TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Tietokonetekniikan suuntautumisvaihtoehto

Jukka Haavisto

JAVALLA TOTEUTETTU TIEDONSIIRTO-OHJELMA

Työn valvoja Yliopettaja Kai Poutanen

Tekijä: Jukka Haavisto
Työn nimi: Javalla toteutettu tiedonsiirto-ohjelma
Päivämäärä: 20.11.2006
Sivumäärä: 39
Hakusanat: Java, javax.bluetooth, javax.comm
Koulutusohjelma: Tietotekniikka
Suuntautumisvaihtoehto: Tietokonetekniikka

Työn valvoja: Yliopettaja Kai Poutanen

Tässä työssä oli tarkoituksena tutkia eri tiedonsiirtoprotokollia ja suunnitella tiedostonsiirto-ohjelma, jolla dataa voidaan siirtää eri tiedonsiirtokanavia pitkin kahden päätelaitteen välillä. Käytännössä päätelaitteet ovat tavallisia koti-PC:itä, joissa on tarvittava Java-ympäristö sekä tiedonsiirtoon tarvittavat tietoliikenneportit. Ohjelma on tehty Java-ohjelmointikielellä ja ohjelman käyttöä ohjataan graafisesta käyttöliittymästä. Käytettävät tiedonsiirtokanavat ovat IP-verkko, sarja- ja rinnakkaisporttien kautta tapahtuva tiedonsiirto sekä langaton tiedonsiirto Bluetooth-yhteyttä pitkin.

Siirrettäessä dataa IP-verkkoa pitkin ohjelma käyttää normaalia TCP/IP-protokollaa. Kyseessä on perinteinen asiakas–palvelin-malli, joka on toteutettu Javan Socket- ja ServerSocket-rajapintoja käyttämällä. Sarja- ja rinnakkaisportteja käytettäessä tiedonsiirtoprotokollana ovat asynkronisen sarjamuotoisen datansiirron ja asynkronisen rinnakkaisen datansiirron protokollat, joita käytetään Javan javax.comm-lisäpaketin määrittelemien valmiiden rajapintojen kautta. Langaton tiedonsiirto Bluetooth-yhteyttä pitkin noudattaa RFCOMM-protokollaa, johon päästään käsiksi javax.bluetooth-lisäpaketin määrittelemien rajapintojen kautta.

Author:	Jukka Haavisto
Name of the thesis:	Java Filetransfer Program
Date:	20.11.2006
Pages:	39
Keywords:	Java, javax.bluetooth, javax.comm
Degree programme:	Computer Systems Engineering
Specialization:	Computer Engineering
Supervisor:	Principal Lecturer Kai Poutanen

This final thesis consists of investigating different kind of data transfer protocols and file transfer program that was programmed using Java programming language. The basic idea is transferring files between two separate data terminals using different kinds of connecting medias. In practice, data terminals are normal home PCs that have proper java support and all the necessary communication ports like LAN, serial- and parallel ports and Bluetooth capability.

When using IP network as a connecting media between data terminals, transferring files is done by following normal TCP/IP protocol. This is handled by using common client-server principle and by using Socket and ServerSocket interfaces provided by Java. Serial and parallel communication uses basic asynchronous serial- and parallel datatransfer protocols, which are used via interfaces provided by javax.comm extension package. Bluetooth communication is using RFCOMM protocol which is accessed through an interface provided by javax.bluetooth extension package.

SISÄLLYSLUETTELO

SISÄLLYSLUETTELO I
TERMIT JA LYHENTEET II

1 JOHDANTO 1
2 PROTOKOLLAT 1
2.1 TCP/IP-protokollat 2
2.1.1 IP-protokolla 2
2.1.2 TCP-protokolla 3
2.2 Asynkroninen datansiirto 5
2.3 IEEE-1284-standardi 6
2.4 Bluetooth ja RFCOMM-protokolla 6
2.5 Big/Little endian 8
3 KÄYTTÖYMPÄRISTÖ 8
4 OHJELMAN ASENNUSOHJE 9
5 OHJELMAN KÄYTTÖOHJE 10
6 LÄHDEKOODIN RAKENNE JA LUOKAT 14
6.1 Kirjastot 14
6.2 Pääohjelmaluokka 14
6.3 Käyttöliittymäluokka 15
6.3.1 Rakentaja 15
6.3.2 Progress-luokka 16
6.3.3 Filelkm- ja filelengths-luokat 16
6.3.4 Browsefiles-luokka 17
6.3.5 LanIP-luokka 17
6.3.6 Kuuntelijaluokat 17
6.3.7 Bluetooth-apuluokat 18
6.3.7.1 DeviceDiscoverer-luokka 18
6.3.7.2 ServiceDiscoverer-luokka 21
6.3.8 Send- ja receive-luokat 24
6.3.8.1 Send-luokka 25
6.3.8.2 Receive-luokka 26
7 YHTEYDEN MUODOSTAMINEN 28
7.1 LAN-yhteys 28
7.2 Sarjaportti-yhteys 29
7.3 Rinnakkaisportti-yhteys 30
7.4 Bluetooth-yhteys 31
8 OHJELMAN TESTAUS 33
8.1 Testilaitteisto 33
8.2 Testitulokset 33
9 OHJELMAN KEHITYSEHDOTUKSIA 35

10 YHTEENVETO 35
LÄHDELUETTELO 38
LIITELUETTELO 39

KÄYTETYT LYHENTEET JA TERMIT

baud	Tietoliikennetekniikassa käytetty mitta, joka kuvaa tiedonsiirtonopeutta.
Big endian	Ilmaisee tavujärjestyksen, jossa eniten merkitsevä tavu esitetään ensin.
bluecove	Laajennuspaketti Javan kirjastoihin. Sisältää valmiita luokkia Bluetoothia varten, kuten javax.bluetooth ja javax.microedition.io.
Bluetooth	Langaton tiedonsiirtotekniikka.
CTS	Clear To Send. RS-232:n määrittelemä signaali.
C++	Oliopohjainen ohjelmointikieli.
DNS	Domain Name System. Internetin nimipalvelujärjestelmä.
ECP	Extended Capability Port. IEEE-1284-protokollan määrittelemä toimintamoodi.
EPP	Enhanced Parallel Port. IEEE-1284-protokollan määrittelemä toimintamoodi.
Firewire	Tietokoneen ulkoisten oheislaitteiden liitäntästandardi.
HCI	Host Controller Interface. Bluetooth-protokollapinon kerros.
IEEE-1284	Standardi tietokoneen rinnakkaisliitynnälle.
IP	Internet Protocol. Internetin käyttämä protokolla pakettikytkentäiseen kommunikointiin.
IPv4	Internet Protocol version 4. Yleisimmin käytössä oleva versio IP-protokollasta.
Java	Oliopohjainen ohjelmointikieli.
javax.comm	Lisäpaketti Javan kirjastoihin. Sisältää valmiita luokkia sarja- ja rinnakkaisliikennettä varten.
JSR-82	Java Specification Request-82. Standardi Bluetooth-ominaisuuksien sulauttamisesta Javaan.
j2sdk1.4.2	Java-käännösympäristön versio 1.4.2.
LAN	Local Area Network. Yleisesti käytössä oleva tietoliikenneverkko.

Little endian	Ilmaisee tavujärjestyksen, jossa vähiten merkitsevä tavu esitetään ensin.
LPT	Nimi liityntärajapinnalle tietokoneen rinnakkaisporttiin.
L2CAP	Locigal Link Control and Adaptation Protocol. Bluetooth-protokollapinon osa.
MP3	MPEG-1 –standardiin perustuva häviöllinen äänenpakkausmenetelmä**.**
Nibble mode	IEEE-1284 protokollan määrittelemä toimintamoodi.
OBEX	Object Exchange. Protokolla objektien siirtoon laitteiden välillä.
RFCOMM	Radio Frequency Communication. Bluetooth-protokolla, joka emuloi RS232-yhteyttä.
RS-232	Tietoliikenneprotokolla sarjamuotoisen datan siirtämiseen.
RTS	Request To Send. RS-232-standardin määrittelemä signaali.
SDP	Service Discovery Protocol. Protokolla Bluetooth-laitteiden palvelujen havainnointiin.
SPP	Serial Port Profile. Profiili, jonka avulla Bluetooth-laitteet voivat kommunikoida keskenään.
TCP	Transmission Control Protocol. Tietoliikenneprotokolla yhteyksien muodostamiseen Internet-tietokoneiden välillä.
USB	Universal Serial Bus. Tietokoneen ulkoisten oheislaitteiden liitäntästandardi.
8N1	Lyhenne datan esitysmuodolle asynkronisessa sarjaliikenteessä.

1 JOHDANTO

Tietotekniikan kehittyessä ohjelmistojen ja niitä käyttävien järjestelmien koot ovat kasvaneet nopeasti. Ohjelmistojen kasvun myötä laskentatehon tarpeeseen on jouduttu kiinnittämään yhä enemmän huomiota. Usein tätä ongelmaa on lievitetty hajauttamalla ohjelmistot useamman laskentayksikön kesken. Toinen syy hajauttamiselle on tarve kontrolloida useita erillisiä laitteita yhdestä paikasta tai mahdollistaa laitteiden keskinäinen kommunikointi. Esimerkiksi suurissa tehtaissa saattaa olla suuri määrä laitteita ja järjestelmäkokonaisuuksia, jotka kommunikoivat keskenään. Jotta kommunikointi onnistuisi, täytyy valita käyttökohteeseen sopiva tiedonsiirtomekanismi ja protokolla.

Tässä työssä oli tarkoituksena tutkia erilaisia tiedonsiirtoprotokollia ja sitä, miten Java-ohjelmointikielellä voidaan kirjoittaa ohjelmia, joilla luodaan tietoliikenneyhteyksiä kahden eri koneen välillä. Tätä varten suunniteltiin Filetransfer-ohjelma, jolla on mahdollista siirtää dataa eri tiedonsiirtoteitä pitkin kahden eri koneen välillä. Java-ohjelmointikieleen päädyttiin siksi, että sillä on helppo toteuttaa graafisia käyttöliittymiä sisältäviä ohjelmia ja että se on laitteistojen tehokkuuksien kasvaessa lisännyt suosiotaan jopa aivan pienimmissäkin käyttökohteissa.

2 PROTOKOLLAT

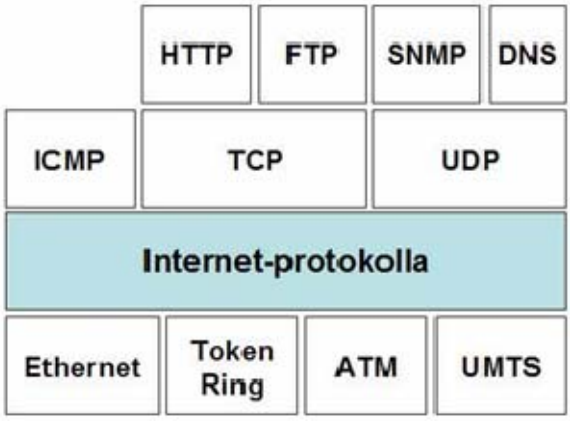
Protokollia tarvitaan mm. siirrettäessä dataa verkotettujen laitteiden välillä. Protokolla määrittelee mallin, miten kommunikaatio tulee hoitaa kahden eri laitteen välillä molempien ymmärtämällä tavalla. Filetransfer-ohjelma toimii neljällä eri tiedonsiirtoprotokollalla, joita tarkastellaan tässä luvussa hieman tarkemmin. Ensimmäisenä selvitetään TCP/IP-protokollien toimintaperiaatetta siirrettäessä tietoa IP-verkossa. Sen jälkeen tarkastellaan sarja- ja rinnakkaisporttiliikenteen käyttämiä protokollia ja lopuksi tutustutaan Bluetooth-yhteydessä käytössä olevaan RFCOMM-protokollaan.

2.1 TCP/IP-protokollat

TCP/IP on usean tietoverkkoprotokollan yhdistelmä, jota käytetään Internet-liikennöinnissä. Tästä IP-protokolla on alemman tason protokolla, joka vastaa päätelaitteiden osoitteistamisesta ja pakettien reitittämisestä verkossa. Sen päällä voidaan ajaa useita muita verkko- tai kuljetuskerroksen protokollia, joista TCP-protokolla on yleisin. Sen tehtävänä on vastata kahden päätelaitteen välisestä tiedonsiirtoyhteydestä, pakettien järjestämisestä ja hukkuneiden pakettien uudelleenlähetyksestä. [11.]

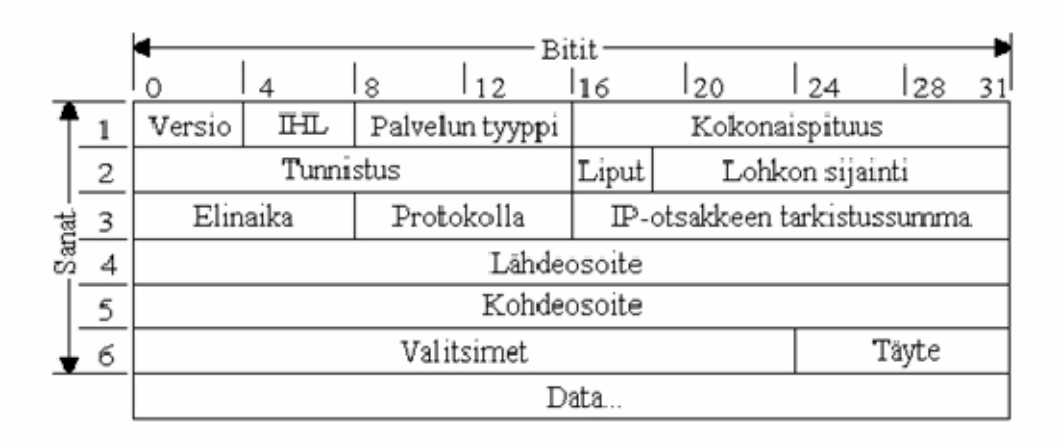
2.1.1 IP-protokolla

”IP-protokolla on yhteydetön protokolla joka vastaa IP-pakettien kuljetuksesta verkossa. Sen tärkeimpiä tehtäviä ovat pakettien fragmentointi eli osioiminen, liikenteen reititys IP-osoitteen perusteella, peruspaketin koon määrittäminen kyseisessä verkossa ja optioiden käyttö pakettien yhteydessä.” [8.] ” IP-protokollaa voidaan ajaa lähes minkä tahansa verkon päällä, joten sillä on helppo yhdistää erilaisia verkkoja isommiksi kokonaisuuksiksi. Internet on vain yksi, mutta merkittävin tällä tavalla rakentunut verkko.” [8.] IP-protokollan sijainti protokollapinossa on esitetty kuvassa 1.



Kuva 1 IP-protokollan sijainti protokollapinossa. [11.]

Tällä hetkellä IP-protokollasta on käytössä versio IPv4, joka määrittelee jokaiselle tietokoneelle yksilöllisen 32-bittisen IP-osoitteen. Osoite esitetään neljänä desimaalilukuna pisteillä erotettuna. Tämän osoitteen perusteella verkossa olevat reitittimet osaavat ohjata lähetetyt paketit oikeisiin osoitteisiin. Osoite löytyy jokaisesta paketista protokollan määräämästä kohdasta. Tarkempi IP-paketin rakenne on esitetty kuvassa 2.



Kuva 2 IP-paketin rakenne ja kenttien pituudet. [11.]

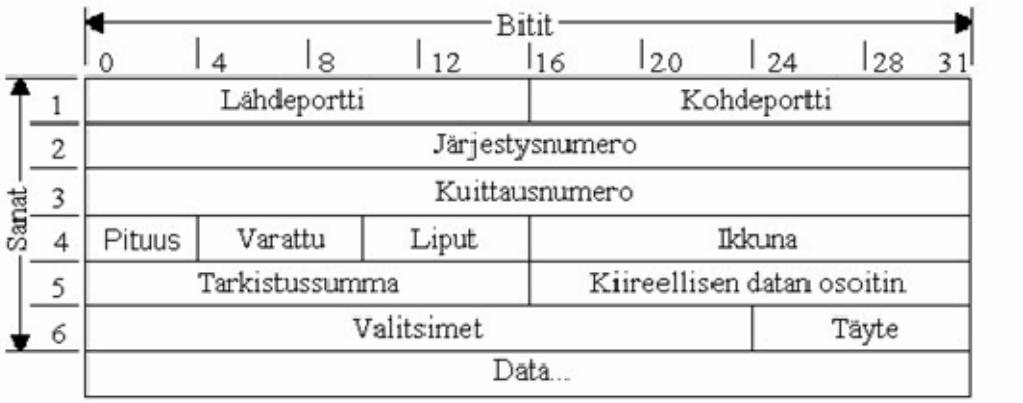
Tässä työssä olennaisimmat kentät, joihin voidaan koodissa vaikuttaa, ovat osoitekentät ja itse data. Näiden kenttien näkyminen kooditasolla esitellään myöhemmin.

IP-protokollasta puhuttaessa vastaan tulee myös DNS-palvelu. Koska IP-protokolla ei tunnista domain-nimiä osoitteena, tulee nämä nimet muuttaa ns. ”dotted desimal”-muotoon. Tässä työssä DNS-nimipalvelua ei ole toteutettu ohjelmaan, joten yhteyden muodostuksessa tullaan käyttämään suoraan numeerista osoitetta.

2.1.2 TCP-protokolla

Toisin kuin IP-protokolla TCP-protokolla on yhteydellinen protokolla. Se oikeastaan täydentää IP-protokollaa ja sen tehtävänä on taata luotettava

tiedonsiirto. Se huolehtii IP-pakettien kuljettamisesta ja tarvittaessa niiden uudelleen lähetyksestä sekä oikeasta järjestyksestä. Tämä tapahtuu siten, että vastaanottaja lähettää jokaisesta saamastaan viestistä (paketista) kuittauksen lähettäjälle, jossa se kertoo, kuinka paljon tietoa se on ottanut vastaan. Jos lähettäjä ei saa kuittausta viestiinsä tai jos ilmoitettu tietomäärä on liian pieni, matkalla hukkuneet tiedot lähetetään uudelleen.[8.], [9.] TCP-paketin rakenne on esitetty kuvassa 3.



Kuva 3 TCP-paketin rakenne ja kenttien pituudet. [8]

TCP-yhteys muodostetaan kolmivaiheisen kättelyn tuloksena, jossa aloittava kone lähettää segmentin, jossa SYN-bitti on asetettu. Vastaanottava laite tietää lähettäjän haluavan avata yhteyden ja siirtyy tilaan SYN-RECEIVED. [8.] ”Yhteydenottopyynnön saatuaan laite tarkistaa, onko sillä resursseja uuteen yhteyteen. Mikäli on, laite lähettää ensimmäisen sanoman lähettäjälle segmentin, jossa SYN- ja ACK-bitit on asetettu. Tämä ilmaisee, että yhteys halutaan muodostaa myös tästä päästä ja että edellinen sanoma tuli perille.” [8.] ”Viimeisenä aloitteen tehnyt kone lähettää yhteyden varmistuspaketin, jossa ACK-bitti on asetettu. SYN-bitti on nollassa, joka osoittaa, että yhteys on muodostunut. Kaikissa seuraavissa segmenteissä ACK-bitti on asetettuna ja SYN-bitti nollana.” [8.]

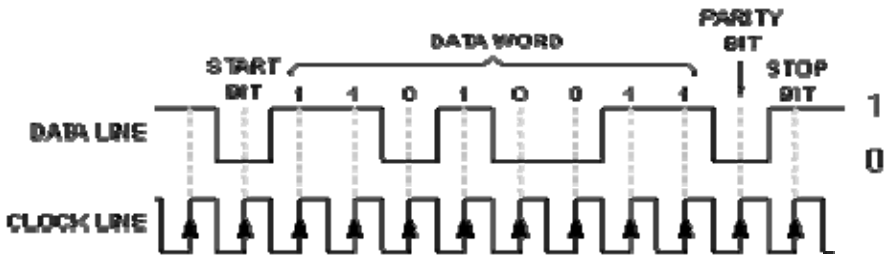
Tässä työssä näistä olennaisimmat ohjelmoijan vastuulla olevat kentät ovat lähdeportti, kohdeportti ja itse siirrettävä data.

2.2 Asynkroninen datansiirto

Asynkronista datansiirtoa käytetään, kun siirretään dataa esim. kahden laitteen välillä sarjaporttien kautta. Kaapeleita kyseiseen tarkoitukseen on erilaisia, mutta yleisin niistä on kaapeli, jossa on 9-napainen liitin. Itse dataa liikkuu näistä ainoastaan kahdessa. Toinen on lähetystä varten ja toinen vastaanottoa varten. Muut johtimet ovat yhteyden kontrollointia ja maadoitusta varten. Data lähetetään kehyksissä, joita on käytössä muutamia erilaisia käyttötarkoituksen mukaan.

”Yleisin nykyisin käytetty siirtoprotokolla on '8N1' mikä tarkoittaa, että tietoa siirretään paketeissa, joihin sisältyy 8 databittiä, ei yhtään (None) pariteettibittiä ja yksi stop-bitti. Kun siirrettävä paketti aloitetaan aina start-bitillä, on paketin kokonaispituus 10 bittiä. Uudempien laitteiden ja ohjelmien mukana ovat tulleet tätäkin korkeammat siirtonopeudet n. 1Mb:iin asti. Nämä vaativat kuitenkin yleensä erikoislaitteita.” [6.]

Myös tässä työssä on käytössä ns. 8N1-protokolla, ja tiedonsiirtonopeudeksi on määritelty vakio 9600 bittiä sekunnissa. Kuvassa 4 on havainnollistettu käytettyä protokollaa ja sen kehysrakennetta.



Kuva 4 Asynkronisen datansiirron 8N1 kehysmalli. [3.]

Kuvasta 4 nähdään esimerkki, millaisena data lähetetään siirtotielle. Datakehys siis koostuu kymmenestä bitistä, joista keskimmäiset kahdeksan sisältävät itse informaation. Vastaanottopäässä bitit luetaan kuvan osoittamalla tavalla kellopulssin tahdissa keskeltä bittiä.

2.3 IEEE-1284-standardi

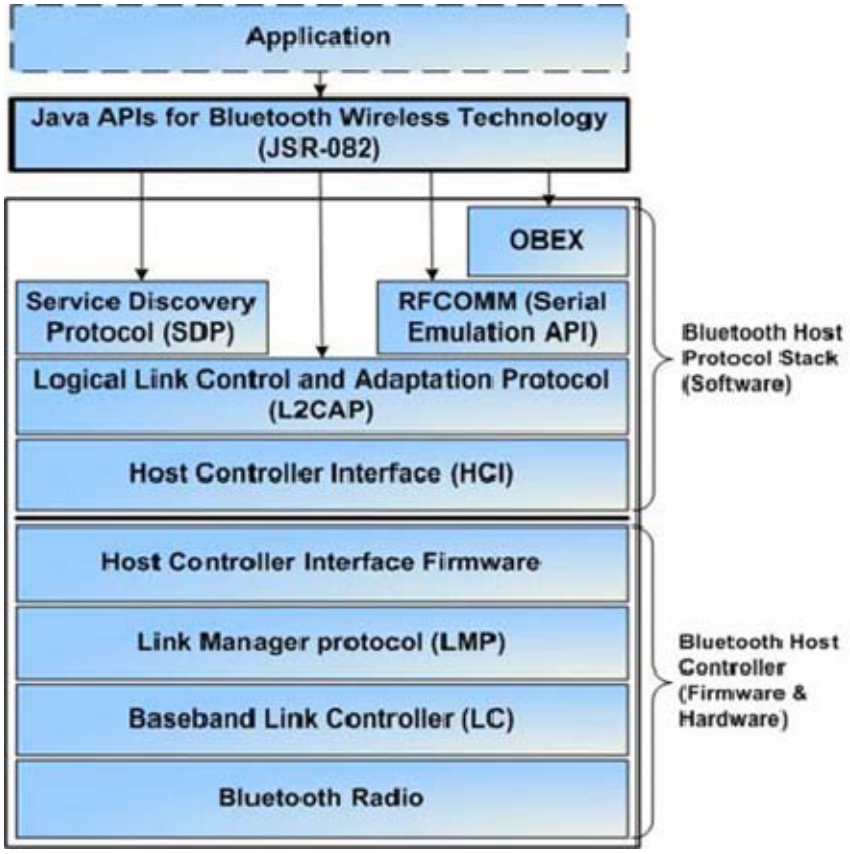
IEEE-1284 tarkoittaa standardia, joka määrittelee rinnakkaisen kaksisuuntaisen kommunikaation kahden laitteen välillä. Rinnakkaisuuden ansiosta tämä siirtotapa on paljon nopeampi kuin asynkroninen sarjaliikenne. Teoreettinen maksimitiedonsiirtonopeus on jopa 4 000 000 bittiä sekunnissa, mutta käytännössä raja liikkuu kahden miljoonan paikkeilla. Myös IEEE-1284 määrittelee erilaisia liittimiä, joita on kolme kappaletta: tyypin A, B ja C-liittimet. [12.] Näistä perinteisesti kotimikroissa käytetään tyypin A-liitintä, jossa on 25 napaa. Näistä kahdeksan on varattu databiteille ja loput yhteyden kontrollointia ja maadoitusta varten.

IEEE-1284-standardi toimii viidellä eri toimintamoodilla. Nämä ovat SPP-, Nibble-, Byte mode-, EPP- ja ECP-toimintamoodit. Näistä kolme ensimmäistä ovat yksisuuntaisia toimintamoodeja ja kaksi jälkimmäistä kaksisuuntaisia. [12.] Tämä tarkoittaa sitä, että jos halutaan sekä lähettää että vastaanottaa dataa, pitää käytössä olla joko EPP- tai ECP-toimintamoodi.

2.4 Bluetooth ja RFCOMM-protokolla

Neljäntenä tiedonsiirtoyhteytenä ohjelma käyttää Bluetooth-yhteyttä. Bluetooth on avoin standardi langattomalle tiedonsiirrolle joka toimii maksimissaan 100 metrin etäisyyksillä. Bluetooth-radioyhteys toimii alueella 2,4 GHz−2,4835 GHz ja se hyödyntää taajuushyppelytekniikkaa. Kommunikointia varten sillä on myös oma protokollapino. [10.] Bluetooth-protokollapino sisältää sekä erityisesti Bluetoothiin määritettyjä että yleisempiä protokollia. Bluetooth soveltuu käytettäväksi monissa erilaisissa laitteissa, jolloin kukin laite käyttää protokollista vain tiettyä osaa. L2CAP (Logical Link Control and Adaptation Protocol) kanavoi protokollat ja useita protokollia voidaan käyttää samassa järjestelmässä. [10.]

Kuva 5 havainnollistaa protokollapinon rakennetta.



Kuva 5 Bluetooth-protokollapino. [7.]

Kuvan 5 neljä alinta protokollapinon osaa kuuluvat laitteistotasolle ja sisältävät tarvittavan tiedon yhteyden muodostukseen ja hallintaan. Ylöspäin mentäessä seuraavana tulee HCI-kerros, jonka tehtävänä on tarjota rajapinta protokollapinon ylempien osien sekä Bluetooth-laitteen fyysisten ominaisuuksien välille. HCI-kerroksen päällä sijaitsee L2CAP-kerros. Sen tehtävä on yhdistää erilaiset yläpuolella olevat protokollat samalle siirtotielle ja kätkeä alemman kerroksen fyysisiä ominaisuuksia ylemmiltä kerroksilta. Seuraavaksi pinossa on kolme muuta protokollaa, joita ovat SDP-, RFCOMM- ja OBEX-protokollat. SDP-protokollaa käytetään etsittäessä palveluita muista Bluetooth-laitteista. Kaksi muuta protokollaa ovat Bluetooth-protokollapinoon lainattuja yleisempiä protokollia. OBEX-protokollaa käytetään objektien siirtoon ja RFCOMM-protokollaa yleisesti datan siirtoon. [7.], [13.], [14.] Tässä työssä data siirretään juuri RFCOMM-protokollaa hyödyntämällä.

RFCOMM-protokolla on L2CAP-kerroksen päällä toimiva yksinkertainen tiedonsiirtoprotokolla, joka emuloi perinteistä asynkronista sarjaliikennettä. Se toimii sitä käyttäville applikaatioille rajapintana langattomaan tiedonsiirtoon Bluetooth-yhteyden yli. [13.] RFCOMM toimii asynkronisen sarjasiirron määrittelemällä tavalla mahdollistamalla ohjelmistona sarjaportin kaikki fyysiset signaalit sitä käyttäville sovelluksille [16.]. RFCOMM-protokolla yhdessä sitä käyttävän Bluetooth-profiilin, SPP:n, kanssa muodostaa tässä työssä käytetyn tiedonsiirtoon tarvittavan kommunikointimallin.

2.5 Big/Little endian

Kun tietoa siirretään kahden eri paikan ja usein eri laitteistojen välillä, tulee mieleen näiden laitteiden yhteensopivuus. Käytännössä C:llä tai C++:lla kirjoitetut ohjelmistot olisi hyvä suunnitella siten, että ne osaisivat ottaa huomioon eri laitteistoissa käytettävän tavujärjestyksen. Tavujärjestys tarkoittaa sitä, miten prosessori tallettaa yli tavun mittaiset sanat muistiin. On käytössä Little-Endian- (esim. Intel) ja Big-Endian (esim. Motorola ja Sparc) -tavujärjestys. Jotta sekaannusta ei tulisi, on ennen verkkoon lähetystä tavut järjestettävä ns. verkon tavujärjestykseen, joka on sama kuin Big-Endianin. Tällöin vältytään ongelmilta, jotka aiheutuisivat väärästä tavujärjestyksestä. Koska tässä työssä ohjelmointikielenä on Java, tästä ei tarvitse erikseen välittää. Java säilyttää sisäisesti dataa aina verkon tavujärjestyksessä, jolloin ohjelmoijan ei tästä tarvitse välittää. Tämä noudattaa täysin Javan ideologiaa ”kirjoita kerran, aja kaikkialla”.

3 KÄYTTÖYMPÄRISTÖ

Koska tiedostonsiirto-ohjelma on kirjoitettu Javalla ja se osaa hyödyntää datan siirtoa sarja- ja rinnakkaisporttia pitkin sekä Bluetooth-yhteyttä pitkin, tämä asettaa käyttöympäristölle omat rajoituksensa. Täydellisesti toimiakseen ohjelma vaatii tietokoneelta Windows-käyttöjärjestelmän sekä Java-käännösympäristön esim. j2sdk1.4.2. Lisäksi tarvitaan kaksi laajennuspakettia Javan kirjastoihin. Sarja- ja rinnakkaisporttiliikennettä varten pitää asentaa

javax.comm-laajennuspaketti, joka tarjoaa tarvittavat rajapinnat sarja- ja rinnakkaisporttien käyttöön ja niiden kautta tapahtuvaan liikenteeseen. Lisäksi tarvitaan Bluetooth-yhteyden hallintaa varten erikseen asennettava bluecove-laajennuspaketti. Tämä tunnetaan myös nimellä JSR-82. Siinä on valmiit luokat Bluetooth-protokollapinon käyttöön, jonka kautta saadaan suora yhteys itse Bluetooth-laitteeseen. Ohjelmistojen lisäksi tietokoneessa pitää olla verkkokortti, sarja- ja rinnakkaisportti sekä joko sisään integroitu Bluetooth tai ulkoisesti USB-väylään asennettava Bluetooth-sovitin.

4 OHJELMAN ASENNUSOHJE

Tietokoneessa, johon Filetransfer-ohjelmaa asennetaan, tulee olla Windows XP -käyttöjärjestelmä tai uudempi. Tämän jälkeen pitää asentaa Java-käännösympäristö. Tämän saa ladattua ilmaiseksi osoitteesta http://java.sun.com. Seuraavaksi on asennettava joko javax.comm- tai bluecove-laajennuspaketit. Javax.comm voidaan ladata osoitteesta http://java.sun.com/products/javacomm/downloads/index.html tai ottaa se mukana tulevista liitetiedostoista. Tämä paketti sisältää useita tiedostoja, mutta välttämättömiä tiedostoja on vain kolme, jotka ovat javax.comm properties, win32com.dll ja comm.jar. Win32com.dll-tiedosto tulee kopioida asennetun Java-käännösympäristön bin-hakemistoon esim. C:\j2sdk1.4.2\bin. Javax.comm properties- ja comm.jar-tiedostot tulee kopioida käännösympäristön lib-hakemistoon esim. C:\j2sdk1.4.2\lib.

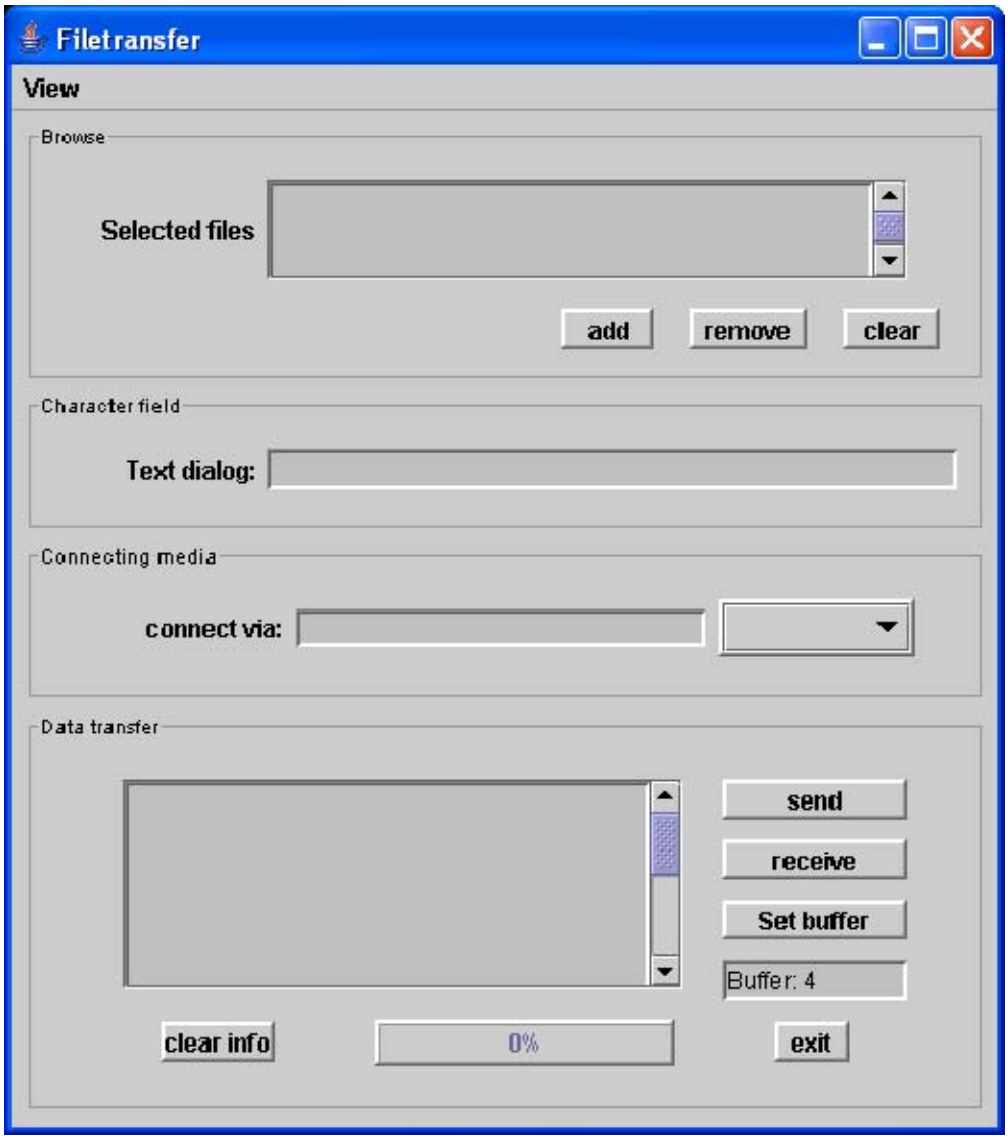
Kun nämä tiedostot on kopioitu oikeisiin hakemistoihin, Java Virtuaalikoneelle pitää kertoa näiden tiedostojen olemassaolosta ja sijainnista. Helpoimmin tämä käy muokkaamalla ympäristömuuttuja asetuksia. Nämä asetukset löytyvät ohjauspaneelista kohdasta järjestelmä, josta taas valitaan lisäasetukset. Lisäasetukset-välilehdeltä valitaan ympäristömuuttujat. Tällä välilehdellä näkyy kaksi eri kenttää, jotka ovat käyttäjän muuttujat sekä ympäristömuuttujat. Jo olemassa olevaa ympäristömuuttujaa nimeltä PATH pitää muokata ja lisätä loppuun

teksti ;C:\JDK1.1.8\BIN\. Seuraavaksi järjestelmämuuttujiin pitää lisätä uusi muuttuja nimeltä CLASSPATH johon tulee lisätä teksti ;C:\JDK1.1.8\LIB\COMM.JAR. [17.]

Bluecove-laajennuspaketin asentaminen on hieman yksinkertaisempaa. Hakemistoon C:\JDK1.1.8\ täytyy kopioida purettu bluecove-paketti jolloin saadaan uusi hakemistorakenne C:\JDK1.1.8\BlueCoveWinXP. Seuraavaksi pitää kopioida tiedosto intelbth.dll kirjastohakemistoon esim. C:\WINDOWS\ SYSTEM32\. Tämän jälkeen on lisättävä BlueCove.jar-tiedosto CLASSPATH-muuttujaan. Tämä tapahtuu vastaavalla tavalla kuten aiemmin eli lisätään kyseinen tiedosto hakemistorakenteineen CLASSPATH-muuttujaan. Tässä tapauksessa lisätään muuttujan perään teksti ;C:\j2sdk1.4.2_08\BlueCoveWinXP\dist\lib\BlueCove-20050514.jar. [18.]

5 OHJELMAN KÄYTTÖOHJE

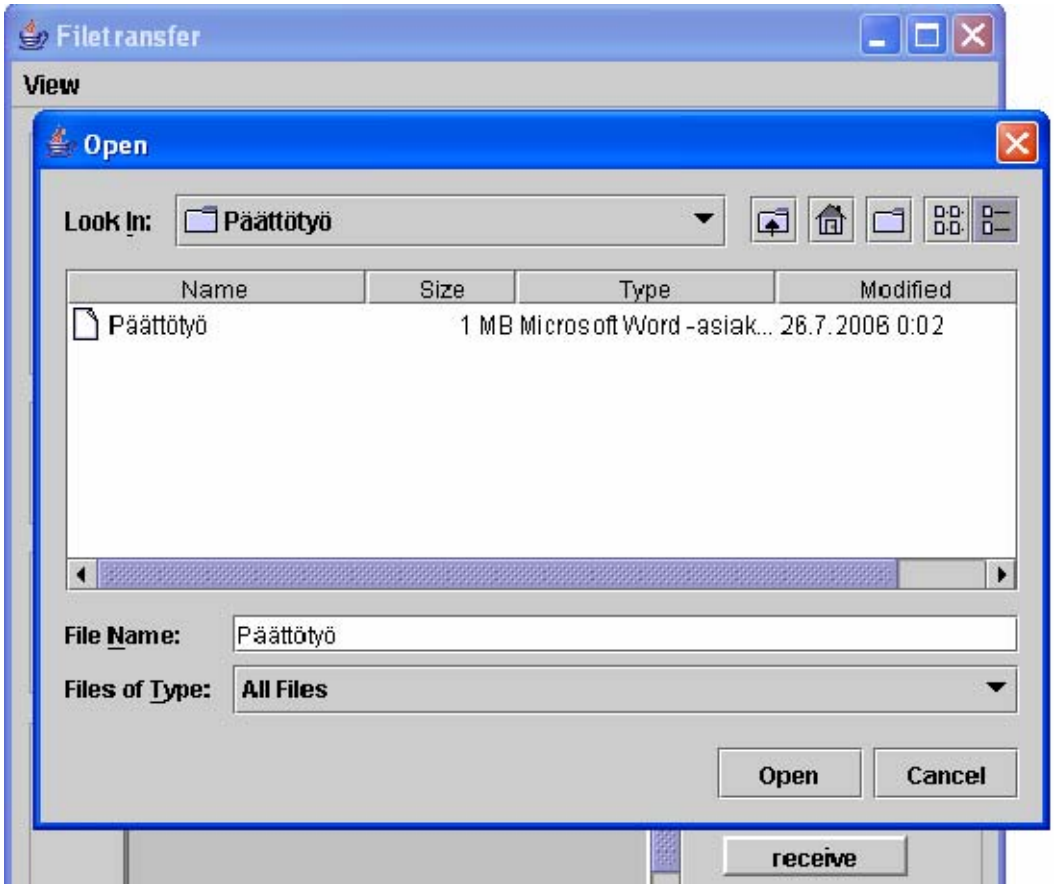
Ensimmäistä kertaa ohjelmaa käynnistettäessä ohjelma pitää ensin kääntää, jolloin siitä muodostetaan Java-tulkilla ajettavaa tavukoodia. Tämä tapahtuu komentoriviltä komennolla javac Filetransfer.java. Itse ohjelma käynnistetään komennolla java Filetransfer. Tämän jälkeen näytölle avautuu Filetransfer-ohjelman graafinen käyttöliittymä, joka on esitetty kuvassa 6.



Kuva 6 Filetransfer-ohjelman graafinen käyttöliittymä.

Käyttöliittymä sisältää neljä eri kenttää. *Browse*-, *Character field*-, *Connecting media*- ja *Data transfer* -kentistä löytyvillä toiminnallisuuksilla käyttäjä voi valita siirrettävän datan, siirtoon käytettävän tiedonsiirtotien ja joko lähettää tai vastaanottaa dataa.

Browse field -kentässä käyttäjä voi *add*-painikkeen avulla valita siirrettävät tiedostot, joita on maksimissaan 10 kappaletta. *Add*-painiketta painamalla avautuu valikko, josta käyttäjä voi valita tiedoston mistä tahansa hakemistosta, johon hänellä on pääsymahdollisuus. Valikko on esitetty kuvassa 7.

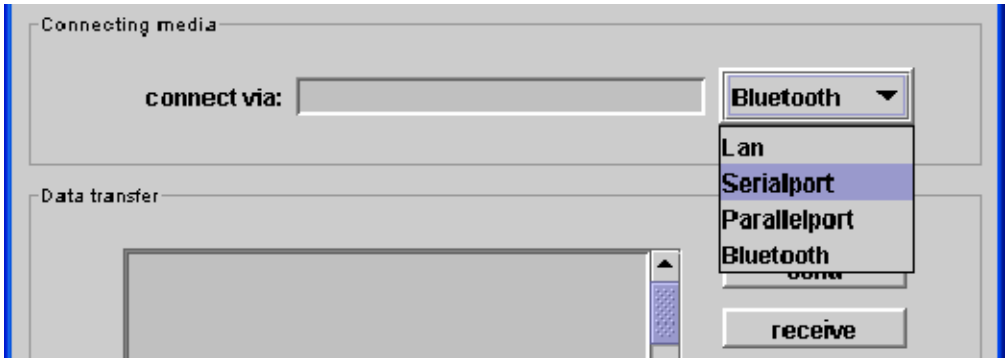


Kuva 7 Tiedostojen valinta Filetransfer-ohjelmalla.

Tämän jälkeen valitun tiedoston nimi näkyy kentässä olevassa *Selected files* -listassa. *Remove*-painikkeella käyttäjä voi poistaa viimeisimmän valitun tiedoston listasta ja *clear*-painikkeella voidaan tyhjentää koko lista kerralla.

Character field -kentässä käyttäjä voi kirjoittaa tekstiä sille varatulle alueelle. Lähetyksen yhteydessä tämä teksti lähetetään vastaanottajalle. Lähetyksen jälkeen teksti kirjautuu *Data transfer* -kentässä olevaan info-ruutuun. Vastaanottajalla tämä teksti tulee myös näkymään samaisessa info-ruudussa. Tekstin voi lähettää joko yksistään tai tiedostojen lähetyksen ohessa.

Connecting media -kentässä käyttäjä voi valita halutun tiedonsiirtokanavan. Kentän oikeassa reunassa on valikko, josta käyttäjä voi valita yhteydeksi joko *Lan, Serialport, Parallelport* tai *Bluetooth* kuvan 8 osoittamalla tavalla.



Kuva 8 Siirtotien valinta Filetransfer-ohjelmalla.

Data transfer -kentässä käyttäjä voi aloittaa joko datan lähetyksen tai vastaanoton. Lähetys tapahtuu painamalla nappia *Send*. Tätä ennen on oltava valittuna vähintään joko yksi tiedosto tai kirjoitettuna tekstiä *Character field* -kenttään. Lisäksi on oltava valittuna yksi neljästä tiedonsiirtotiestä. Jos näitä esiehtoja ei ole täytetty, ohjelma ilmoittaa siitä käyttäjälle. Jos tiedonsiirtotieksi on valittu LAN-yhteys, ohjelma kysyy käyttäjältä vastaanottajan IP-osoitetta. Osoite tulee antaa numeerisessa muodossa, koska ohjelmaan ei ole toteutettu nimipalvelimen käyttömahdollisuutta. Tiedostojen vastaanotto tapahtuu *Receive*-painikkeella. Tätä ennen on oltava valittuna käytettävä tiedonsiirtotie. Jos näin ei ole tehty, ohjelma antaa tästä käyttäjälle ilmoituksen. *Data transfer* -kentässä on myös *Set Buffer* –painike, jolla käyttäjä voi valita haluamansa lähetyspuskurin koon. Periaatteessa mitä suurempi puskurin koko on, sitä nopeammin tiedonsiirto tapahtuu. Puskurin koolle on kuitenkin käytännön rajoituksensa, esimerkiksi siirrettäessä dataa sarjaporttia pitkin koneessa olevalla I/O-piirillä on oma fyysinen puskurinsa, jonka kokoa ohjelmallinen puskuri ei saisi ylittää. Jos näin käy, tiedonsiirto voi hidastua ja siirrettävä tiedosto korruptoitua tai tiedonsiirto jopa katketa kokonaan. Käytännössä puskurin koon määrittäminen jää käyttäjän oman kokemuksen vastuulle. Puskurin koko on silloin hyvä, kun tiedonsiirtoa ei enää voisi nopeuttaa, vaikka puskurin kokoa lisättäisiinkin. Tähän vaikuttaa paljolti käytettävä laitteisto. Valittaessa tiedonsiirtoyhteyttä ohjelma kuitenkin antaa oman alkuarvon puskurin koolle. Nämä arvot on todettu toimiviksi ohjelmaa testattaessa. Lisäksi *Data transfer* -kentässä on painikkeet *Exit* ja *Clear info*. *Clear info* -painikkeella käyttäjä voi tyhjentää tapahtumahistorian inforuudusta. *Exit*-painikkeella poistutaan ohjelmasta. *Data transfer* -kentässä on myös

palkki, joka kertoo kulloisenkin tiedonsiirron statuksen prosentteina koko siirrettävästä datan määrästä. Graafisen käyttöliittymän yläreunassa on myös valikko View -> My IP-address, josta käyttäjä näkee oman koneensa IP-osoitteen numeerisessa muodossa.

6 LÄHDEKOODIN RAKENNE JA LUOKAT

Tässä luvussa esitellään tarkemmin ohjelman toimintaa kooditasolla ja tarkastellaan, miten tiedonsiirto on toteutettu eri yhteyksiä pitkin. Ohjelman perusrakenne on melko yksinkertainen. Aluksi luodaan graafinen käyttöliittymä kaikkine komponentteineen. Sen jälkeen on rakennettu muutama apuluokka, joita käytetään hyväksi tiedonsiirron alustukseen ja itse tiedonsiirtoon. Lopussa on kaksi luokkaa, *Send* ja *Receive*, joista ohjelma luo dynaamisesti säikeitä, jotka hoitavat itse datan lähetyksen ja vastaanoton. Käynnistyttyään säie hoitaa yhteyden muodostuksen ja huolehtii datan siirrosta apuluokkien avustuksella.

6.1 Kirjastot

Lähdekoodi on kirjoitettu tiedostoon nimeltä *Filetransfer.java*. Tiedoston alussa on esitelty kaikki koodin käyttämät kirjastot. Kirjastot tarjoavat suuren määrän valmiita luokkia ja metodeita joita ohjelmasta kutsutaan. Erikseen näistä mainittakoon työn alussa mainitut javax.comm- ja javax.bluetooth-kirjastot, jotka ovat olennainen osa luotaessa tiedonsiirtoyhteyksiä.

6.2 Pääohjelmaluokka

Kirjastojen sisällytyksen jälkeen koodissa on pääohjelmaluokka, joka on esitetty listauksessa 1.

```
public class Filetransfer  {
  public static void main (String[] args){
    Filetransf FT = new Filetransf();
    FT.nayta();
  }
}
```

Listaus 1 Lähdekoodi pääohjelmaluokasta.

Pääohjelmaluokassa *Filetransfer* luodaan *FT*-niminen olio käyttöliittymäluokasta *Filetransf*. Tämän jälkeen kutsutaan *Filetransf*-luokan metodia *public void nayta()*, joka asettaa luodun olion eli käyttöliittymän näkyviin.

6.3 Käyttöliittymäluokka

Loppuosuus koodista on käyttöliittymäluokkaa, joka sisältää sisäisiä luokkia sekä metodeita, joita käyttöliittymäluokka tarvitsee. Järjestyksessä alusta alkaen se sisältää graafisen käyttöliittymän luonnin rakentajassa *public Filetransf()*, käyttöliittymän näkyviin asettavan metodin *public void nayta()*, tiedonsiirron statuksen esittävän luokan *progress*, siirrettävien tiedostojen lukumäärän laskevan luokan *filelkm*, siirrettävien tiedostojen yhteiskoon laskevan luokan *filelengths*, tiedostojen valitsemisen hoitavan luokan *browsefiles* sekä LAN-yhteyden muodostukseen käytettävän IP-osoitteen selvittämiseen luokan *LanIP*. Lisäksi käyttöliittymäluokka sisältää kaksi käyttäjän laukaisemia tapahtumia kuuntelevaa kuuntelijaluokkaa *Buttonlistener* ja *connectionlistener* ja kaksi Bluetoothin käyttämää luokkaa *DeviceDiscoverer* ja *ServiceDiscoverer*. Viimeisenä käyttöliittymäluokassa on kaksi luokkaa *Send* ja *Receive*, jotka hoitavat itse tiedonsiirron.

6.3.1 Rakentaja

Käyttöliittymäluokan rakentajassa *public Filetransf* määritellään käyttöliittymän komponentit kuten itse kehys, nappulat, tekstikentät ja paneelit. Kaikki nappulat ja tekstikentät on sijoitettu paneeleihin, jotka taas on upotettu kehykseen. Näin on saatu luotua yhtenäinen kokonaisuus, joka näkyy käyttäjälle graafisena käyttöliittymänä. Graafinen käyttöliittymä asetetaan näkyväksi käyttöliittymäluokan metodilla *public void nayta()*, jossa kutsutaan java.awt-kirjaston *Frame*-luokan metodia *Show()*. Rakentajan ja graafisen käyttöliittymän näyttämisen yksinkertaisuuden vuoksi ei tarkempiin yksityiskohtiin tässä kohtaa mennä.

6.3.2 Progress-luokka

Progress-luokka on julkinen luokka, jonka tehtävänä on päivittää graafisessa käyttöliittymässä olevaa javax.swing-kirjaston komponenttia *JProgressBar*, joka kertoo käyttäjälle tiedostojen siirron vaiheen prosentuaalisena osuutena kokonaistiedonsiirrosta. *Progress*-luokalla on joukko metodeita, joilla päivitetään ja hallitaan käyttöliittymässä olevaa komponenttia. Metodia *public void updateprogress()* kutsutaan tiedonsiirron yhteydessä aina, kun yksi tavupuskurillinen dataa on siirretty. Metodi *public void setprogress(long yhteispit)* alustaa komponentin minimi- ja maksimiarvoilla. Maksimiarvo tulee kutsun yhteydessä parametrina ja se kertoo komponentille siirrettävän datan yhteismäärän. Metodi *public void setBufferSize(int Buf_Size)* kertoo *progress*-luokalle sen, miten paljon dataa siirretään yhden puskurillisen aikana. Tämän tiedon perusteella *JProgressBar*-komponentti osaa päätellä prosentuaalisen osuuden statusta päivitettäessä. Luokalla on myös metodit *public int getBufferSize()* sekä *public void clearProgressBar()*, joista ensimmäinen palauttaa integer-tyyppisen paluuarvon joka kertoo senhetkisen valitun puskurin koon. Jälkimmäistä metodia kutsutaan, kun tiedonsiirron jälkeen halutaan nollata *JProgressBar*-komponentin arvot.

6.3.3 Filelkm- ja filelengths-luokat

Seuraavaksi koodissa on kaksi julkista luokkaa *public class filelkm* ja *public class filelengths*. Näistä ensimmäisen luokan tehtävä on laskea siirrettävien tiedostojen yhteislukumäärä. Laskenta toteutetaan kutsumalla metodia *public int getfilelkm()*. Toisen luokan tehtävä on laskea siirrettävien tiedostojen yhteiskoko. Tämä lasketaan metodissa *public long getfilelengths(int lkm)*. Se palauttaa long-tyyppisen muuttujan, joka sisältää kaikkien tiedostojen yhteenlasketun koon tavuina.

6.3.4 Browsefiles-luokka

Browsefiles-luokan tehtävä on ylläpitää graafisessa käyttöliittymässä olevaa listaa, joka näyttää käyttäjän valitsemat tiedostot erillisellä javax.swing-kirjaston *JTextField*-komponentilla. Luokalla on kolme eri metodia, jotka ovat *public void add()*, *public void clear()* ja *public void remove()*. Näistä ensimmäinen metodi käyttää javax.swing-kirjaston komponenttia *JFileChooser*, jolla valitaan tietokoneen kovalevyltä haluttu tiedosto. Tiedoston instanssi tallennetaan sitä varten luotuun taulukkoon ja samoin tallennetaan valitun tiedoston nimi omaan taulukkoonsa. Lopuksi nimi tulostetaan *JTextField*-komponentille. Metodi *public void clear()* tyhjentää kaikki instanssit tiedostotaulukosta sekä nimet nimitaulukosta. Metodi *public void remove()* poistaa ainoastaan viimeisimmän tiedoston taulukoista.

6.3.5 LanIP-luokka

LanIP-luokan tehtävä on kysyä käyttäjältä vastaanottajan IP-osoite silloin, kun dataa siirretään IP-verkkoa pitkin. Osoitteen kysyminen tehdään metodissa *getLanIP()*, joka palauttaa osoitteen String-tyyppisenä. Luokka tarkastaa automaattisesti osoitteen oikean muodon eli sen, että osoite on numeerisessa muodossa ja oikean pituinen.

6.3.6 Kuuntelijaluokat

Kuuntelijaluokkia koodiin on rakennettu kaksi kappaletta. Ne ovat *private class ButtonListener* sekä *private class ConnectionListener*. *ButtonListener*-luokka toteuttaa *ActionListener*-rajapinnan ja sen tehtävänä on kuunnella graafisessa käyttöliittymässä olevia nappuloita. Aiemmin rakentajassa määritellyille nappuloille on lisätty kuuntelijaolio, joka generoi tapahtuman aina, kun kyseistä nappulaa painetaan. *ActionListener*-tyyppinen kuuntelijaolio lisätään käyttämäl-

lä metodia *addActionListener()*. Nappulaa painettaessa kutsutaan *Buttonlistener*-luokan metodia *actionPerformed(),* joka toteuttaa kaiken logiikan, jota liittyy jonkin nappulan painamiseen.

Toinen kuuntelija-luokka on *connectionlistener*-luokka, jonka toiminta on samanlainen kuin aiemmin esitelty. Ainoa ero on se, että *connectionlistener*-luokka toteuttaa *ActionListener*-rajapinnan sijasta *ItemListner*-rajapinnan. *Connectionlistener*-luokan tehtävä on kuunnella javax.swing-kirjaston komponenttia *JComboBox*, jota painamalla avautuu alasvetovalikko, josta käyttäjä voi valita halutun siirtotien. Valinta aiheuttaa tapahtuman, joka kutsuu *connectionlistener*-luokan metodia *itemStateChanged()*, joka puolestaan toteuttaa valintaan liittyvän logiikan.

6.3.7 Bluetooth-apuluokat

Seuraavaksi koodiin on rakennettu kaksi luokkaa, joiden tarkoitus on alustaa tiedonsiirtoa Bluetooth-yhteyttä pitkin. Ensimmäinen luokka on *class DeviceDiscoverer*, joka implementoi *javax.bluetooth.DiscoveryListenerin*. Tämän luokan tarkoitus on etsiä Bluetooth-kantaman sisällä olevia muita Bluetooth-laitteita. Toinen luokka on *public class ServiceDiscoverer*, joka myös implementoi *javax.bluetooth.DiscoveryListenerin*. Lisäksi luokka perii *Thread*-luokan, jokainen luokasta luotu olio toimii siis säikeenä. Tämän luokan tehtävä on etsiä palveluita löydetyistä Bluetooth-laitteista. Koska näiden luokkien toteutus perustuu javax.bluetooth-laajennuspakettiin, ja siten sisältävät olennaisia osia Bluetooth-tiedonsiirtoa varten, käydään niiden rakenne ja toiminta läpi hieman yksityiskohtaisemmin.

6.3.7.1 DeviceDiscoverer-luokka

DeviceDiscoverer-luokka on apuluokka, jonka tehtävänä on etsiä kantaman sisällä olevia muita Bluetooth-laitteita. Etsintä alkaa, kun yhteydeksi on valittu Bluetooth ja graafisesta käyttöliittymästä painettu nappia *Send*. Valinta käyn-

nistää *Laheta*-säikeen, jossa Bluetooth-yhteyden osalta luodaan *DeviceDiscoverer*-luokasta olio *dev_discov*. Tämä on esitetty listauksessa 2.

```
DeviceDiscoverer dev_discov = new DeviceDiscoverer();
```

Listaus 2 *Dev_discov*-olion luonti ja Bluetooth-laitteiden etsinnän aloitus.

Olion luominen käynnistää sarjan tapahtumia, jotka on esitetty listauksissa 3 ja 4.

```
class DeviceDiscoverer implements DiscoveryListener {

        Vector remoteDevices = new Vector();
        DiscoveryAgent discoveryAgent;
  …
```

Listaus 3 *DeviceDiscoverer*-luokan attribuutit. [2.]

DeviceDiscoverer-luokan alussa luodaan vektori *remoteDevices*, johon talletetaan kaikki löydetyt Bluetooth-laitteet. Seuraavaksi esitellään *DiscoveryAgent*-tyyppinen muuttuja *discoveryAgent*, jonka avulla laitteiden etsintä myöhemmin käynnistetään. Seuraavassa on listaus 4 *DeviceDiscoverer*-luokan rakentajasta.

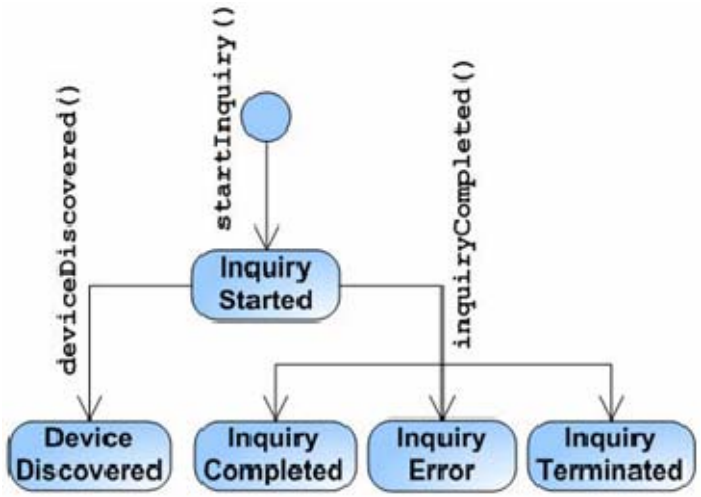
```
public DeviceDiscoverer() {

  try {
  LocalDevice localDevice = LocalDevice.getLocalDevice();
  discoveryAgent = localDevice.getDiscoveryAgent();
  infoarea.append("Searching Filetransfer from bluetooth devices...\n");

  discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
  } catch(Exception e) {
    e.printStackTrace();
  }
}
```

Listaus 4 *DeviceDiscoverer*-luokan rakentaja. [2.]

Rakentajassa luodaan *localDevice*-olio luokasta *javax.bluetooth.LocalDevice* ja siihen alustetaan tiedot oman koneen Bluetooth-laitteesta. Tämä mahdollistaa sen, että aiemmin luotu *discoveryAgent*-muuttuja voidaan nyt alustaa *DiscoveryAgent*-objektilla, joka voidaan hakea *localDevice*-olion kautta kutsulla *getDiscoveryAgent()*. Kun alustukset on tehty, voidaan *discoveryAgent.startInquiry()*-metodia kutsumalla käynnistää kantaman sisällä olevien Bluetooth-laitteiden etsintä. Tämä kutsu asettaa omassa koneessa olevan Bluetooth-laitteen etsintä moodiin. Kuvassa 9 on tilakaavio etsintäprosessista.



Kuva 9 Tilakaavio Bluetooth-laitteiden etsinnästä. [7.]

Jos etsinnän tuloksena löytyy muita laitteita, generoi Java Virtuaalikone *DiscoveryListener*-rajapinnan kautta kutsun *deviceDiscoverer*-luokan metodille *public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass cod)*. Parametrina metodi saa instanssin löydettyyn laitteeseen sekä tiedot laitteen tyypistä sekä tarjolla olevista palveluista. Tämä metodi on esitetty listauksessa 5.

```
public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass
cod) {
  try{
    remoteDevices.addElement(remoteDevice);
  }catch(Exception e){
    e.printStackTrace();
  }
}
```

Listaus 5 *DeviceDiscoverer*-luokan metodi *deviceDiscovered()*. [2.]

Kun kantaman sisällä olevat laitteet on etsitty, generoi Java virtuaalikone *DiscoveryListener*-rajapinnan kautta kutsun metodille *public void inquiryCompleted(int discType)*. Parametrina metodi saa tiedon siitä, mikä kutsun aiheutti. Tässä tapauksessa paluuarvona tulisi olla tieto siitä, että haku päättyi normaalisti, eikä minkään virhetilanteen johdosta. Listauksessa 6 on koodi metodista *public void inquiryCompleted(int discType)*.

```
public void inquiryCompleted(int discType) {

  if (discType == DiscoveryListener.INQUIRY_COMPLETED) {
    infoarea.append("Inquiry completed\n");
  }
  else if (discType == DiscoveryListener.INQUIRY_TERMINATED) {
    infoarea.append("Inquiry terminated\n");
  }
  else if (discType == DiscoveryListener.INQUIRY_ERROR) {
    infoarea.append("Inquiry error\n");
  }
  infoarea.append("Searching for services from Bluetooth-devices\n");
  ServiceDiscoverer  Serv_discov = new ServiceDiscoverer(remoteDevices);
  Serv_discov.start();
}
```

Listaus 6 *DeviceDiscoverer*-luokan metodi *inquiryCompleted*. [2.]

Metodin sisällä tulostetaan käyttäjälle ilmoitus haun tuloksesta. Metodin lopussa luodaan uusi olio luokasta *ServiceDiscoverer*. Olio on itse asiassa säie, jonka start()-metodia kutsumalla aikaansaadaan *ServiceDiscoverer* -apuluokan määrittelemä palvelu käynnistettyä. Parametrina oliolle annetaan lista löydetyistä Bluetooth-laitteista.

6.3.7.2 ServiceDiscoverer-luokka

ServiceDiscoverer on apuluokka, jonka tehtävänä on hakea palveluita löydetyistä Bluetooth-laitteista. Luokan toimintaperiaate on hyvin samanlainen edellä mainittuun luokkaan nähden. Se implementoi *DiscoveryListener*-rajapinnan Java-virtuaalikoneen generoimia kutsuja varten, mutta erona on se, että se perii

myös *Thread*-luokan, jolloin sen *run*-metodia ajetaan omana säikeenä. Luokan alussa on luotu yksilöllinen tunniste, joka tarvitaan yksilöimään palvelu muista koneella olevista Bluetooth-palveluista. Tämän tunnisteen perusteella myös Filetransfer-ohjelmat löytävät toisensa luotaessa Bluetooth-yhteyttä. *ServiceDiscoverer*-luokan rakentajassa otetaan vastaan parametrina vektori *deviceList*, joka luotiin aiemmin *deviceDiscoverer*-luokassa. Tämä vektori sisältää listan kaikista löydetyistä Bluetooth-laitteista. Seuraavassa on listaus 7 *ServiceDiscoverer*-luokan alustuksista.

```
public class ServiceDiscoverer extends Thread implements
DiscoveryListener {
  UUID[] uuidSet = {new UUID("2112", true)};
  int[] attrSet = {0x0100, 0x0003, 0x0004};
  ServiceRecord serviceRecord;
  String connectionURL;
  Vector deviceList;
  String FiletransferDevice;

  public ServiceDiscoverer(Vector deviceList) {
    this.deviceList = deviceList;
  }
```

Listaus 7. *ServiceDiscoverer*-luokan alustukset ja rakentaja. [2.]

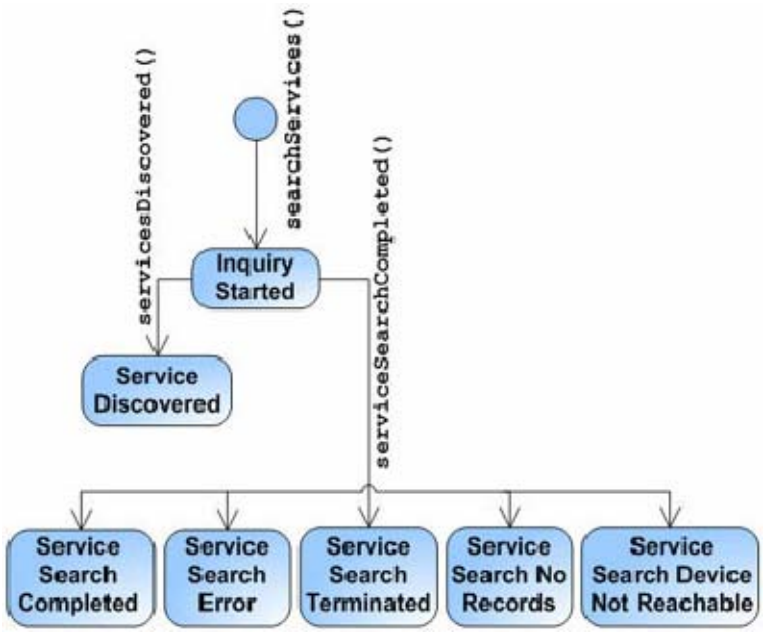
Listauksessa 8 on esitelty *ServiceDiscoverer* luokan *run*-metodin lähdekoodi.

```
public void run(){
  try {
    LocalDevice localDevice = LocalDevice.getLocalDevice();
    DiscoveryAgent discoveryAgent = localDevice.getDiscoveryAgent();
    RemoteDevice remoteDevice = null;

    for(int i=0; i < deviceList.size(); i++){
      remoteDevice = (RemoteDevice)deviceList.get(i);
      FiletransferDevice = remoteDevice.getFriendlyName(true);
      discoveryAgent.searchServices(attrSet, uuidSet,
      remoteDevice, this);
      try{ Thread.sleep(2000); } catch (Exception e){ }
    }
  }
  catch(Exception e) { e.printStackTrace();  }
}
```

Listaus 8. *ServiceDiscoverer*-luokan *run*-metodi. [2.]

Listauksesta 8 huomataan, että *run*-metodissa tehdään samat alustukset kuin *deviceDiscoverer*-luokassa, eli luodaan instanssit *localDevice*- sekä *discoveryAgent* -objekteille. Kun nämä on luotu, käydään parametrina saatu vektori *deviceList* läpi for-silmukassa ja etsitään löydetyistä laitteista palveluita. Palvelujen etsintä käynnistyy *DiscoveryAgent*-luokan metodilla *searchServices(attrSet, uuidSet, remoteDevice, this)*. Kuva 10 havainnollistaa palvelun etsintäprosessia.



Kuva 10 Tilakaavio Bluetooth-palveluiden etsinnästä. [7.]

SearchServices()-metodille annetaan parametrina mm. haetun palvelun yksilöllinen tunniste *UUID*, jonka perusteella löydetyistä laitteista löydetään oikea palvelu. *Run*-metodissa on myös koodi, joka pysäyttää säikeen suorituksen kahdeksi sekunniksi. Tämä aika riippuu laitteistosta. Säikeen pysäytystä ei välttämättä tarvita, mutta sen poisjättäminen saattaa aiheuttaa sen, että palveluita ei etsitä muista kuin ensimmäisistä parametrina saaduista laitteista. [2.]

Aina, kun *searchServices()*-metodi löytää haetuista laitteista *UUID*:tä vastaavan palvelun, generoi Java-virtuaalikone kutsun *DiscoveryListener*-rajapinnan kautta metodille *servicesDiscovered()*. Tämän metodin lähdekoodi on esitetty listauksessa 9.

```
public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {

  for(int i = 0; i < servRecord.length; i++) {

    DataElement serviceNameElement =
    servRecord[i].getAttributeValue(0x0100);
    String serviceName = (String)serviceNameElement.getValue();

    if(serviceName.equals("Filetransfer")){

      try {
        connectionURL = servRecord[i].getConnectionURL(1,false);
        BTconnectionURL = connectionURL;
        infoarea.append("Filetransfer found from " + FiletransferDevice + "\n");
      } catch (Exception e){
      }
    }
  }
}
```

Listaus 9 *ServiceDiscoverer*-luokan *servicesDiscovered()*-metodi. [2.]

Metodin sisällä otetaan Java-virtuaalikoneelta parametrina tulleiden laitteiden tiedot vastaan. Jos tiedoista löytyy haluttu palvelun tarjoaja, kyseisen laitteen URL-osoite talletetaan muuttujaan. Tämän muuttujan avulla voidaan myöhemmin avata yhteys vastaanottavaan osapuoleen dataa siirrettäessä.

6.3.8 Send- ja receive-luokat

Datan siirtoa varten on luotu kaksi luokkaa, jotka perivät *Thread*-luokan. Tämä tarkoittaa sitä, että luokista dynaamisesti luodut oliot toimivat säikeinä. Tämä myös mahdollistaa sen, että graafinen käyttöliittymä ei lukkiudu tiedonsiirron ajaksi. Datan lähettämistä varten on oma luokkansa nimeltä *Send* ja vastaanottoa varten omansa *Receive*.

6.3.8.1 Send-luokka

Send-luokan rakenne on melko yksinkertainen. Ennen *run*-metodin suoritusta luodaan kaksi oliota *flkm* ja *fl*. Kuten aiemmin selvitettiin, näillä voidaan alustaa tiedonsiirtoa selvittämällä siirrettävien tiedostojen lukumäärä ja niiden pituus. Tämän jälkeen suoritetaan *run*-metodi, jossa if-lauseella selvitetään käyttäjän valitsema tiedonsiirtomedia. Jokaista mediaa varten on oma lohkonsa ja niiden toiminta keskenään on hyvin samanlaista. Aluksi muodostetaan yhteys vastaanottavaan koneeseen ja luodaan rajapinnat tiedonsiirtoa varten. Tiedon kirjoittamiseen käytetään java.io-kirjaston luokkia *OutputStream* ja *DataOutputStream* sekä lukemiseen *InputStream* ja *DataInputStream*. Yhteyksiä lähettävän ja vastaanottavan koneen välillä muodostetaan ja katkaistaan useampaan kertaan jos siirrettäviä tiedostoja on useampia kuin yksi. Ensimmäisellä yhteyskerralla siirretään tarvittava informaatio siirrettävästä datasta, kuten siirrettävien tiedostojen lukumäärä, käytetyn tavupuskurin koko ja tiedostojen yhteispituus. Seuraavalla yhteyskerralla siirretään ensimmäinen valituista tiedostoista. Tiedostonsiirron jälkeen yhteys katkaistaan ja muodostetaan tarvittaessa uusi, jos siirrettäviä tiedostoja on enemmän. Useampien yhteyksien muodostuksella on pyritty saamaan aikaan helpompi ja luotettavampi tiedonsiirtomekanismi, jolloin yhteyden katkeaminen kesken datan siirron aiheuttaisi mahdollisimman vähän vahinkoa. Ongelmia saattaa tulla esimerkiksi valittaessa liian suuri tavupuskuri tai yhteyden katketessa jostain muusta syystä. Listauksessa 10 on esitetty lähdekoodi, jolla tiedosto lähetetään vastaanottajalle. Esimerkki on lähetyksestä Bluetooth-yhteyttä pitkin.

```
for(int i=0; i < lkm; i++)
{
 StreamConnection connection = (StreamConnection) Connector.open(url);
 DataOutputStream dos =  new
 DataOutputStream(connection.openDataOutputStream());
 FileInputStream fis = new FileInputStream(outfile[indeksi-1]);

 dos.writeUTF(tiedoston_nimi[indeksi-1]);
 dos.writeLong(outfile[indeksi-1].length());
 infoarea.append("Sending file: " + tiedoston_nimi[indeksi-1] + "...\n");

 int bytesRead;
 while((bytesRead = fis.read(buffer)) != -1){
  dos.write(buffer);
   pro.updateprogress();
 }
 fis.close();
 dos.close();
 infoarea.append( tiedoston_nimi[indeksi-1] + " sent.\n");
 infoarea.setCaretPosition( infoarea.getDocument().getLength() );
 bf.remove();
 connection.close();
}
```

Listaus 10 Tiedoston lähetys Bluetooth-yhteyttä pitkin.

Lähetyksen alussa avataan yhteys vastaanottajaan, jonka jälkeen luodaan rajapinnat tiedoston lähetykseen ja sen lukemiseen kovalevyltä. Itse tiedosto lähetetään while-silmukassa, jossa *FileInputStream*-rajapinnan kautta luetaan tiedostoa kovalevyltä tavupuskuriin. Kun puskuri on täynnä, se lähetetään *DataOutputStream*- ja *StreamConnection*-rajapintojen avulla vastaanottajalle. Seuraavaksi päivitetään tiedonsiirron status palkkia kutsumalla *progress*-luokan *updateprogress()*-metodia. Kun silmukka on käyty läpi, suljetaan rajapinnat ja päivitetään graafista käyttöliittymää kertomaan tiedoston lähetyksestä käyttäjälle. Sama idea toistuu muidenkin tiedonsiirtomedioiden kohdalla, joten niihin ei tarkemmin tässä kohtaa mennä.

6.3.8.2 Receive-luokka

Receive-luokan rakenne on vastaava *Send*-luokan rakenteen kanssa. Toiminnallisuudessa on kuitenkin pieniä eroja. *Receive*-luokan alussa ei ole vastaavia

alustuksia, vaan tiedot saadaan lähettäjältä ensimmäisen yhteyden aikana. Tiedon lukemiseen käytetään java.io-kirjaston luokkia *InputStream* ja *DataInputStream* sekä kirjoittamiseen *OutputStream* ja *DataOutputStream*. Seuraavalla yhteyskerralla vastaanotetaan itse tiedosto. Listauksessa 11 on esitetty lähdekoodi tiedoston vastaanottamisesta Bluetooth-yhteyttä pitkin.

```
StreamConnection conn = server.acceptAndOpen();
DataInputStream dis =  new DataInputStream(conn.openDataInputStream());
String infile = dis.readUTF();
FileOutputStream fos = new FileOutputStream(new File(infile) );

int length = (int)(dis.readLong());
int jako = (length/pro.getBufferSize());
int jaannos = (length-(jako*pro.getBufferSize()));
byte[] bu = new byte[jaannos];
int kierrokset=0;
infoarea.append("Receiving file: " + infile + "...\n");

if(jako != 0){
  while( kierrokset <= (jako-1) ){
    dis.read(buffer);
    fos.write(buffer);
    pro.updateprogress();
    if (kierrokset == (jako-1)){
      dis.read(bu);
      fos.write(bu);
      fos.close();
      pro.updateprogress();
      kierrokset++;
    }
    kierrokset++;
  }
  dis.close();
  conn.close();
}
else{
  dis.read(bu);
  fos.write(bu);
  fos.close();
  pro.updateprogress();
  dis.close();
  conn.close();
}
```

Listaus 11 Lähdekoodi tiedoston vastaanottamisesta.

Listauksesta huomataan, että tiedoston vastaanottomekanismi on hieman monimutkaisempi. Yhteyden muodostuksen ja *DataInputStream*-luokan instanssin luonnin jälkeen vastaanotetaan tieto siirrettävän tiedoston nimestä sekä siirrettävän tiedoston koosta. Sitten lasketaan vastaanotettavien kokonaisten tavupuskurien määrä ja suoritetaan vastanotto yhtä monessa while-silmukan kierroksessa. Jos vastaanotettavan tiedoston koko on pienempi kuin kokonainen tavupuskurillinen, vastaanotetaan data erillisessä else-silmukassa, koska tuntemattomasta syystä tiedoston lopetusmerkkiä ei saatu vastaanotettua datavirrasta.

7 YHTEYDEN MUODOSTAMINEN

Tähän osioon on koottu tarkemmat tiedot eri yhteystyyppien muodostamisesta. Tässä kerrotaan lyhyesti kooditasolla, miten kahden koneen välille muodostetaan LAN-, sarja-, rinnakkais- ja Bluetooth-yhteyksiä.

7.1 LAN-yhteys

LAN-yhteys muodostetaan kahden koneen välille käyttämällä java.net-kirjaston *Socket*- ja *ServerSocket*-luokkia. Nämä tarjoavat tarvittavat metodit perinteisen asiakas - palvelin-mallin rakentamiseen. *Serversocket*-rajapintaa käytetään yhteyden vastaanottavassa päässä ja *socket*-rajapintaa yhteyden muodostavassa päässä. Alla on listaus 12 LAN-yhteyden muodostamiseen tarvittavasta lähdekoodista vastaanottavassa päässä.

```
ServerSocket serverSocket1 = new ServerSocket( 8001 );
Socket connect1 = serverSocket1.accept();
```

Listaus 12 ServerSocketin käyttö LAN-yhteyden muodostamiseen.

Aluksi luodaan *ServerSocket*-luokasta olio *serverSocket1*, joka sidotaan tässä tapauksessa porttiin 8001. Seuraavaksi luodaan toinen olio *connect1* luokasta *Socket*, joka asetetaan kuuntelemaan määritettyä porttia aiemmin luodun olion metodilla *accept()*. Tämä keskeyttää säikeen suorituksen ja suoritus jatkuu vasta kun asiakaspuolelta tulee yhteydenotto palvelimen kuuntelemaan porttiin. Listauksessa 13 on lähdekoodi, jolla toteutetaan asiakaspuolen yhteydenotto.

```
private Socket socket;
…
socket = new Socket(osote,8001);
```

Listaus 13 Yhteyden muodostus palvelimelle LAN-yhteyttä pitkin

Asiakaspuolella yhteyden muodostamiseen ei tarvita muuta kuin se, että luodaan *Socket*-luokasta olio *socket* ja annetaan sille parametrina palvelimen IP-osoite sekä palvelimen kuuntelema porttinumero. Tällöin asiakas ottaa yhteyden palvelimeen ja yhteys on aktivoitu. Tämän jälkeen kommunikaatio asiakkaan ja palvelimen välillä hoidetaan aiemmin esiteltyjä *Input*- ja *OutputStream*-rajapintoja käyttämällä.

7.2 Sarjaportti-yhteys

Muodostettaessa yhteyttä sarjaportin kautta kahden koneen välille pitää tutkia hieman tarkemmin javax.comm-kirjastosta löytyviä luokkia ja metodeita. Tässä tapauksessa tarvitaan kahta luokkaa, jotka ovat *CommPortIdentifier*-luokka ja *CommPort*-luokka. Näistä ensimmäistä tarvitaan tunnistamaan paikallisella koneella olevat COM-portit ja jälkimmäistä kommunikointiin käytössä olevien COM-porttien kanssa. Alla on listaus lähdekoodista, jota käytetään yhteyden muodostamiseen molemmissa päissä. Nyt ei ole käytössä perinteistä asiakas-palvelin- mallia, vaan yhteys avataan vain oman koneen COM-porttiin ja kommunikaatio kahden koneen välillä suoritetaan käyttäen normaaleja *InputStream*- ja *OutputStream*-rajapintoja.

```
SerialPort port_;

CommPortIdentifier portId =
CommPortIdentifier.getPortIdentifier("COM1");

port_ =  (SerialPort)portId.open("Filetransfer", 30000);
```

Listaus 13 Yhteyden muodostus sarjaportin kautta. [5.]

Aluksi luodaan COM-porttien tunnistamista varten olio *portId*, johon haetaan tässä tapauksessa parametrina annetun COM1-portin instanssi metodilla *getPortIdentifier(”COM1”)*. Tämän jälkeen avataan portti *CommPortIdentifier*-luokan metodilla *open(”Filetransfer”,30000)* ja annetaan tulos *port_*-oliolle. Parametrina metodille annetaan porttia käyttävän applikaation nimi ja aikaraja portin avaamiselle. Muita parametreja ei yhteyden muodostukselle nyt anneta. Tämä johtuu Javax.comm-kirjaston tarjoamasta automatiikasta, jossa yhteyden muodostuksessa käytetään automaattisesti ns. 8N1-tyyppistä protokollaa. Haluttaessa yhteydelle voitaisiin antaa parametrit toisentyyppisestä yhteysprotokollasta, mutta tässä työssä päädyttiin käyttämään oletusarvoja sen yleisyyden takia.

7.3 Rinnakkaisportti-yhteys

Luotaessa yhteyttä kahden koneen välille rinnakkaisportin kautta tarvitaan javax.comm-kirjaston luokkia *CommPortIdentifier* ja *ParallelPort*. Näistä ensimmäistä tarvitaan tunnistamaan paikallisella koneella olevat LPT-portit ja jälkimmäistä kommunikointiin käytössä olevien rinnakkaisporttien kanssa. Yhteyden muodostaminen ja kommunikointi on periaatteessa samanlaista kuin sarjaporttia pitkin, mutta portin kanssa kommunikointiin käytettävä olio luodaan *ParallelPort*-luokasta. Koska lähdekoodi muuten on samanlainen, sitä ei tässä tarkemmin esitetä.

7.4 Bluetooth-yhteys

Bluetooth-yhteyden muodostaminen on käytettävistä yhteystyypeistä selvästi monimutkaisin. Yhteyden alustukseen kuului paljon tehtäviä, kuten Bluetooth-laitteiden etsintä sekä palveluiden etsintä löydetyistä laitteista. Näihin ei kuitenkaan enää tässä luvussa mennä, koska nämä käytiin läpi aiemmin luvussa 6.3.7. Bluetooth-yhteyden muodostus tapahtuu taas perinteisen asiakas - palvelin-mallin mukaisesti, eli palvelin asetetaan odottamaan asiakkaan yhteydenottoa. Ennen kuin asiakas voi yrittää muodostaa yhteyttä, on palvelimen tehtävänä ensin luoda ns. *ServiceRecord* ja lisätä se palvelimen *Service Discovery DataBaseen*. *ServiceRecord* sisältää kuvauksen palvelimen palvelusta ja *Service Discovery DataBase* on tietokanta, jossa palvelu asetetaan näkyväksi asiakkaille. [7.] Listauksessa 14 lähdekoodi, jossa palvelin asetetaan odottamaan asiakkaan yhteydenottoa.

```
final UUID uuid = new UUID("2112", true);
String name = "Filetransfer";
try {
  …
  StreamConnectionNotifier server = (StreamConnectionNotifier
  )Connector.open("btspp://localhost:"+uuid+";
  name="+name+";authorize=false;authenticate=false;encrypt=false");

  LocalDevice local = LocalDevice.getLocalDevice();
  ServiceRecord sr = local.getRecord(server);


  …
  StreamConnection conn_ = server.acceptAndOpen();
```

Listaus 14 Palvelimen asettaminen odottamaan Bluetooth-asiakasta. [1]

Aluksi muodostetaan yksilöllinen tunniste *UUID*, joka saa arvokseen kokonaisluvun 2112, ja lisäksi luodaan palvelulle nimi, joka tässä tapauksessa on Filetransfer. Tämän jälkeen pitää luoda *ServiceRecord*, jota varten on ensin luotava olio *server*, joka edustaa palvelua. Tämä alustetaan kutsumalla

javax.microedition.io kirjaston *Connector*-luokan metodia *open()*, jolle annetaan parametrina mm. osoite, tunniste *uuid*, palvelun nimi ja tiedot mahdollisesta salauksesta. Nämä parametrit siis yksilöivät palvelun ja tulevat näkymään julkisesti yhteyttä muodostaville asiakkaille. Osoite-parametrista mainittakoon, että osoitteen *localhost* edessä on lisämääre *btspp*, joka kertoo asiakkaille yhteyden käyttävän Bluetooth Serial Port Profilea, joka tarjoaa rajapinnan RS-232-yhteyttä emuloivaan RFCOMM-protokollaan. Seuraavaksi voidaan luoda itse *ServiceRecord* ja alustaa se kutsumalla *LocalDevice*-luokan metodia *getRecord()*. Metodille annetaan parametrina aiemmin luotu palvelua edustava olio *server*. Nyt tarvitsee enää aktivoida palvelin odottamaan asiakkailta tulevia kutsuja ja se tapahtuu kutsumalla metodia *acceptAndOpen()*, jolloin palvelu asetetaan näkyväksi *Service Discovery DataBaseen*. Tämä kutsu lukitsee säikeen suorituksen, kunnes asiakas ottaa yhteyden palvelimeen.

Kun palvelin on aktivoitu odottamaan asiakkaan yhteyksiä, voidaan asiakkaan puolelta muodostaa yhteys palvelimeen. Lähdekoodi tästä on esitetty listauksessa 15.

```
StreamConnection connection_ = (StreamConnection) Connector.open(url);
```

Listaus 15 Asiakaspuolen yhteyden muodostus Bluetooth-yhteyttä pitkin. [2.]

Ainoa asia, mikä tässä vaiheessa asiakkaan tehtäväksi jää, on muodostaa yhteys palvelimeen kutsumalla javax.microedition.io-kirjaston *Connector*-luokan metodia *open()*, jolle annetaan parametrina palvelimen osoite. Tämä osoite saatiin selville alustuksia suoritettaessa, kun etsittiin palveluita löydetyistä Bluetooth-laitteista. Käytännössä tämä tarkoittaa sitä, että palvelimen on ensin asetettava palvelu julkiseksi ja aktivoitava yhteyksien vastaanotto, ennen kuin asiakas voi aloittaa omat alustukset ja yhteyden muodostuksen.

8 OHJELMAN TESTAUS

Tähän lukuun on koottu käytännön testitulokset ja testauksessa käytetty laitteistokokoonpano. Tässä tarkastellaan myös testituloksiin vaikuttavia tekijöitä ja rajoituksia.

8.1 Testilaitteisto

Testauksessa on käytetty kolmea eri laitteistokokoonpanoa. Pääpiirteissään ensimmäinen kone on Acer-merkkinen kannettava tietokone, jossa on Intel Pentium 1.6 GHz:n M 725 -prosessori, 400 MHz:n väylänopeus, 512Mt:n DDR-tyyppinen keskusmuisti, 2 Mt:n suuruinen välimuisti sekä verkkokortti. Toinen kone on 1.4 GHz:n Intel Pentium -prosessorilla varustettu pöytämikro, jossa on 133 MHz:n väylänopeus, 512 Mt:n DDR-tyyppinen keskusmuisti sekä verkkokortti. Kolmantena laitteena on Fujitsu Siemens -merkkinen pöytämikro, jossa on 1,6 GHz:n Intel Pentium -prosessori, 256 Mt:n DDR-tyyppinen keskusmuisti, 400 MHz:n väylänopeus sekä verkkokortti. Kaikissa kolmessa tietokoneessa on Windows XP Professional -käyttöjärjestelmä. Lisäksi käytössä oli kaksi kappaletta USB-porttiin sovitettavaa Belkin Bluetooth USB-adapteria sekä nollamodeemikaapeli RS-232-liityntää varten.

8.2 Testitulokset

Filetransfer-ohjelman testaus RS-232-yhteyden osalta suoritettiin kahden aiemmin kuvatun pöytämikron avulla. Testattaessa yhteyttä lähetettiin kaksi 105 kb:n kuvatiedostoa. Tiedonsiirtopalkkia seuraamalla käsin mitatuksi siirtoajaksi saatiin 220 sekuntia. Kun muistetaan, että RS-232-yhteydestä oli käytössä ns. 8N1-kehysmalli, niin tämä vastaa hyvin koodissa asetettua n. 9600 b/s nopeutta. Java tarjoaa kyllä mahdollisuuden nopeuttaa tiedonsiirtoa asettamalla yhteydelle toiset parametrit. Se, miten nopeaa tiedonsiirtoa voidaan käyttää, riippuu täysin käytettävästä laitteistosta ja kaapelin pituudesta. Tässä työssä tyydyttiin käyttämään Javan asettamia oletusarvoja sen yleisyyden takia. Testissä käytetty tavupuskurin koko oli neljä tavua. Ohjelma testattiin myös paljon suuremmalla tavupuskurin koolla, yli 64 tavua, mutta tästä seurauksena oli jumiutunut yhteys.

Koska kaikissa Windows 2000- ja uudemmissa käyttöjärjestelmissä on tietoturvan takia rajoitettu rinnakkaisportin kautta tapahtuvaa liikennettä, ei testituloksissa ole mainintaa IEEE-1284-yhteyttä pitkin tapahtuvasta tiedonsiirrosta. Tietoturvaa on kohennettu kyseisissä käyttöjärjestelmissä siltä osin, että rinnakkaisportista ei suoraan pysty lukemaan dataa. Tähän on ratkaisuna tarjolla monenlaisia erillisiä ajureita, joiden avulla voidaan saada lukuoperaatioita suoritettua rinnakkaisportista, mutta näitä ei tämän työn osalta ole kokeiltu. [19.]

Bluetooth-yhteys testattiin kannettavan tietokoneen ja 1.4 GHz:n Intel Pentium -prosessorilla varustetun pöytämikron välillä. Testitiedostona käytettiin 3 205 448 tavun kokoista mp3-musiikkitiedostoa. Yhteyden alustusten kestoajaksi mitattiin n. 17 sekuntia ja tiedonsiirtonopeudeksi 3 205 448 / 85s = n. 38 kilotavua sekunnissa. Tämä nopeus saavutettiin tavupuskurin koolla 512. Käytettäessä pienempää tavupuskuria, kuten 128 tai 64, tiedonsiirto alkaa hidastua selvästi. Tämä johtuu siitä, että ohjelma joutuu suorittamaan useampia erillisiä lukuoperaatioita kovalevyltä, jolloin tästä alkaa muodostua pullonkaula. Huomattavasti suurempien tavupuskurien koon käyttämisellä ei saavuteta enää parannusta tiedonsiirtonopeuteen, koska itse laite ei enää pysty suurempaan nopeuteen.

LAN-yhteyden testaaminen jätettiin ainoastaan sille tasolle, että sen toiminta varmistettiin juoksuttamalla dataa kannettavan tietokoneen oman protokollapinon läpi. Tämä tapahtui siten että ensin käynnistettiin kaksi komentoriviä ja sitten molemmilta komentoriveiltä käynnistettiin oma Filetransfer-ohjelma. Tämän jälkeen siirrettiin dataa näiden kahden ohjelman välillä. Tällä tavoin varmistuttiin ainoastaan siitä, että tiedonsiirto ylipäätään toimii loogisesti oikein. Todellinen tiedonsiirtonopeus tulee riippumaan täysin Internet-palvelun tarjoajasta ja siitä, mistä datansiirtonopeudesta palveluntarjoajan kanssa on etukäteen sovittu.

9 OHJELMAN KEHITYSEHDOTUKSIA

Tähän lukuun on koottu joitakin ideoita ja parannusehdotuksia, joita työtä tehdessä on tullut esille. Yksi ehkä mielenkiintoisimmista ideoista olisi hyödyntää Java RMI:tä. RMI on Javan sisään rakennettu mekanismi, jonka avulla voidaan suorittaa etäoliokutsuja, jotka on toteutettu fyysisesti eri koneelle. Kutsut suoritettaisiin IP-verkon yli, joten mekanismi vaatisi jatkuvan LAN-yhteyden. Ideaa voisi hyödyntää esimerkiksi siihen, että työssä käytettyjä lisäkirjastoja javax.comm ja javax.bluetooth ei tarvitsisi olla muualla kuin palvelinkoneella. Koska RMI-mekanismilla voi ajon aikana ladata dynaamisesti koodia, voitaisiin lisäkirjastojen vaatimat koodirivit suorittaa palvelinkoneella, jolloin asiakkaalla ei tarvitse olla asennettuna mainittuja lisäkirjastoja omalla koneellaan. Tämä lisäisi huomattavasti ohjelman siirrettävyyttä eri koneiden ja ajoympäristöjen välillä.

Kun dataa siirretään IP-verkossa, on perusteltua ottaa huomioon tietoturva. Filetransfer-ohjelmaan ei ole toteutettu salausta, mutta sen voisi pienellä vaivalla toteuttaa LAN-yhteyksiä varten. Tämä mahdollistaisi arkaluontoisenkin datan lähettämisen luottamuksellisesti ja täten lisäisi ohjelman käytettävyyttä.

Jos ohjelmaa halutaan vielä monipuolistaa, voisi siihen lisätä tiedonsiirron USB- ja Firewire-liityntöjä varten. Tähän työhön niitä ei ole laitettu, koska Javasta ei löydy vielä tällä hetkellä tukea niihin. USB:stä on saatavilla joitakin Beta-versioita Javan kirjastoihin, mutta niiden toimivuutta ei ole vielä täysin testattu. Tuki Firewirelle on myös tekeillä, mutta sen valmistuminen menee todennäköisesti vielä myöhempään kuin USB:n.

10 YHTEENVETO

Käytettävistä neljästä tiedonsiirtokanavasta voisi sanoa, että kaikille löytynee käyttökohteensa huolimatta siitä, että osa niistä on ollut jo pitkään käytössä ja ovat siten hieman vanhentuneita. Sarja- ja rinnakkaisliityntöjä varten

käyttökohteet ovat enemmänkin pienempiä laitteita, joiden kapasiteetti ei riitä muihin dataliityntöihin. Tällaisia voivat olla esimerkiksi pienissä sulautetuissa järjestelmissä käytettävät prosessorikortit, jotka ohjaavat joukkoa muita erillisiä laitteita. Kunhan vain näiden prosessorien käyttämä protokolla tunnetaan, on Javalla helppo tehdä ohjelmia, jotka osaavat kommunikoida näiden laitteiden kanssa käyttäen RS-232-yhteyttä. Tosin pitää ottaa huomioon, että yksinkertaisimmat laitteet eivät tunnista Javan käyttämiä tietotyyppejä, mutta yksinkertaisen binääridatan kirjoittaminen tai lukeminen onnistuu siitä huolimatta varsin yksinkertaisesti. Rinnakkaisliityntä on vieläkin laajasti käytössä tulostimien ohjaamista varten. Tosin nykyään USB-liitäntäiset tulostimet ovat korvaamassa perinteisiä rinnakkaisportti-liitäntäisiä tulostimia.

LAN- ja Bluetooth-yhteyksien käyttökohteet ovatkin sitten jo laajemmat. LAN-yhteys on tällä hetkellä laajasti käytössä ja mahdollistaa tiedon siirron reitittimien kautta periaatteessa miten kauas tahansa. Erityisesti web-sovelluksia ohjelmoitaessa Java on saavuttanut laajan suosion. Javalla on helppo ja nopea toteuttaa erilaisia verkkopiirteitä. Työssä havaittiin, että normaalin asiakas-palvelin-mallin mukaisen yhteyden luominen ja tiedonsiirto Javalla käyttäen *Socket*- ja *ServerSocket*-rajapintoja on todella yksinkertaista.

Bluetooth-yhteyksien käyttö rajoittuu lyhyen kantamansa takia pienempien laitteiden tai laitteistojen väliseen kommunikointiin. Tällä hetkellä sen suosituin käyttötarkoitus on kannettavien tietokoneiden ja muiden laitteiden keskinäinen langaton tietojen siirto muutaman metrin etäisyyksillä. Markkinoilla kuitenkin on jo ratkaisuja, joilla kantamaa saadaan kasvatettua jopa kymmenkertaiseksi eli n.100 metriin asti. On varsin todennäköistä, että vielä hintojen pudotessa Bluetooth-tekniikka tulee langattomuutensa vuoksi yleistymään suuresti ja siten käyttökohteita tulee jatkuvasti lisää. Etenkin teollisuudessa sen mukana tuomat edut ovat kiistattomat. Ensinnäkin on olemassa paikkoja, jonne signalointi tavallisella kaapelilla ei onnistuisi. Toiseksi säästetään kustannuksissa, kun kalliita kaapelointeja voidaan jättää kokonaan pois. Voidaankin sanoa, että Bluetooth tarjoaa vaivatonta ratkaisua tiedonsiirtoon. Tässä työssä havaittiin, että

Bluetooth-tekniikan käyttöönotto Java-ohjelmointikielellä käy melko helposti. Javax.bluetooth-paketin lisääminen Javan kirjastoihin on yksinkertaista ja itse Bluetooth-laitteen käyttö valmiiden luokkien kautta koodissa onnistuu pienen perehtymisen jälkeen ilman suurempia ongelmia. On kuitenkin odotettavaa, että jossain vaiheessa Bluetooth-luokkien lisääminen Javan kirjastoihin tulee standardiksi, jolloin ohjelmoijan ei tarvitse välittää muusta kuin valmiiden luokkien käytöstä. Näiden lisäksi päätettäväksi jää enää käytettävä protokolla. Työssä käytetty RFCOMM-protokolla ei suinkaan ole ainoa vaihtoehto, vaan on muitakin vaihtoehtoja. Yksi mahdollisuus on käyttää RFCOMM-protokollan päällä toimivaa OBEX-protokollaa, jolla voidaan siirtää objekteja kuten tiedostoja esimerkiksi joidenkin Nokian puhelinmallien kanssa.

LÄHDELUETTELO

[1] Hopkins, Bruce: File transfer with JSR-82 and OBEX, Sep 2005. http://www-128.ibm.com/developerworks/wireless/library/wi-boogie1/, 12.9.2006

[2] Hopkins, Bruce: Creating the Bluetooth Music Store, Sep 2005. http://www-128.ibm.com/developerworks/wireless/library/wi-boogie2/, 12.9.2006

[3] HW-server: RS-232 - overview of RS-232 standard, http://www.hw-server.com/rs232, 10.7.2006

[4] Java Communications API, http://java.sun.com/products/javacomm/reference/api/index.html, 12.9.2006

[5] JDC Tech Tips, Jan 2002. http://java.sun.com/developer/JDCTechTips/2002/tt0122.html, 12.9.2006

[6] Mustalahti, Pasi: RS232, Joulukuu 2004. http://users.utu.fi/ptmusta/rs232.shtml, 7.7.2006.

[7] Ortiz, Enrique: Using the Java APIs for Bluetooth Wireless Technology, Part 1 - API Overview, Joulukuu 2004. http://developers.sun.com/techtopics/mobility/apis/articles/bluetoothintro/index.html, 7.8.2006.

[8] Rintala, Matti: Päättötyö TCP/IP, Maaliskuu 2001. http://koti.mbnet.fi/mrin/paattotyo/, 12.09.2006.

[9] Vanhala-Nurmi, Vuokko: Koulutusaineistoa, tiedon siirtäminen Internetissä. http://myy.helia.fi/~vanvu/tietoliikenne/internet/tiedonsiirto.html, 10.8.2006.

[10] Wikipedia: Bluetooth, Elokuu 2006. http://fi.wikipedia.org/wiki/Bluetooth, 5.5.2006.

[11] Wikipedia: TCP/IP, Elokuu 2006. http://fi.wikipedia.org/wiki/TCP/IP, 5.5.2006.

[12] Wikipedia: IEEE-1284, Syyskuu 2006. http://en.wikipedia.org/wiki/IEEE_1284, 15.5.2006.

[13] Bluetooth Device Access Guide: Bluetooth Architecture, Toukokuu 2006. https://developer.apple.com/documentation/DeviceDrivers/Conceptual/Bluetooth/BT_Bluetooth_Basics/chapter_2_section_4.html, 12.9.2006.

[14] Qusay H, Mahmoud:Sun Developer Network: Wireless Application Programming with J2ME and Bluetooth, Helmikuu 2003. http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth/ 14.4.2006.

[15] Qusay H, Mahmoud:Sun Developer Network: Part II: The Java APIs for Bluetooth Wireless Technology, Helmikuu 2003. http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth2/ 14.4.2006.

[16] Laakkonen, Ilmari: Siirtyvä tietoliikenne, Maaliskuu 2004. http://www.it.lut.fi/kurssit/03-04/010651000/luennot/Bluetooth_siirtyva.ppt#24, 18.4.2006.

[17] Fred G, Martin: Installing & Configuring Java for HC11 Code Download, Elokuu 2004. http://www.cs.uml.edu/~fredm/courses/91.305-fall04/javasetup.shtml, 9.12.2005

[18] Java Bluetooth.com: Bluecove development kit. http://sourceforge.net/projects/bluecove, 1.2.2006

[19] Peacock, Graig: PortTalk - A Windows NT I/O Port Device Driver, Toukokuu 2005. http://www.beyondlogic.org/porttalk/porttalk.htm 5.8.2006

LIITELUETTELO

Liite 2. CD-ROM