaan esimerkki-sovelluksessa käytetty toteutus *RecordManager,* joka käyttää singleton-rakennetta. Tästä on se hyöty että luokkaan päästään käsiksi mistä vain aina, kun halutaan kirjoittaa dataa recordstoreen. Listauksessa 15 on singletonin rakenteeseen liittyvät konstruktori ja *getInstance*–metodi. Viitettä *RecordManager*-ilmentymään pidetään singleton-mallin mukaan staattisessa muuttujassa *instance*, johon päästään käsiksi staattisessa metodissa *getInstance*. *RecordManager* käyttää staattisia muuttujia myös tyyppitiedoille, joita esimerkissä on kaksi.

```
public class RecordManager implements
                          RmsHandlerListener{
  public static RecordManager instance;
  public static final int ENNATYS = 1;
  public static final int NIMI = 2;
  private int[] recordsToRead;
  private RmsHandler handler;

  private RecordManager() {
          instance = this;
          handler = new RmsHandler(this,"OmaRecord");
  }

  public static RecordManager getInstance() {
          if (instance == null) {
                    instance = new RecordManager();
          }
          return instance;
  }
```

Listaus 15: RmsHandlerListener-esimerkkitoteutusta

Listauksessa 16 on esitetty kaksi eri *RmsHandler*ia käyttävää kirjoitus-metodia *writeRecord(...)*, joista toista käytetään kokonaislukujen ja toista merkkijonojen kirjoittamiseen. Apuna tavudatan muodostamisessa käytetään *Converter*-luokkaa.

```
public boolean writeRecord(int type,int value) throws
RecordManagerException{
           byte[] data = new byte[4];
           Converter.writeInt(data, value);
           return handler.addToRecordStore(
                      type, data, true, true, true);
}

public boolean writeRecord(int type,String value) throws
RecordManagerException{
           byte[] data = value.getBytes();
           return handler.addToRecordStore(
                      type, data, true, true, true);
}
```

Listaus 16: Kirjoitusmetodit

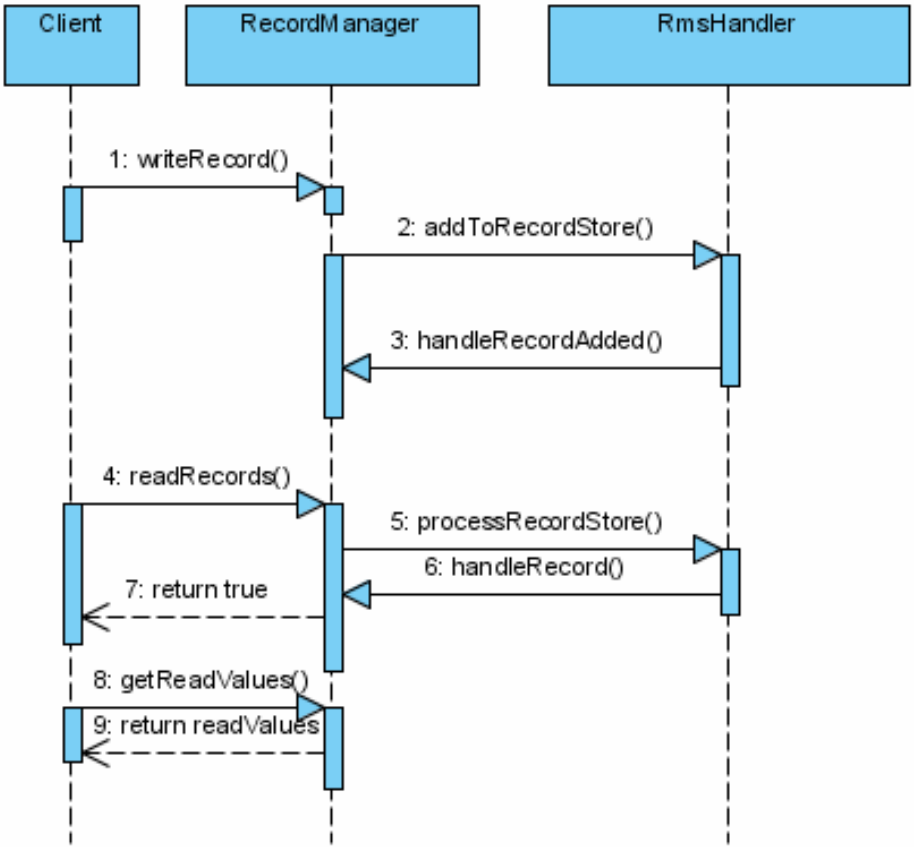
Datan lukemiseen käytetään metodia *readRecords(int[] recs)*, joka ottaa parametriksi haluttavat datatyypit ja tallentaa ne *recordsToRead*-taulukkoon ja käynnistää *RmsHandler*in *processRecordStore*-metodilla. *RmsHandler*in lähettämä data käsitellään *handleRecord*-metodissa, jossa tavuista muodostetaan tyypin perusteella kokonaisluku tai merkkijono. Vain halutut tyypit käsitellään ja muut jätetään huomiotta. Lukumetodit on esitetty listauksessa 17.

```
public boolean readRecords(int[] recs) throws
                                        RecordManagerException{
           if (recs == null) return false;
           recordsToRead = recs;
           values = new Object[recs.length];
           return handler.processRecordStore();
}

public void handleRecord(int type, byte[] content) {
           int index = -1;
           for(int i=0;i<recordsToRead.length;i++) {
             if(type==recordsToRead[i]) {
               index = i;
             }
           }

           if (index == -1) return;
```

```
            switch(type) {
              case ENNATYS:
                values[index] = new Integer(Converter.
                                readInt(content));
                 break;
              default:
                break;
            }
}
```

Listaus 17: Lukumetodit

Kuvan 18 tapahtumasekvenssikaaviossa on havainnollistettu RecordStore-pakkauksen käyttöä. Asiakassovellus *Client* käyttää kirjoittaa ensin edellä esitetyn kaltaisella *RecordManager*in *writeRecord*-metodilla dataa. Kirjoituksen päätyttyä luetaan arvoja *readRecords*-metodilla. Jos haluttuja arvoja on useita, on *handleRecord*-kutsuja vastaavasti useita. Lopuksi haetaan luetut arvot *get*-metodilla.



Kuva 18: Tapahtumasekvenssikaavio Recorstore-pakkauksen käytöstä

4.5.3 Recordstore-pakkauksen soveltuvuus ja resurssivaatimukset

Sovelluskehyksen recordstore-pakkauksen toteutus soveltuu kaikille RMS:ää käyttäville sovelluksille. Jos kuitenkin halutaan ainoastaan tallentaa

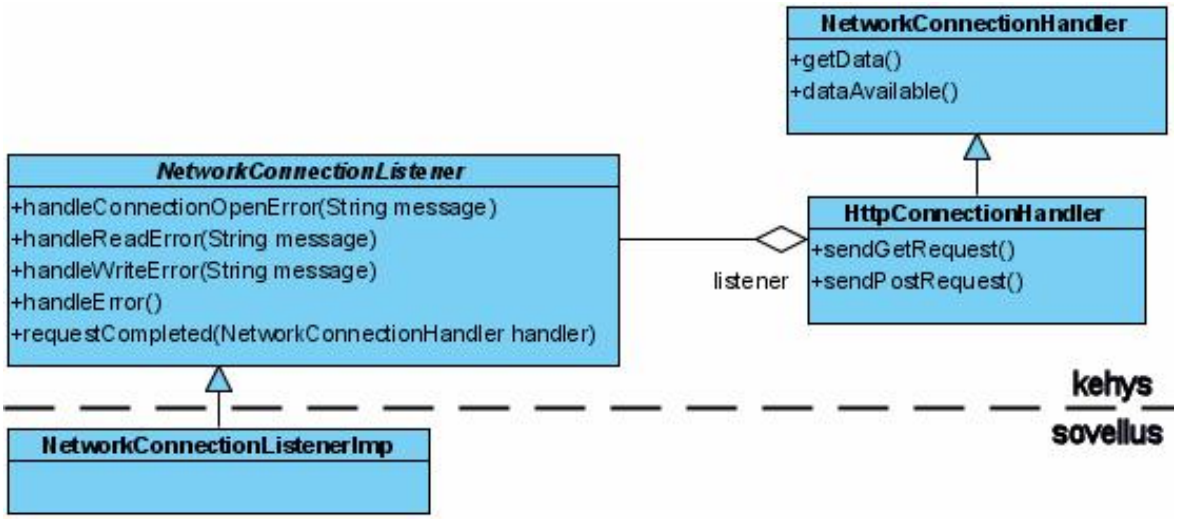
yksittäinen merkkijono, se voi olla turhan vaativa. Jos taas halutaan tallentaa erityyppistä dataa ja eritellä näitä tyypin mukaan on kehyksen tarjoama toteutus varteenotettava vaihtoehto. Kehyksen toteutus ei kuitenkaan tarjoa valmiita ratkaisuja tallennetun datan käsittelyyn vaan tämä jää sovelluksen toteutettavaksi. Järjestelmän käyttö ei aseta ohjelmille tai laitteistolle erityisvaatimuksia. Jos laitteen tarjoama tallennustila on liian pieni, järjestelmä antaa mahdollisuuden käsitellä tästä aiheutuva virhe tallennusta yritettäessä.

4.6 Network-pakkaus

4.6.1 Network-pakkauksen toteutus

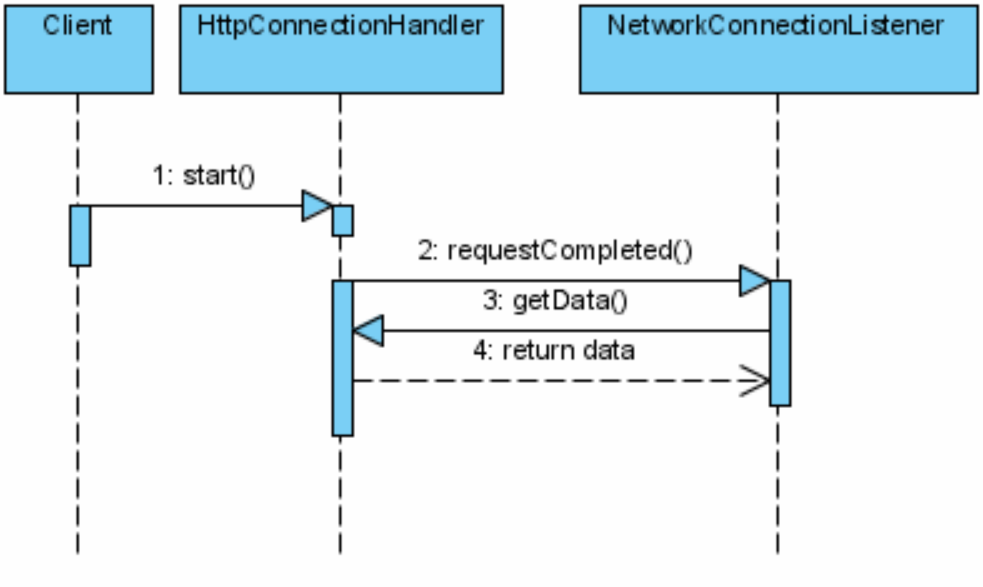
Network-pakkauksen tarjoamaksi toiminnallisuudeksi riittää datan lähettäminen ja vastaanotto käyttäen HTTP-protokollaa. Se muistuttaa rakenteeltaan edellä esitettyä Bluetooth-yhteyden toteutusta.

Network-pakkauksen luokkakaavio on esitetty kuvassa 19. Rajapinta *NetworkConnectionListener* määrittelee metodit, joilla yhteyden toteuttava luokka *HttpConnectionHandler* kommunikoi sovelluksen kanssa. *HttpConnectionHandler* on säie, joka käynnistyessään hoitaa yhteyden muodostamisen ja datan lähetyksen ja lukemisen. Lopetettuaan onnistuneesti tai virhetilanteeseen luokka ilmoittaa omalle *NetworkConnectionListener*-rajapintaa toteuttavalle kuuntelijaluokalle tilanteesta. *NetworkConnectionHandler*-rajapinta määrittelee metodit datan saamiseksi *ConnectionHandler*-olioilta. Asiakassovellus toteuttaa *NetworkConnectionListener*-rajapinnan. Toteutus on kaaviossa esitetty *NetworkConnectionListenerImp*-luokkana. Toistaiseksi *HttpConnection-Handler* on ainoa kehyksen *NetworkConnectionHandler*-rajapintaa toteuttava luokka.



Kuva 19: Network-pakkauksen luokkakaavio

Kuva 20 esittää asiakassovelluksen ja rajapinnan toteutusten välistä kommunikointia tilanteessa, jossa asiakassovellus käynnistää ensin *HttpConnectionHandler*in, joka suorittaa onnistuneesti yhteydenoton palvelimeen. *NetworkConnectionListener*-rajapinnan toteuttava luokka hakee tämän jälkeen datan *getData*-metodilla.



Kuva 20: Network-pakkauksen tapahtumasekvenssikaavio

4.6.2 Network-pakkauksen käyttö

Oletetaan, että on olemassa palvelin, joka vastaanottaa esimerkkisovelluksen ennätyksen HTTP POST-pyynnön avulla. Esimerkkisovelluksen halutaan lähettävän ennätys osoitteeseen www.osoite.net/palvelin. Lisätään esimerkkisovelluksen *MIDlet*-luokkaan metodi *networksend* listauksen 18 mukaisesti. Metodissa luodaan ensin *HttpConnectionHandler*, jolle annetaan palvelimen osoite, *NetworkConnectionListener*-rajapinta, sekä ohje käyttää POST-pyyntöä. Metodilla *setSendData* asetetaan *Converter*-luokan avulla muunnettu lähetettävä binääridata. Tämän jälkeen säie käynnistetään.

```
public void networkSend() {
  HttpConnectionHandler handler =
                       new HttpConnectionHandler(
                         "http://www.osoite.net/palvelin",
                       this, true);
  byte[] sendData = new byte[4];
  Converter.writeInt(sendData, record);
  handler.setSendData(sendData);
  handler.start();
}
```

Listaus 18: Esimerkkisovelluksen verkkoyhteys

Esimerkkisovelluksessa *MIDlet*-luokka itse toteuttaa *NetworkConnection-Listener*-rajapinnan. Esimerkkisovellus voi esimerkiksi näyttää virhetilanteet näytöllä seuraavasti:

```
public void handleReadError(String message) {
  screen.setMessage(message);
}
```

Vaihtoehtona sovellus voi antaa oman virheviestinsä. Muiden metodien toteutus voidaan hoitaa vastaavasti tai jättää tässä tapauksessa tyhjäksi. Lähetyksen aktivointia varten voidaan lisätä luvussa 4.3 käsiteltyyn menurakenteeseen valinta ja komento *networkSend*–metodin kutsumiseksi.

4.6.3 Network-pakkauksen soveltuvuus ja resurssivaatimukset

Kehyksen network-pakkauksen toteutus soveltuu sovelluksille, jotka haluavat lähettää ja lukea dataa palvelimelta käyttäen HTTP-protokollaa. Se tukee kuitenkin toistaiseksi ainoastaan GET- ja POST-pyyntöjä. Toteutus ei tarjoa valmiita ratkaisuja datan käsittelyyn, sillä tämä on sovellus- ja palvelinkohtaista. Myös POST-pyynnöllä lähetettävän datan muodostaminen jää sovelluksen tehtäväksi.

4.7 Muut toiminnot

4.7.1 Lokalisointi

Lokalisoinnin teoriaa

Määrittelyn yhteydessä valittiin että sovellus toimisi aina yhdellä kielellä. Sovelluksen kääntäminen muille kielille on kuitenkin otettava huomioon. Sun Microsystems esittää kolme erilaista vaihtoehtoa mobiilisovellusten lokalisoinnin toteuttamiseksi: [3]

1. Lokalisoidut merkkijonot kirjoitetaan jad-tiedostoon.
2. Lokalisoidut merkkijonot sijaitsevat erillisessä resurssitiedostossa.
3. Luokkatiedostoja käytetään lokalisointiin.

Vaihtoehdon 1 hyvät puolet ovat ratkaisun helppous. Lisää ohjelmakoodia ei tarvita. Lisäksi kääntäjien työ helpottuu, sillä merkkijonot ovat jad-tiedostosta helposti luettavissa. Lisäksi jad-tiedostojen käyttö mahdollistaa jakelujärjestelmän, jossa jad-tiedostot jaetaan sijainnin mukaan.

Ratkaisun huonona puolena esitetään sen soveltumattomuus suurille tekstimäärille. Lisäksi suuri jad-tiedoston koko saattaa vaikuttaa sovelluksen nopeuteen ratkaisevalla tavalla. Lisäksi jad-tiedostojen käyttötarkoituksia rajoittaa ratkaisun rajoittuminen vain tekstiresursseihin eikä esim. kuviin.

Vaihtoehto 2 on myös helppo kääntäjille. Lisäksi sen etuna on jokaisen kielen rajoittuminen omaan tiedostoon. Tämä mahdollistaa myös monikieliset sovellukset suhteellisen pienillä lisäyksillä, jolloin lisää kieliä saisi lisäämällä kielipaketteja.

Resurssitiedostojen käytön huono puoli on lisääntynyt koodaustyö. Ohjelmoijan täytyy toteuttaa tiedoston lukeminen ja käsittely. Lisäksi on hyvä huomata, että Java ME ei tarjoa *StringTokenizer*-luokkaa, joten vastaavat merkkijonojen manipulointitoiminnot pitää toteuttaa itse. Toteutukseen liittyy myös byte streamin muuntaminen merkeiksi. Sovelluksen on vielä säilytettävä luettuja merkkijonoja jossain sisäisessä tietorakenteessa käyttöä varten. Tiedostojen lukeminen ja käsittely myös hidastaa käynnistämistä.

Kolmas vaihtoehto, eli luokkien käyttö, helpottaa ohjelmoijan työtä, sillä merkkijonojen käsittelyä ja lukemista varten ei tarvitse lisää ohjelmakoodia. Lisäksi ratkaisussa voidaan ottaa huomioon myös muut resurssit kuin merkkijonot. Lisäksi ratkaisu toimii nopeasti.

Huonona puolena lokalisointi vaikeutuu, sillä merkkijonot eivät ole yhtä helposti ei-teknisten henkilöiden luettavissa kuin muuten. Lisäksi ohjelman koko kasvaa ja muistin käyttö lisääntyy.

Vaihtoehtona Mobile Internationalization API

Sun esittää vielä yhden vaihtoehtoisen tavan toteuttaa lokalisointi käyttäen apuna Mobile Internationalization APIa (JSR 238). Se tarjoaa tavan tuottaa lokalisoitua päivämäärä-, valuutta- yms. -formaattia sekä merkkijonojen lajittelua kielen mukaan. Mobile Internationalization APIn käyttö vaatii kuitenkin mobiililaitteen, jossa tuki ominaisuudelle löytyy. Tällaisia laitteita ei vielä 2007 ollut paljon markkinoilla, mistä johtuen se ei vielä ole varteenotettava vaihtoehto, jos halutaan ohjelmiston toimivan mahdollisimman monessa laitteessa.

Lokalisointiratkaisu

Ohjelman lokalisoinnissa päädyttiin luokkatiedostojen käyttöön, sillä ohjelma sisältää myös käännettäviä audioresursseja, jolloin muista ratkaisuista saatavat hyödyt jäävät osittain käyttämättä. Ohjelmasta pitää kuitenkin tehdä eri kieliä varten oma jar-paketti. Käännöstyötä pyritään helpottamaan sijoittamalla merkkijonot vain pariin tiedostoon, joissa ei ole muuta ohjelmakoodia. Näin esimerkiksi kaikki sovelluskehyksen yleisien toimintojen käyttämät merkkijonot löytyvät samasta luokasta ja sovelluksen muut merkkijonot tätä laajentavasta aliluokasta.

Lokalisointiratkaisun käyttö

Esimerkkisovelluksessa sovelluskohtaiset merkkijonot laitettiin luokkaan OmatStringit ja vaihdettiin kehyksen käyttämän merkkijonoluokan tilalle vastaava suomenkielinen luokka. Sovellus toimi tämän jälkeen odotetusti suomenkielisillä teksteillä. Käytettävät audioresurssit olisi kuitenkin vielä käännettävä erikseen.

4.7.2 Util-pakkaus

J2ME:n kirjastot ovat rajoittuneempia kuin esimerkiksi Standard Editionin. Siksi joitain tarpeellisia metodeja lisättiin util-pakkaukseen täydentämään J2ME:n luokkien tarjontaa. Näihin kuuluvat mm. metodit kokonaislukujen muuntamiseksi tavutaulukoksi ja takaisin, sekä joitain matemaattisia funktioita. Muuntometodit on sijoitettu luokkaan Converter ja matemaattiset metodit luokkaan MathExtended. Luokkien metodit ovat staattisia, joten niiden käyttö ei edellytä ilmentymän luomista. Esimerkiksi kokonaisluvun kirjoittaminen tavutaulukkoon Converter-luokan avulla onnistuu seuraavasti:

```
int luku = 12345;
byte[] data = new byte[4];
Converter.writeInt(data, luku);
```

5 TESTAUS JA VIIMEISTELY

5.1 Obfuskaattori

Obfuskaattorit (obfuscator) ovat ohjelmistoja, jotka muokkaavat koodia siten, että siitä tulee vaikeampaa kääntää takaisin lähdekoodiksi. Erityisesti Javan luokkatiedostot on helppo kääntää takaisin koodiksi. Vaikka katsottaisiin, ettei ohjelman suojaaminen takaisin kääntämiseltä (reverse engineering) olisi olennaista, voi obfuskaattoriohjelmissa olla myös muita hyödyllisiä ominaisuuksia.

Proguard

Proguard on ilmaisohjelmisto Java-luokkatiedostojen koon pienentämiseen, optimointiin sekä obfuskoimiseen. Se osaa havaita ja poistaa käyttämättömiä luokkia, kenttiä, metodeja ja attribuutteja sekä optimoida koodia. Se osaa myös nimetä uudelleen luokkia, kenttiä ja metodeja käyttäen lyhyitä ja merkityksettömiä nimiä, jolloin ohjelmaa on vaikeampi kääntää takaisin lähdekoodiksi. Lopputuloksena syntyvä jar-paketti on myös pienempi kuin alkuperäinen. Erityisesti mobiiliohjelmoinnista tästä on konkreettista hyötyä sillä laitteiden muistin koko on usein hyvin rajoittunut. Proguardista tai vastaavista ohjelmistoista voi olla hyötyä myös käytettäessä sovelluskehystä, jonka sisältämiä kaikkia luokkia ei tarvitakaan, tai ohjelman

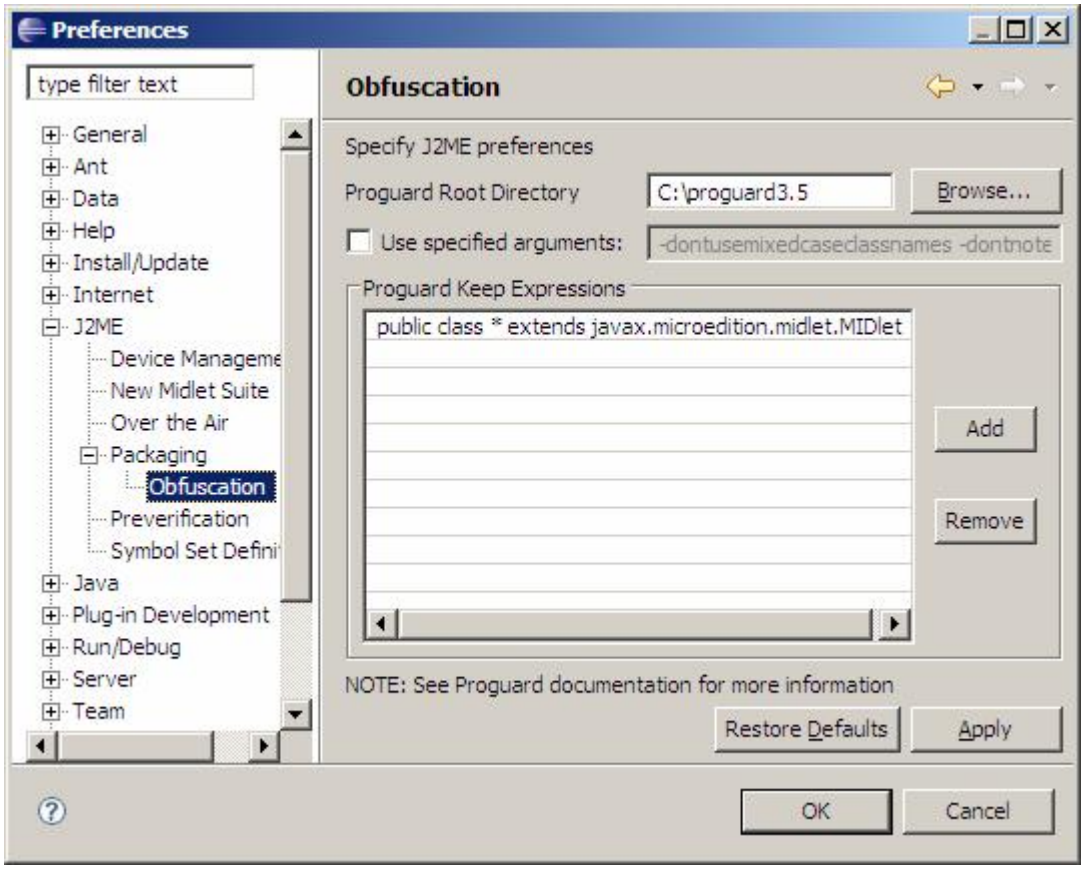
sisältäessä luokkia, joita ei tarvita juuri käännöstä tehtäessä. Tällöin nämä jäävät automaattisesti pois lopullisesta jar-paketista. [4]

Proguardin lisenssi sallii myös sen vapaan käytön kaupallisiin sovelluksiin ilman muutoksia sovelluksien lisensseihin. Proguard itse on GNU General Public Licence:n (GPL) alainen. [4]

Proguardin käyttö

Proguardin käyttö on suhteellisen helppoa. Se tarjoaa komentorivipohjaisen käyttöliittymän lisäksi myös graafisen käyttöliittymän, jonka avulla eri optioita on helppo kokeilla. Lisäksi EclipseME-laajennus tarjoaa mahdollisuuden integroida Proguardin toiminnallisuus osaksi kehitysympäristöä. Tässä yhteydessä käsitellään vain käyttö Eclipse-ympäristöstä.

EclipseME:n konfigurointi tapahtuu asettamalla preferences-asetuksista kohta J2ME->obfuscation->proguard->”proguard root directory” osoittamaan proguardin asennushakemistoon. Kuva 21 esittää EclipseME:n proguard asetusvalikkoa. Proguard keep expressions kohdassa tulee olla rivi ”public class * extends javax.microedition.midlet.MIDlet”, joka käskee proguardia säilyttämään kaikki luokan MIDlet aliluokat. Oletuksena tämän pitäisi olla jo valmiiksi täytettynä. Tämän jälkeen obfuskoitu paketti muodostetaan projektiasetuksista valitsemalla ”create obfuscated package”.



Kuva 21: Proguard asetukset

Esimerkkisovellus optimoitiin käyttäen Proguardin versiota 3.5. Lopputuloksena syntynyt jar-paketti pieneni noin 15 %. Jos poistetaan jar-paketista resurssitiedostot, jotka eivät vaikuta optimointiin ja tarkastellaan muutosta pelkästä koodista koostuvassa paketissa on kokoero noin 50 %.

5.2 Testaus

Sovelluskehyksen toimintaa testattiin sovelluskehityksen yhteydessä sekä Sun Wireless Toolkit:n emulaattorilla, että Nokia 5500 -matkapuhelimella. Ohjelmistokehityksen aikana yksittäisten toimintojen testaus on helpompaa emulaattorilla, mutta myös testaus sopivin väliajoin myös oikeassa mobiililaitteessa on tärkeää, sillä se paljasti ongelmia ja huomioitavia seikkoja, jotka eivät tule ilmi testattaessa ohjelmistoa emulaattorilla.

Seuraavat havainnot koskevat vain Nokia 5500 -matkapuhelinta, eikä niitä voi välttämättä yleistää koskemaan kaikkia laitteita. Havainnot on saatu testatessa kehystä käyttävää mobiilisovellusta.

MMAPIn ongelmat

MMAPIn toteutuksissa on ongelmia useissa mobiililaitteissa, joten sen käyttö voi olla hankalaa. Usein ongelmat eivät lisäksi ilmene Java-kielelle ominaisina poikkeuksina vaan itse laitteen käyttäytymisessä. Esimerkiksi yritys toistaa useita äänitiedostoja yhtä aikaa aiheuttaa Nokia 5500 sen, että vain pieni osa toisesta äänestä toistetaan. Lisäksi MMAPIn *Player*-luokan *deallocate*-metodin käyttö sekoittaa äänentoiston niin, ettei ääniä välttämättä saa kuuluviin enää ollenkaan tai ne alkavat toimia vasta pitkähkön viiveen jälkeen. Ratkaisuna on jättää metodi käyttämättä ja luoda uusi *Player*-luokan ilmentymä.

Sivusto j2meforums.com on koonnut laitekohtaisia MMAPIssa ilmenneitä ongelmia. [5] Sivuston mukaan kaikille laitteille pätevät seuraavat seikat:

- Kahden äänen yhtäaikaista toistamista pitäisi välttää.
- *SetLoopCount*-metodin käyttöä pitäisi välttää.
- *Deallocate*-metodin jälkeen tulisi pitää viive ennen uuden *Player*-luokan käyttöä.

Sivustolla vahvistetaan myös testatessa ilmennyt ongelma *deallocate*-metodissa Nokian puhelimissa ja mainitaan, että metodia ei tulisi käyttää. Muiden listattujen ilmiöiden paikkansapitävyyttä ei voida vahvistaa tämän työn yhteydessä, mutta niitä on pyritty ottamaan huomioon äänentoistojärjestelmän suunnittelussa.

Äänitiedostojen vaikutus nopeuteen

Ennen kuin MMAPIn *Player*-luokan avulla toistettava äänitiedosto saadaan toistettua, on *Player*-luokka valmisteltava *prefetch*-metodilla, joka ottaa käyttöön puhelimen äänentoistoresurssit ja minimoi viiveen äänentoiston aloittamisessa. Valmistelu tapahtuu myös automaattisesti *play*-metodin yhteydessä, jos sitä ei ole vielä tehty. Tästä huolimatta äänentoistossa ilmenee pieni viive ennen uuden äänen toistamista. Testattaessa äänentoistojärjestelmää sovelluksessa sekä wav- että amr-äänitiedostoilla huomattiin, että viive pienenee hieman käytettäessä pienempiä amr-äänitiedostoja. Viiveen täydellinen poistaminen lienee kuitenkin mahdotonta.

Bluetooth APIssa ilmenneet ongelmat

Testattaessa Bluetooth yhteyden käyttöä Nokia 5500 -kännykässä ilmeni ongelma Bluetooth APIn käytössä. Bluetooth APIn mukaan käytettäessä luokan *ServiceRecord* metodia *getConnectionURL* yhteysosoitteen saamiseksi on annettava haluttu tietoturvataso muuttujana. Esim.

```
url=serviceRecord.getConnectionURL(
                ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
                false);
```

Myös yhteyden palvelinpäässä annetaan osoite yhteyttä käynnistettäessä. Esim.

```
url=OMAURL + ”;authenticate=false;encrypt=false";
connectionNotifier=(StreamConnectionNotifier)
                    Connector.open(url);
connectionNotifier.acceptAndOpen();
```

Jos annetut tietoturvatasot poikkeavat toisistaan, ei synnykään poikkeusta vaan koko Java-virtuaalikone kaatuu ilmoittaen syyksi Symbian virheen 34 eli ”Failed to connect”. Tätä voi olla vaikea huomata, sillä emulaattorissa tilanne ei aiheuttanut ongelmaa eikä Symbianin virhekoodi ilmaise syytä tarkemmin. Ongelman ratkaisu oli tässä tapauksessa sen varmistaminen, että tietoturvatasovaatimukset vastaavat toisiaan.

6 YHTEENVETO

Työn tavoitteena oli muodostaa määrittelyjen pohjalta sovelluskehys, joka tukee haluttua toiminnallisuutta ja kiinnittää sen suunnittelussa huomiota mahdollisuuksiin käyttää apuna suunnittelumalleja. Käytännön sovelluksella tehtyjen testausten perusteella voidaan katsoa että kehyksen toiminnot täyttävät sille asetetut vaatimukset. Esimerkkisovelluksessa kehyksen käytöllä hoidetun koodin osuus on noin 80% koko sovelluksen ohjelmakoodista. Syntynyt sovelluskehys soveltuu yksikielisiin, mahdollisesti lokalisoitaviin sovelluksiin, joiden käyttöliittymän voidaan katsoa muodostuvan toisistaan erillisistä komponenteista – erityisesti mittareista ja valikoista. Komponentit voivat olla animoituja. Kehys tukee Bluetooth-sovelluksia, jotka tarvitsevat HTTP-protokollaa käyttäviä verkkoyhteyksiä, usean erityyppisten datan RMS-tallennusmahdollisuutta tai äänitiedostoista

muodostuvaa äänentoistoa. Myös suunnitelumalleista löytyi toimivia ratkaisuja käytettäväksi apuna suunnittelussa.

On mahdollista, että MMAPIn ongelmat saattavat aiheuttaa ongelmia joissakin mobiililaitteissa, mutta tämä selviää vain käytännön testauksen kautta. Ongelmatilanteissa äänentoiston toteutus mahdollistaa kuitenkin vaihtoehtoisien äänentoistojärjestelmien kehittämisen nykyisen rinnalle. Kehyksen toiminnallisuutta voidaan myöhemmin laajentaa lisäämällä uusia graafisia komponentteja. Bluetooth-laitehaun rajoitteeksi jää riippuvuus laitteen tarjoamista käyttöliittymäkomponenteista, sillä laitevalintalomake on integroitu toteutukseen. Jos halutaan myöhemmin, että laitehaun yhteydessä käytetään omaa graafista ulkoasua, on rajapintaa ja/tai toteutusta muutettava.

VIITELUETTELO

[1] Gamma, Helm, Johnson, Vlissides. *Olio-ohjelmointi suunnittelumallit.* Helsinki: Edita. 2001.

[2] Java ME Guideline: How to create localized application [verkkodokumentti, viitattu 9.4.2007] Saatavissa: http://developers.sun.com/techtopics/mobility/reference/techart/design_guidelines/localization.html .

[3] Overview JSR-135 [verkkodokumentti, viitattu 9.4.2007] Saatavissa: http://java.sun.com/javame/reference/apis/jsr135/overview-summary.html .

[4] ProGuard [verkkodokumentti, viitattu 11.4.2007] Saatavissa: http://proguard.sourceforge.net/ .

[5] MMAPI/Sound Information [verkkodokumentti, viitattu 9.4.2007] Saatavissa: http://www.j2meforums.com/wiki/index.php/MMAPI/Sound_Information#Device_Specific .

LIITE 1: Esimerkkisovelluksen lähdekoodi

```
//OmaMidlet-luokka
//
//MIDlet. Tarjoaa singleton-tyyppisen tavan päästä luokkaan
//käsiksi
//
//Toteuttaa NetworkConnectionListener-rajapinnan

import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.UUID;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;



import RecordManager.RecordManagerException;
import a4sp.audio.AudioInterface;
import a4sp.audio.AudioSystem;
import a4sp.bluetooth.ServiceDiscovery;
import a4sp.gui.GenericScreen;
import a4sp.gui.SplashScreen;
import a4sp.network.HttpConnectionHandler;
import a4sp.network.NetworkConnectionHandler;
import a4sp.network.NetworkConnectionListener;
import a4sp.util.Converter;
import a4sp.audio.AudioFactory;



public class OmaMidlet extends MIDlet implements  NetworkConnectionListener{
              public static OmaMidlet instance;

              AudioInterface audio;
              OmaScreen screen;
              ConnectionManager conn;

              private long messageInTime;
              private int record = 999;


              protected void startApp() throws MIDletStateChangeException {
                            instance = this;
                            screen = new OmaScreen(false,this);
                            SplashScreen loadScreen = new SplashScreen(screen,
                              "/challenge.png", OmatStringit.loading, this);
                            loadScreen.execute();

                            audio = AudioFactory.getAudio();
                            conn = new ConnectionManager();

                            int[] recs = {RecordManager.ENNATYS};
                            try {
                                          RecordManager.getInstance().readRecords(recs);
                                          Integer ennatys =
                                            (Integer)RecordManager.getInstance().
                                            getReadValues()[0];
                                          if (ennatys != null) {
                                                        record = ennatys.intValue();
                                          }
                            } catch (RecordManagerException e) {
                                          screen.setMessage(e.toString());
                            }

                            if(!audio.supportsUsedContentType()) {
                                            screen.setMessage(OmatStringit.noAudio);
                            }


                            screen.setRecord(record);
              }
```

```
public void display(Displayable disp) {
		Display.getDisplay(this).setCurrent(disp);
}

protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
		// TODO Auto-generated method stub

}

protected void pauseApp() {
		// TODO Auto-generated method stub

}

public void messageIn() {
		messageInTime = System.currentTimeMillis();
		screen.setMessage(OmatStringit.now);
}

public int getReactionTime() {
		if (messageInTime != 0) {
				int time = (int)(System.currentTimeMillis() –
					messageInTime);
				messageInTime = 0;
				return time;
		}
		else return 999;
}

public ConnectionManager getConnectionManager() {
		return conn;
}

public GenericScreen getScreen() {
		return screen;
}

public static OmaMidlet getInstance() {
		return instance;
}

public void closeApp() {

		try {

RecordManager.getInstance().writeRecord(RecordManager.ENNATYS, record);
		} catch (RecordManagerException e) {
				System.out.println(e.toString());
		}

		try {
				destroyApp(false);
				notifyDestroyed();
		} catch (MIDletStateChangeException ex) {
		}
}

public void startBluetooth() {
		if(!conn.isConnected()) {
				try {
					UUID uuid = new
				  UUID("AAAAA5EE9F9146109085C2055C888B39",false);

				UUID[] uuids = {uuid};
					ServiceDiscovery disc = new
				  ServiceDiscovery(getConnectionManager(),
				  1, 1, uuids, this);
					disc.start();
				} catch (BluetoothStateException e) {
					screen.setMessage(e.toString());
				}
		}
}
```

```
                public void reactionTime() {
                        int time = getReactionTime();
                        screen.setReactionTime(time);
                        screen.setMessage("");

                        if (time < record) {
                                record = time;
                                screen.setGauge(0);
                                screen.setRecord(record);
                                screen.draw();

                audio.requestAudio(AudioSystem.intToStrings(time));
                        }
                        else screen.setGauge(time-record);

                }

                public void networkSend() {
                        HttpConnectionHandler handler = new HttpConnectionHandler(
                                                "http://www.osoite.net/palvelin",
                                                this, true);
                        byte[] sendData = new byte[4];
                        Converter.writeInt(sendData, record);
                        handler.setSendData(sendData);
                        handler.start();
                }

                public void handleConnectionOpenError(String message) {
                        screen.setMessage(message);
                }

                public void handleError() {
                }

                public void handleReadError(String message) {
                        screen.setMessage(message);
                }

                public void handleWriteError(String message) {
                        screen.setMessage(message);
                }

                public void requestCompleted(NetworkConnectionHandler handler) {
                        screen.setMessage("OK");
                }

}
```

```
//ConnectionManager-luokka
//
//Esimerkkisovelluksen rajapinnat
//ServiceDiscoveryListener ja ConnectionHandlerListener
//toteuttava luokka

import javax.bluetooth.ServiceRecord;

import a4sp.bluetooth.ConnectionHandler;
import a4sp.bluetooth.ConnectionHandlerListener;
import a4sp.bluetooth.ServiceDiscoveryListener;


public class ConnectionManager implements ServiceDiscoveryListener,
ConnectionHandlerListener{

              ConnectionHandler conn;

              public void handleConnect(ServiceRecord s) {
                            OmaMidlet.getInstance().getScreen().show();
                            conn = new ConnectionHandler(this, s,
                                        ServiceRecord.NOAUTHENTICATE_NOENCRYPT);
                            conn.start();
              }

              public void handleMessage(String message) {
                            OmaMidlet.getInstance().getScreen().setMessage(message);
              }

              public void handleNewSearch() {
                            OmaMidlet.getInstance().getScreen().show();
                            OmaMidlet.getInstance().startBluetooth();
              }

              public void handleSearchCompleteNoCompatibleDevicesFound() {
                            OmaMidlet.getInstance().getScreen().setMessage("Ei löydy
                            bt-laitetta.");
              }

              public void handleSearchCompleteNothingFound() {
                            OmaMidlet.getInstance().getScreen().setMessage("Ei löydy
                            bt-laitetta.");
              }

              public void handleSearchError(String message) {
                            OmaMidlet.getInstance().getScreen().setMessage(message);
              }

              public void handleClose(ConnectionHandler handler) {
                            OmaMidlet.getInstance().getScreen().setMessage("Yhteys
                            suljettu.");
              }

              public void handleErrorClose(ConnectionHandler handler, String errorMessage)
              {

              }

              public void handleOpen(ConnectionHandler handler) {
                            OmaMidlet.getInstance().getScreen().setMessage("Valmis.");
              }

              public void handleOpenError(ConnectionHandler handler, String errorMessage) {
                            OmaMidlet.getInstance().closeApp();
              }

              public void handleMessageWasSent(ConnectionHandler handler) {

              }

              public void handleReceivedMessage(ConnectionHandler handler, byte[]
                                                  messageBytes) {
                            OmaMidlet.getInstance().messageIn();
              }
```

```
		public boolean isConnected() {
			if (conn!= null) {
				if(!conn.isTerminated()) return true;
			}
			return false;
		}

		public void handleOpenSecurityError(ConnectionHandler handler, String error)
		{
			OmaMidlet.getInstance().getScreen().setMessage("Ei onnistu.");
		}

}
```

```
//Command-rajapinnan toteutus

import a4sp.gui.GuiCommand;


public class ExitCommand implements GuiCommand{

		public void execute() {
			OmaMidlet.getInstance().closeApp();
		}

}
```

```
//Command-rajapinnan toteutus

import a4sp.gui.GuiCommand;


public class SendCommand implements GuiCommand{

		public void execute() {
			OmaMidlet.getInstance().networkSend();
		}

}
```

```
//Command-rajapinnan toteutus

import a4sp.gui.GuiCommand;


public class StartCommand implements GuiCommand{

		public void execute() {
			OmaMidlet.getInstance().startBluetooth();
		}

}
```

```
//OmaScreen-luokka
//
//Esimerkkisovelluksen GenericScreen-toteutus

import java.io.IOException;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;
import javax.microedition.midlet.MIDlet;

import a4sp.gui.GenericScreen;
import a4sp.gui.GuiCommand;
import a4sp.gui.GuiDigitalDisplay;
import a4sp.gui.GuiGauge;
import a4sp.gui.GuiPopupMessage;
import a4sp.gui.GuiScrollMenu;



public class OmaScreen extends GenericScreen implements CommandListener{

              private GuiScrollMenu menu;
              private GuiGauge gauge1;
              private GuiDigitalDisplay display1;
              private GuiDigitalDisplay display2;
              private GuiPopupMessage message;

              public OmaScreen(boolean arg0, MIDlet mid) {
                            super(arg0, null, mid);


                            menu = new GuiScrollMenu(this.getWidth()*3/4,
                                          this.getHeight(), 0);
                            try {
                                          GuiCommand exit = new ExitCommand();
                                          GuiCommand start = new StartCommand();
                                          GuiCommand send = new SendCommand();
                                          Sprite s = new Sprite(
                                                        Image.createImage("/nappi.png"),69,33);
                                          s.setPosition(getWidth()-69,0);
                                          Sprite s2 = new Sprite(
                                                        Image.createImage("/nappi.png"),69,33);
                                          s2.setPosition(getWidth()-69-7,33);
                                          Sprite s3 = new Sprite(
                                                        Image.createImage("/nappi.png"),69,33);
                                          s3.setPosition(getWidth()-69-14,66);
                                          menu.addSelection(OmatStringit.quit, s, exit);
                                          menu.addSelection(OmatStringit.start, s2, start);
                                          menu.addSelection(OmatStringit.send, s3, send);
                            } catch (IOException e) {
                            }

                            display1 = new GuiDigitalDisplay(
                                                3,this.getWidth()/5,this.getHeight()/4+90);
                            display2 = new GuiDigitalDisplay(
                                                3,this.getWidth()*4/5,this.getHeight()/4+90);
                            display2.setTitle(OmatStringit.best);
                            gauge1 = new GuiGauge(this.getWidth()/4,this.getHeight()/4);
                            gauge1.setSmoothMode(false);
                            message = new GuiPopupMessage(
                                      false, getWidth()/2,getHeight()-15,getWidth()-20,30);

                            this.addComponent(display1);
                            this.addComponent(display2);
                            this.addComponent(gauge1);
                            this.addComponent(menu);
                            this.addComponent(message);

                            this.setCommandListener(this);
              }
```

```
		protected void keyPressed(int keyCode) {
			int action = getGameAction(keyCode);
			if (action == FIRE) {
				menu.getSelectedItem().select();
			}
			else if (action == DOWN) {
				menu.moveDown();
			}
			else if (action == UP) {
				menu.moveUp();
			}
			else if(action == LEFT) {
			}
			else if (action == RIGHT) {
				OmaMidlet.getInstance().reactionTime();
			}
			this.updateScreen();
			this.draw();
		}

		public void setMessage(String m) {
			message.activate(m);
			updateScreen();
			draw();
		}

		public void setGauge(int angle) {
			gauge1.setAngle(angle);
		}

		public void setReactionTime(int time) {
			display1.setValue(time);
		}

		public void setRecord(int rec) {
			display2.setValue(rec);
			updateScreen();
			draw();
		}

		public void commandAction(Command arg0, Displayable arg1) {
			// TODO Auto-generated method stub

		}

}
```

```
//RecordManager-luokka
//
//Esimerkkisovelluksen RmsHandlerListener-toteutus

import a4sp.recordstore.RmsHandler;
import a4sp.recordstore.RmsHandlerListener;
import a4sp.util.Converter;

public class RecordManager implements RmsHandlerListener{
            public static RecordManager instance;

            public static final int ENNATYS = 1;
            public static final int NIMI = 2;


            private RmsHandler handler;
            private int[] recordsToRead = {0};
            private Object[] values;

            private RecordManager() {
                        instance = this;
                        handler = new RmsHandler(this,"OmaRecord");
            }

            public boolean writeRecord(int type,int value) throws RecordManagerException{
                        byte[] data = new byte[4];
                        Converter.writeInt(data, value);
                        return handler.addToRecordStore(type, data, true, true, true);
            }

            public boolean writeRecord(int type,String value) throws
                                                            RecordManagerException{
                        byte[] data = value.getBytes();
                        return handler.addToRecordStore(type, data, true, true, true);
            }

            public boolean readRecords(int[] recs) throws RecordManagerException{
                        if (recs == null) return false;
                        recordsToRead = recs;
                        values = new Object[recs.length];
                        return handler.processRecordStore();
            }

            public Object[] getReadValues() {
                        return values;
            }

            public boolean recordStoreExists() {
                        return handler.processRecordStore();
            }

            public boolean deleteEverything() {
                        return handler.deleteRecordStore();
            }

            public void handleRecord(int type, byte[] content) {
                        int index = -1;
                        for(int i=0;i<recordsToRead.length;i++) {
                                    if(type==recordsToRead[i]) {
                                                index = i;
                                    }
                        }
                        if (index == -1) return;

                        switch(type) {
                                    case ENNATYS:
                                                values[index] = new
                                                 Integer(Converter.readInt(content));
                                                break;
                                    case NIMI:
                                                values[index] = new String(content);
                                                break;
                                    default:
                                                values[index] = null;
                                                break;
                        }
            }
```

```
                public void handleRecordAdded() {

                }


                public void handleRecordDuplicateError() {

                }


                public void handleRecordStoreDoesNotExist() {

                }


                public void handleRecordStoreFullError() {
                        OmaMidlet.getInstance().getScreen().setMessage(
                                                                OmatStringit.recerr);
                }


                public void handleRecordTypeExistsError() {

                }

                public static RecordManager getInstance() {
                        if (instance == null) {
                                instance = new RecordManager();
                        }
                        return instance;
                }

                public class RecordManagerException extends Exception {
                }


}
```

```
//Esimerkkisovelluksen lokalisoitavat
//sovelluskohtaiset merkkijonot

public class OmatStringit {
                public static String quit = "lopeta";
                public static String start = "aloita";
                public static String send = "lähetä";
                public static String best = "paras";
                public static String loading = "ladataan...";
                public static String now = "nyt!";
                public static String recerr = "RMS Tallennus ei onnistu.";
                public static String noAudio = "Laite ei tue äänien käyttöä";
}
```