UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sara Perčič

RAZVOJ BLUETOOTH APLIKACIJE V OKOLJU JAVA ME IN JAVA SE

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01583/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **SARA PERČIČ**

Naslov: **RAZVOJ BLUETOOTH APLIKACIJE V OKOLJU JAVA ME IN JAVA SE**

DEVELOPMENT OF BLUETOOTH APPLICATION IN JAVA ME AND JAVA SE ENVIRONMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Diplomska naloga opisuje razvoj Bluetooth klepetalnice za okolje Java ME in Java SE ter uporabo Java API funkcij za upravljanje z bluetooth.

Opišite tehnologije o bluetooth razvoju za okolje Java ME ter o API funkcijah za upravljanje z bluetooth. Dokumentirajte razvoj Bluetooth klepetalnice za komuniciranje med uporabniki preko mobilnih telefonov in osebnih računalnikov. Opišite razloge za izbiro določenih tehnologij, njihovo implementacijo, in probleme, do katerih ste prišli pri razvoju klepetalnice. Podajte možne razširitve in izboljšave.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sara Perčič

RAZVOJ BLUETOOTH APLIKACIJE V OKOLJU JAVA ME IN JAVA SE

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Saša Divjak

Ljubljana, 2010

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Sara Perčič,

z vpisno številko 63030322,

sem avtor/-ica diplomskega dela z naslovom:

Razvoj Bluetooth aplikacije v okolju Java ME in Java SE

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  prof. dr. Saša Divjak
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 01.03.2010 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se prof. dr. Saši Divjaku za mentorstvo in pomoč pri izdelavi diplomskega dela ter družini in vsem ostalim, ki so me podpirali in pomagali v času študija.

Kazalo

Seznam uporabljenih kratic in simbolov

ACL ang. Asynchronous Connectionless Link
Bluetooth povezava, ki je paketno orientirana.

API ang. Application Programming Interface
Niz funkcij, procedur, metod, razredov ali protokolov, ki ga nudi operacijski sistem, knjižnica ali storitev za podporo računalniškemu programu.

BCC ang. Bluetooth Control Center
Opravlja tri specifične naloge: odpravljanje konfliktov med aplikacijami, omogočanje sprememb lastnosti Bluetooth naprave ter upravljanje z varnostnimi operacijami, ki lahko potrebujejo uporabnikovo interakcijo.

BNEP ang. Bluetooth Network Encapsulation Protocol
BNEP je protokol za inkapsulacijo paketov iz različnih Bluetooth protokolov.

CDC ang. Connected Device Configuration
Java ME konfiguracija za močnejše vgrajene naprave.

CLDC ang. Connected Limited Device Configuration
Java ME konfiguracija za majhne vgrajene naprave.

FP ang. Foundation Profile
Fondacijski profil je profil najnižjega nivoja za CDC. FP je mišljen za vgrajene naprave brez uporabniškega vmesnika in z mrežnimi zmožnostmi.

GAP ang. Generic Access Profile
: GAP je eden izmed Bluetooth profilov. Je osnova za vse ostale profile. Definira generične procedure povezane z vzpostavitvijo komunikacije med dvema napravama, vključno z odkrivanjem Bluetooth naprav, upravljanjem povezav in konfiguracij, ter procedur, povezanih z uporabo različnih nivojev varnosti.

GCF ang. Generic Connection Framework
: J2ME API funkcije za povezovanje. Zagotavlja funkcije skupne za vsa bazna povezovanja(paketi in tokovi).

GOEP ang. Generic Object Exchange Profile
: GOEP je abstraktni Bluetooth profil, na katerem so lahko grajeni konkretni profili, kot je OBEX.

GIAC ang. General Inquiry Access Code
: GIAC je eden izmed načinov vidnosti Bluetooth naprave. Če je izbran način GIAC je naprava ostalim vidna.

HCI ang. Host Controler Interface
: Zagotavlja vmesnik za dostop do osnovnopasovnega nivoja.

HTTP ang. HyperText Transfer Protokol
: Protokol, ki se uporablja za prenos podatkov od spletnih strežnikov do brskalnika na strani uporabnika.

IP ang. Internet Protocol
: Numerična vrednost, ki je dodeljena vsem napravam, ki vstopajo v mrežo z internetnim protokolom.

IrDA ang. Infrared Data Association
: Standard za komunikacijo med napravami na kratko razdaljo z uporabo infrardečih signalov.

IrOBEX ang. IrDA Object Exchange Protocol
: Protokol za prenos podatkov z deljenjem na pakete. Pri Bluetooth govorimo o OBEX in spustimo Ir.

JABWT ang. Java APIs for Bluetooth Wireless Technology
: Java API funkcije za Bluetooth brezžično tehnologijo.

JAM ang. Java aplication manager
: Java program za upravljanje z aplikacijami.

Java ME ang. Java Micro Edition
Java okolje, načrtovano za mobilne telefone in vgrajene sisteme.

Java SE ang. Java Standard Edition
Standardno Java okolje.

JSR-197 ang. Java Specification Request - 197
Dokument specifikacij za razvoj programov, ki zajema funkcije za uporabo Bluetooth, namenjen za okolje Java SE.

JSR-82 ang. Java Specification Request - 82
Dokument specifikacij za razvoj programov, ki zajema funkcije za uporabo Bluetooth, namenjen za okolje Java ME.

LAN ang. Local Area Network
Računalniško omrežje, ki krije manjša fizična omrežja(npr. doma ali v pisarni).

LIAC ang. Limited Inquiry Access Code
LIAC je eden izmed načinov vidnosti Bluetooth naprave. Če je izbran način LIAC, je naprava ostalim vidna le za omejen čas.

LMP ang. Link Manager Protocol
LMP je protokol za upravljanje povezav in je odgovoren za postavitev ter konfiguracijo povezav med Bluetooth napravami. Skrbi tudi za varnost, kot so avtentifikacija in enkripcija. Generira, zamenjuje in preverja povezave in ključe enkripcije.

L2CAP ang. Logical Link Control And Adaption Protocol
L2CAP je protokol, ki ga uporablja Bluetooth. Posreduje pakete med višjenivojskimi profili in HCI.

MIDlet je MIDP aplikacija.

MIDP ang. Mobile Information Device Profile
Specifikacija za uporabo Jave na vgrajenih napravah, kot so prenosni telefoni ali pa dlančniki.

MTU ang. Maximum Transmission Unit
Spremenljivka za največjo dovoljeno velikost prenosne enote(paketa).

OBEX ang. Object Exchange Protocol
Protokol za pošiljanje datotek, poslovnih vizitk in drugih objektov, ki se uporablja tudi za infrardeče povezave.

PP ang. Personal Profile
Osebni profil je namenjen napravam, ki potrebujejo uporabniški vmesnik. PP nadomešča PersonalJava tehnologijo in omogoča PersonalJava aplikacijam migracijo na Java ME ogrodje.

PSM PSM vrednost pove L2CAP sloju, na katero strežniško aplikacijo se želi odjemalec povezati. PSM so vrata za Bluetooth.

RAM ang. Random Access Memory
Bralno-pisalni pomnilnik.

RFCOMM ang. Radio Frequency Communication
Bluetooth protokol za prenos.

ROM ang. Read-only memory
Pomnilnik, iz katerega lahko samo beremo. Podatkov se ne da spreminjati.

SDAP ang. Service Discovery Profile
Profil definira temeljne operacije, potrebne za odkrivanje novih naprav, protokole in procedure, ki jih uporabljajo aplikacije za iskanje storitev na drugih Bluetooth napravah.

SDDB ang. Service Discovery Database
Baza, v katero Bluetooth shranjuje storitvene zapise.

SDP ang. Service Discovery Protocol
Protokol za odkrivanje novih naprav in storitev na le-teh.

SIG ang. Special Interest Group
Bluetooth SIG je pravna oseba, ki nadzira razvoj standardov Bluetooth in zainteresiranim izdelovalcem izdaja licence za tovrstne tehnologije in blagovne znamke.

SPP ang. Serial Port Profile
Je profil, ki zagotavlja procedure za zaporedne žične povezave med Bluetooth napravami, ki uporabljajo RFCOMM.

STB ang. Set Top Box
Set-top box (STB vmesnik) je naprava, ki jo uporabnik postavi pred TV, da digitalni signal spremeni v ustrezno obliko. Ta naprava je pretvornik iz digitalnega v analogni TV signal.

TCP ang. Transmision Control Protocol
Protokol za nadzor prenosa.

TCS Definira kontrolne signale klica za vzpostavitev zvočnega ali podatkovnega klica med Bluetooth napravami. Grajen je na L2CAP.

UART ang. Universal Asynchronous Receiver/Transmitter
Strojna oprema, ki pretvarja med paralelnimi in vzporednimi formami.

URL ang. Uniform Resource Locator
Določa, kje se nahaja identiteta vira in kako dostopati do njega.

USB ang. Universal Serial Bus
Način za povezovanje računalnikov in perifernih naprav.

UUID ang. Universally Unique Identifier
Standardna identiteta, ki se uporablja v razvoju programske opreme.

Povzetek

Diplomska naloga opisuje razvoj Bluetooth klepetalnice za okolje Java ME in Java SE ter uporabo Java API funkcij za upravljanje z Bluetooth.

V prvem delu so opisane tehnologije za Bluetooth, razvoj za okolje Java ME ter API funkcije za upravljanje z Bluetooth. Med opisanimi tehnologijami smo izbrali najprimernejše za našo aplikacijo in jih kasneje uporabili pri razvoju.

Drugi del diplomskega dela opisuje razvoj Bluetooth klepetalnice. Bluetooth klepetalnica je aplikacija za komuniciranje med uporabniki preko mobilnih telefonov in osebnih računalnikov. Uporabniki si lahko med seboj posredujejo sporočila in imajo pregled nad člani klepetalnice.

Opisani so razlogi za izbiro določenih tehnologij, njihova implementacija, podroben opis aplikacije Bluetooth klepetalnice, problemi, do katerih smo prišli, ter možne razširitve in izboljšave.

Ključne besede:

Bluetooth klepetalnica, Bluecove, JSR-82, Java ME, MIDlet

Abstract

The bachelor's thesis describes the development of a Bluetooth chat application for Java ME and Java SE platform. It describes the use of Java API functions for Bluetooth.

In the first chapter the technologies, development for Java ME platform and Bluetooth API functions are described. From all of the technologies we studied, we chose the most appropriate for our application and later we implemented them.

The second part of the bachelor's thesis is about Bluetooth chat development. Bluetooth chat is an application for chatting between mobile phones and personal computers. Users can transmit messages and have survey of all active members in the chat.

Reasons for selection of certain technologies, their implementation, detailed description of application, problems we encountered and possible upgrades and improvements are described.

Key words:

Bluetooth chat, Bluecove, JSR-82, Java ME, MIDlet

Poglavje 1

Uvod

V današnjih časih se tehnologija Bluetooth začenja pojavljati na vse več področjih. Bluetooth je namenjen tako pošiljanju elektronske pošte,
prostoročnemu telefoniranju in prenašanju datotek, kakor tudi igranju iger, brskanju po spletu in tiskanju.

Razcvet je doživel tudi razvoj aplikacij, namenjenih mobilnim telefonom, saj le težko najdemo osebo, ki ne bi imela svojega mobilnega telefona. Da bi oboje povezali v eno, smo prišli na idejo Bluetooth klepetalnice.

Aplikacija sama po sebi ni zelo uporabna, saj je Bluetooth kratkega dosega in bi se člani tako lahko osebno pogovorili namesto pošiljanja sporočil. Je pa dober primer seznanjenja s tehnologijami in API funkcijami za upravljanje z Bluetooth ter za pridobivanje potrebnega znanja pri kasnejšem razvoju bolj uporabnih aplikacij, vodenih z osebnimi računalniki.

Kot primer takšne aplikacije bi lahko bilo izvajanje določenih analiz(npr. štetje mimoidočih preko mobilnih telefonov), posredovanje informacij in oglasov mimoidočim, različni portali, ...

Poglavje 2

Tehnologija

V tem poglavju so opisane tehnologije, s katerimi se je potrebno seznaniti za razumevanje naslednjega poglavja. Slednje opisuje izvedbo praktičnega dela z uporabo opisanih tehnologij.

2.1 Splošno o Bluetooth

Bluetooth je brezžična komunikacija kratkega dosega od 10 do 100m. Nosilni signal deluje s frekvenco 2,4 GHz. Ta frekvenčni pas je globalno prost, zato Bluetooth deluje povsod po svetu. Narejen je robustno, tako da ga interference drugih signalov enake frekvence ne motijo.

Njegove prednosti so tudi v nizki ceni in predvsem varčnosti pri porabi električne energije, saj je na ta način primeren tudi za naprave brez možnosti neposrednega napajanja z električno energijo. Najvišja možna hitrost prenosa je z Bluetooth oddajnik 3.0+HS in sicer 24Mbit/s. Število povezav pa je omejeno na 8 aktivnih in 255 pasivnih povezav. Specifikacija je javna, kar omogoča lažji razvoj in uporabo ter hitro širjenje Bluetootha po svetu.

Bluetooth tehnologija zajema tri dele: oddajnik, Bluetooth sklad in Bluetooth profile.

2.1.1 Arhitektura sklada Bluetooth protokola

Plast protokola je razdeljena na dva dela: Bluetooth odjemalec in Bluetooth upravitelj(ang. controler). Loči ju standardizirani vmesnik odjemalec - upravitelj (Host Controler Interface - HCI).

Bluetooth gostitelj je višji nivo plasti in je ponavadi implementiran v programski opremi. Navadno je integriran v sistemski programski opremi ali op-

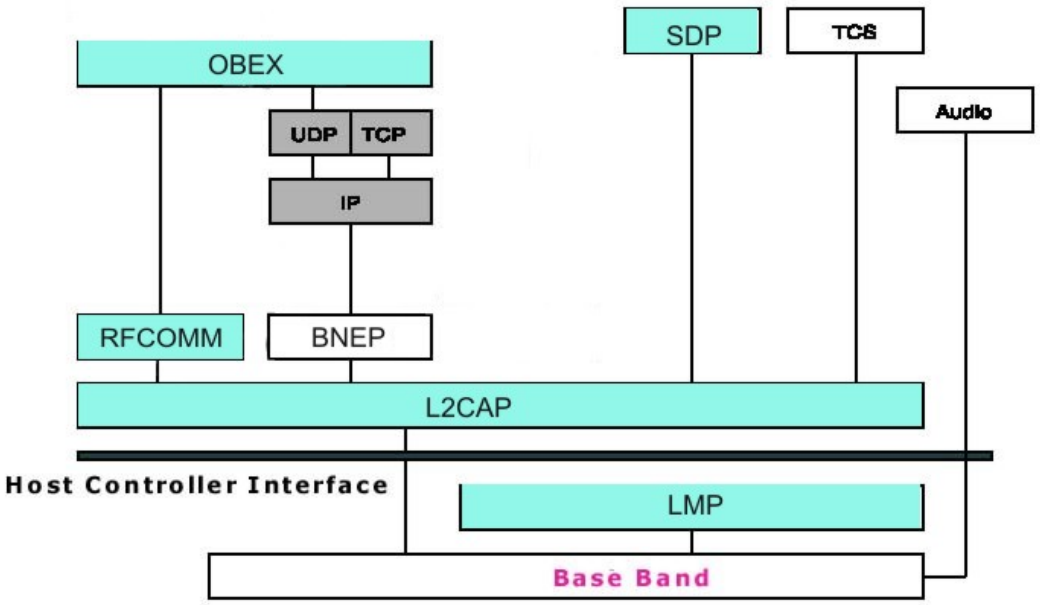
eracijskem sistemu. Ponavadi so Bluetooth gostitelji programski in tečejo na strojni opremi naprave gostitelja. Bluetooth profili so grajeni na vrhu protokolov. Naprava gostitelja je lahko npr. osebni računalnik, pri čemer je Bluetooth gostitelj integriran v operacijski sistem.

Bluetooth upravitelj pa je navadno modul strojne opreme, kot npr. PC kartica, ki se vstavi v napravo.

Vedno več naprav ima Bluetooth oddajnik že vgrajen. Če ga naprava nima, ga lahko vstavimo kot adapter preko USB ali UART vodil.

2.1.2 Bluetooth protokoli

Na sliki 2.1 je blokovni diagram plasti Bluetooth protokola. Z modro so obarvani protokoli iz Java API funkcij za Buetooth brezžično tehnologijo.



Slika 2.1: Arhitektrua Bluetooth sklada

Osnovni pas (ang. Baseband) omogoča fizično povezavo med Bluetooth enotami, ki ustvarjajo povezavo. Osnovni pas skrbi za procesiranje kanalov in časov, ter dostop do kanalov. Obstajata dve vrsti fizičnih povezav - sinhronske in asinhronske.

LMP je protokol za upravljanje povezav(ang. Link Manager protocol) in je odgovoren za postavitev ter konfiguracijo povezav med Bluetooth napravami. Skrbi tudi za varnost, kot so avtentifikacija in enkripcija. Generira, zamenjuje ter preverja povezave in ključe enkripcije.

HCI je standardni vmesnik za dostop do statusov strojne opreme, kontrolnih registrov in zmožnosti osnovnega pasu.

L2CAP(ang. Logical Link control and adaption protocol) leži med višje in nižjenivojskimi protokoli. Razvršča nižjenivojske povezave.

SDP(ang. Service Discovery Protocol-protokol za iskanje storitev) je grajen na L2CAP. Napravam omogoča iskanje storitev na sosednjih napravah. Najprej je potrebno iskanje vseh naprav in šele zatem lahko po napravah iščemo storitve.

RFCOMM omogoča emulacijo serijskih vrat preko L2CAP.

BNEP je protokol inkapsulacije paketov iz različnih protokolov. Potreben je za ustvarjanje omrežij preko Blutooth iz različnih protokolov iz nivoja 3.

TCS definira kontrolne signale klica za vzpostavitev zvočnega ali podatkovnega klica med Bluetooth napravami. Grajen je na L2CAP.

OBEX je protokol za pošiljanje datotek, poslovnih vizitk in drugih objektov, ki se uporablja tudi pri infrardečih povezavah.

Do sedaj je bilo razvitih že več implementacij sloja Bluetooth protokola. V Windows operacijskih sistemih se uporablja WIDCOMM, Microsoft Windows Stack, Blusoleil, Toshiba in drugi. Za Linux operacijske sisteme pa je bil razvit BlueZ.

2.1.3 Bluetooth profili

Bluetooth profil definira standarden način uporabe izbranih protokolov.
Opišemo ga lahko kot vertikalen prerez Bluetooth sklada. Dva profila lahko uporabljata različna nabora plasti in različne funkcionalnosti v isti plasti. Bluetooth naprava lahko podpira enega ali tudi več profilov. Štirje bazni profili so Generic Access Profile(GAP), Serial Port Profile(SPP),Service Discovery Profile(SDAP) in Generic Object Exchange Profile(GOEP).

GAP je osnova za vse ostale profile. Definira generične procedure, povezane z vzpostavitvijo komunikacije med dvema napravama, vključno z odkrivanjem Bluetooth naprav, upravljanjem povezav in konfiguracij, ter procedur, povezanih z uporabo različnih nivojev varnosti.

SPP definira potrebno za vzpostavitev emulacije zaporednih žičnih povezav z uporabo RFCOMM med dvema enakovrednima napravama.

SDAP definira temeljne operacije, potrebne za odkrivanje novih naprav. Ta profil definira protokole in procedure, ki jih uporabljajo aplikacije pri iskanju storitev na drugih Bluetooth napravah.

GOEP je abstraktni profil, na katerem so lahko grajeni konkretni profili. To so profili, ki uporabljajo OBEX. Profil definira vse potrebno za podporo OBEX.To je prenos datotek, sinhronizacija in potiskanje objektov.

Zgoraj našteti bazni profili so znani kot transportni profili, na katerih so lahko grajeni ostali aplikacijski profili.

2.2 JavaME

Java ME je Java okolje, prilagojeno za vgrajene naprave, kot so mobilni telefoni, STB vmesniki, navigacijski sistemi za avtomobile, internetne televizije,itd. Izdelano je bilo zaradi omejitev, ki se pojavljajo pri implementaciji aplikacij za majhne naprave.

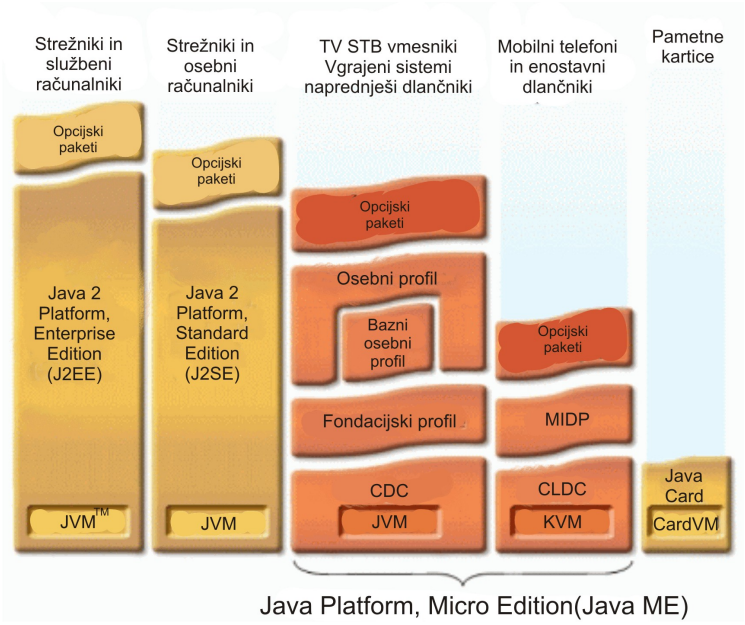
Java ME je ena izmed treh izdaj Java okolja. Prinaša prednosti tehnologije Java(prenosnost kode, objektno usmerjeno programiranje, hiter razvojni cikel) uporabniškim in vgrajenim napravam. Te naprave imajo različne kapacitete spomina, procesorsko moč ter vhodno izhodne zmožnosti. Zato ima arhitektura Java ME različne konfiguracije, profile in opcijske pakete.

Kasneje je bila Java ME razdeljena v dve bazni konfiguraciji - ena za majhne mobilne naprave(CLDC - Connected Limited Device Configuration) ter druga za močnejše mobilne naprave, kot so pametni telefoni in STB vmesniki(CDC - Connected Device Profile).

Slika 2.2 prikazuje pregled Java ME komponent in kako se povezuje z ostalimi Java izdajami.

2.2.1 Profili

Konfiguracije navadno ne dajejo celovite rešitve, zato so bili narejeni profili. Profili dodajo določenim razredom naprav funkcionalnost in potrebne API



Slika 2.2: Java komponente in izdaje.

funkcije. S tem mislimo nabore API funkcij, ki razširjajo funkcionalnost Java ME konfiguracije. Ti profili so:

MIDP Prvi profil, ki je bil narejen, je MIDP. Načrtovan je bil za mobilne telefone, dlančnike in podobne naprave. V kombinaciji s CLDC ponuja dovršeno okolje Java aplikacijam za mobilne telefone in ostale naprave s podobnimi zmožnostmi. MIDP aplikaciji pravimo MIDlet. MIDlet je razred, definiran v MIDP in je superrazred za vse MIDP apliakcije.

FP Fondacijski profil (Foundation Profile) je profil najnižjega nivoja za CDC. Drugi profili so lahko dodani nad njim po potrebi. FP je mišljen za vgrajene naprave brez uporabniškega vmesnika in z mrežnimi zmožnostmi.

PP Osebni profil (Personal Profile - PP) je namenjen napravam, ki potrebujejo uporabniški vmesnik. PP nadomešča PersonalJava tehnologijo in omogoča PersonalJava aplikacijam migracijo na Java ME ogrodje.

2.2.2 MIDlet

MIDlet je MIDP aplikacija za Java ME navidezni stroj. V praksi so to aplikacije ki se izvajajo na vgrajenih sistemih, kot so mobilni telefoni. Tako kot

applet, je tudi MIDlet upravljana aplikacija. Upravlja ga posebna programska oprema, ki je vgrajena v napravo. Upravljanje aplikacije je zelo pomembno, saj jo le tako lahko prekinjamo in kasneje nadaljujemo. Prekinitev, s prednostjo pred aplikacijo, je pri mobilnih telefonih npr. prejeti telefonski klic. V tem primeru je poskrbljeno, da se aplikacija umakne v ozadje.

Pogoj za izvajanje MIDlet-a je naprava, ki implementira vsaj Java ME, CLDC in MIDP. Kot vse druge Java programe je tudi MIDlet-e potrebno prevesti le enkrat, kasneje pa se prenašajo v .jar datotekah. To je glavna distribucijska datoteka za MIDlet. MIDlet ima lahko poleg datoteke tipa .jar še datoteko tipa .jad, ki vsebuje lokacijo in opis vsebine .jar datoteke. Implementacija MIDlet-a lahko zahteva prisotnost .jad datoteke. MIDlet mora izpolniti naslednje zahteve, da se lahko izvaja na mobilnem telefonu:

- glavni razred mora bit podrazred javax.microedition.midlet.MIDlet,
- MIDlet mora biti zapakiran znotraj .jar datoteke,
- .jar datoteka mora biti pred-preverjena,
- v nekaterih primerih mora biti .jar datoteka podpisana s strani mobilne družbe.

Aplikacijam pravimo MIDlet, ker morajo biti vse izpeljane iz posebnega razreda, ki se imenuje MIDlet. MIDlet razred upravlja življenjski cikel aplikacije. Nahaja se v paketu javax.microedition.midlet. MIDlet je lahko v štirih različnih stanjih:

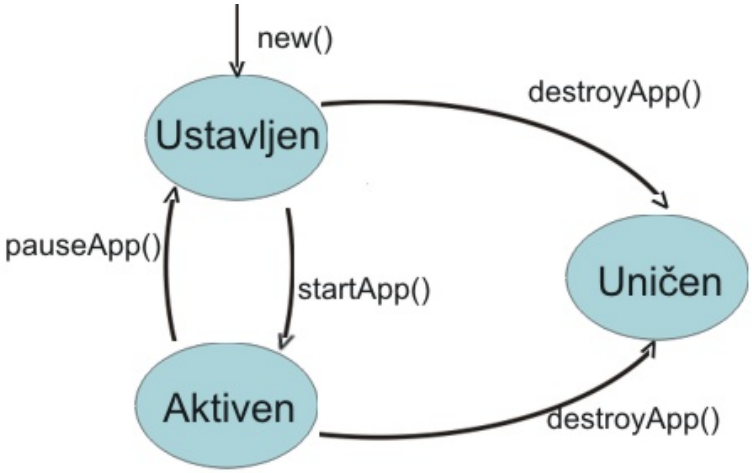
naložen(loaded) Ko je MIDlet naložen na napravo in je poklican konstruktor, se nahaja v stanju loaded. To se lahko zgodi ob kateremkoli času, preden programski upravitelj zažene aplikacijo s klicem metode `startApp()`.

aktiven(active) Po klicu `startApp()` se MIDlet nahaja v aktivnem stanju, dokler aplikacijski upravitelj ne pokliče `pauseApp()` ali `destroyApp()`.

ustavljen(paused) `pauseApp()` začasno ustavi delovanje MIDlet-a in tako preide v stanje, ko je ustavljen.

uničen(destroyed) `destroyApp()` spravi MIDlet v stanje uničenja.

Na sliki 2.3 so prikazana zgoraj opisana stanja, v katerih se lahko nahaja MIDlet.



Slika 2.3: Stanja med katerimi prehaja MIDlet in funkcije, ki povzročajo prehode.

Med delovanjem `pauseApp()` metode aplikacije prekinejo z animacijami in sprostijo vire, ki so nepotrebni, medtem ko se aplikacija začasno ustavi. Tako obnašanje se izogiba konfliktom virov med izvajanjem aplikacije v ozadju in nepotrebni potrošnji baterije. Midlet se lahko ob določenem pogoju izogne prehodu v stanje `destroyed` s klicem točno določene izjeme `MIDletStateChangeException`. MIDlet lahko zahteva ponovno izvajanje aktivnosti s klicem `resumeRequest()`. Če se MIDlet odloči za ustavljeno stanje, naj obvesti aplikacijskega upravitelja s klicem `notifyPaused()`. MIDlet lahko pokliče `notifyDestroyed()` za ustavitev. Klic `System.exit()` ni podprt v MIDP in bo povzročil klic izjeme, namesto da bi ustavil izvajanje aplikacije.

2.3 Java API funkcije za Bluetooth brezžično tehnologijo (JABWT)

JABWT(ang. Java APIs for Bluetooth Wireless Technology) je prva standardna skupina API funkcij za razvijalce Bluetooth aplikacij. Združuje Javo in Bluetooth brezžično tehnologijo. Namenjena je razvoju v Java ME ogrodju. API funkcije JABWT so označene pod JSR-82. V prihodnjosti načrtujejo tudi izdelavo API funkcij za ogrodje Java SE pod označbo JSR - 197, vendar v času pisanja diplomske naloge te še ne obstajajo.

Razvijalci JSR-82 so najprej nameravali razviti API funkcije, ki bi temeljile

na Bluetooth profilih, vendar so se zaradi hitrega nastajanja novih profilov odločili zgolj za podporo baznim Bluetooth profilom, na katerih se lahko gradijo novi Bluetooth profili. Tako so podprli le RFCOMM, L2CAP in OBEX. Torej so podprli profile GAP, SDAP, SPP in GOEP. Naprave ki podpirajo JABWT morajo podpirati aplikacije napisane za okolje Java ME in vsebovati Bluetooth oddajnik. Lastnosti JABWT specifikacij:

- Potrebuje le CLDC knjižnico.
- Deluje na vseh Java ogrodjih, ki podpirajo GCF(ang. Generic Connection Framework).
- OBEX API funkcije so neodvisne od Bluetooth protokolov, saj se lahko OBEX uporablja pri različnih transportih(IrDA, USB, TCP). Iz teh razlogov je OBEX API v svojem paketu `javax.obex`.
- Skrbi, da ne pride do interferenc med Bluetooth aplikacijami. Za to skrbi Bluetooth kontrolni center(BCC - ang. Bluetooth Control Center).
- Zmožnost aplikacij, da so lahko bodisi strežnik bodisi odjemalec, omogoča tvorbo omrežja enakovrednih naprav.
- Možnost razvijanja novih Bluetooth profilov nad RFCOMM, L2CAP in OBEX.

JABWT ni samostojna rešitev. Pogoji pri Java ME konfiguraciji so:

- Vsaj 512 kilobajtov spomina za Java ogrodje (ROM, Flash in RAM).
- Strojna oprema, potrebna za Bluetooth komunikacijo.
- Implementacija Java ME CLDC ali nadnabor CLDC API funkcij, kot je CDC.

JABWT ne dostopa direktno do Bluetooth strojne opreme, temveč preko Bluetooth sklada. Bluetooth sistemske zahteve:

- Morajo biti kvalificirane glede na Bluetooth kvalifikacijo programov za vsaj GAP, SDAP in SPP.
- Podprti morajo biti L2CAP, RFCOMM in SDP sloji.
- BCC mora biti zagotovljena iz strani Bluetooth sklada.

JABWT omogoča naslednje protokole: L2CAP, RFCOMM, SDP, OBEX in LMP. Ima dostop do naslednjih Bluetooth profilov: GAP, SDAP, SPP in GOEP in omogoča naslednje funkcije:

- registriranje storitev,
- odkrivanje naprav in storitev,
- vzpostavitev RFCOMM, L2CAP in OBEX povezav in
- potek teh treh aktivnosti na varen način.

2.3.1 Varnost

Bluetooth tehnologija omogoča različne nivoje varnosti. Obstajajo štirje tipi Bluetooth varnosti : parjenje, avtentifikacija, enkripcija in avtorizacija.

Parjenje je prvi korak pri procesu Bluetooth varnosti. Potreben je, ko se dve napravi prvič srečata in želita vzpostaviti varno povezavo, pri čemer potrebujeta skrivni ključ za avtentifikacijo in enkripcijo. Parjenje zahteva od obeh uporabnikov vpis skrivne kode. Koda je uporabljena zgolj pri prvi avtentifikaciji. Ko se napravi naslednjič srečata, lahko vzpostavita povezavo brez ponovnega parjenja. Proces parjenja je neviden za aplikacijo. Za to skrbi BCC.

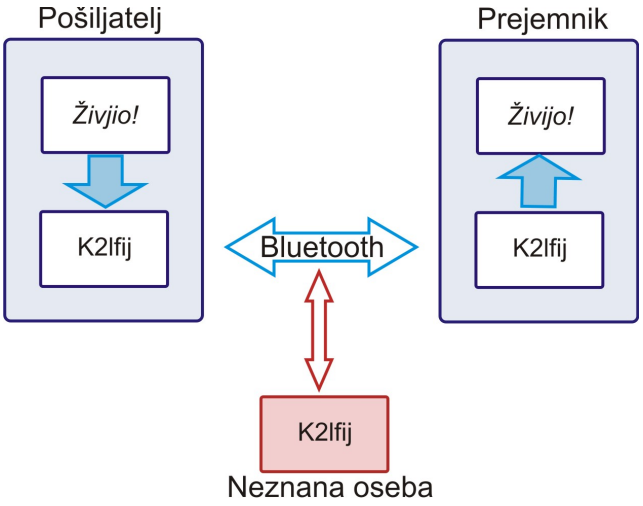
BCC skrbi, da ne pride do konfliktov med Bluetooth aplikacijami. Opravlja tri specifične naloge: odpravljanje konfliktov med aplikacijami, omogočanje sprememb lastnosti Bluetooth naprave ter upravljanje z varnostnimi operacijami, ki lahko potrebujejo uporabnikovo interakcijo.

Bluetooth avtentifikacija preverja identiteto naprav z uporabo “challenge and response“(izziv in odgovor) sheme. Ne identificira uporabnikov, temveč naprave.

Ko želi naprava A avtentificirati napravo B, ji pošlje izziv. Naprava B uporabi deljeno skrivnost pri izzivu in pošlje rezultat nazaj napravi A. Naprava A nato kombinira izziv, ki je bil poslan s svojo deljeno skrivnostjo in ga primerja z rezultatom, prejetim s strani naprave B. Ta proces avtentificira le napravo B napravi A. Naprava B pa nima avtentificirane naprave A.

Ko se konča proces avtentifikacije (v obe smeri) se lahko začne enkripcija. Enkripcija je potrebna za preprečitev prisluškovanja pri komunikaciji med dvema napravama(slika 2.4). Če ena naprava zahteva enkripcijo, jo mora sprejeti tudi druga. Če je le ta ne sprejme, je povezava prekinjena. Če je

pošiljanje v eno smer kriptirano, mora biti kriptirano tudi v drugo. Po drugi strani, če je pošiljanje v eno smer nekriptirano, mora biti nekriptirano tudi v drugo smer.



Slika 2.4: Enkripcija Bluetooth komunikacije in neznana oseba, ki poskuša prisluškovati povezavi.

Druga možnost Bluetooth varnosti je avtorizacija. Avtorizacija je proces, kjer določimo, ali je neka vhodna Bluetooth povezava dovoljena ali zavrnjena. Naprava, ki ji zaupamo, je tista, ki se avtomatično poveže (pri avtorizaciji vedno dovolimo povezavo). Lahko se poveže na katerokoli storitev v lokalni napravi. Če storitev, na katero se povezuje, zahteva avtorizacijo, je zahteva avtomatsko sprejeta, brez spraševanja uporabnika s strani BCC-ja, če se je napravi dovoljeno povezati.

Vsak nivo varnosti je grajen na predhodnjem. Avtentifikacija zahteva parjenje, enkripcija in avtorizacija pa zahtevata avtentifikacijo. Če je na povezavi zahtevana enkripcija in ni bila avtentificirana, vsili JABWT najprej avtentifikacijo.

JABWT dovoli aplikacijam spreminjati nastavitve varnosti na dveh mestih, ob prvi vzpostavitvi povezave in ko je povezava že postavljena. Za spreminjanje pri vzpostavitvi povezave določimo trije parametre v besedi, ki jo podamo `Connection.open()` metodi. BCC skrbi, da so ti tri parametri sprejemljivi in odpravlja konflikte med zahtevami po spremembah varnosti. JABWT ne določa, kako bi moralo biti poskrbljeno za konfliktne zahteve.

Nobeni aplikaciji ne smemo znižati nivoja varnosti, ki ga zahteva. Tako dobimo tri možne implementacije. BCC prisili vse aplikacije v enak nivo varnosti.

Če aplikacija želi drugačen nivo varnosti, ji BCC tega ne dovoli. Druga možnost je, da ustrežemo prvi aplikaciji, ki želi nastaviti nivo varnosti. Če naslednja aplikacija želi višjo varnost, ji zahtevo BCC zavrne. Zadnja možnost je najbolj kompliciranа. Prva aplikacija dobi željeni nivo varnosti. Če naslednja aplikacija zahteva višji nivo, potem JABWT poskuša dvigniti nivo varnosti na povezavi. Če mu to uspe, dobi druga aplikacija zahtevani nivo. Če druga aplikacija zahteva nižji nivo, potem dobi nivo prve aplikacije, ki je višji od željenega.

Pri vsaki Bluetooth povezavi je ena izmed naprav glavna, druga pa podrejena. Glavna naprava vodi "frequency-hopping" sekvenco obeh naprav. Pri večini aplikacij ni pomembno določanje glavne in podrejene naprave, za ustvarjanje novih profilov pa je to pomembno. Pomembno je tudi pri postavljanju ¿¿piconet¡¡ omrežij. To je omrežje z do 7 Bluetooth napravami. V tem primeru je glavna tista naprava, ki poskuša vzpostaviti povezavo, naprava, ki ponuja storitev, pa je podrejena.

2.3.2 RFCOMM

RFCOMM(ang. Radio Frequency Communication) je emulacija RS-232 povezave zaporednih vrat med dvema napravama po brezžični povezavi preko L2CAP. Brez JABWT je komunikacija z oddaljeno naprav preko RFCOMM podobna kot komunikacija preko vtičnic. Podatki se pošiljajo preko tokov. Med napravama, ki sta povezani preko Bluetooth povezave, obstaja ena sama fizična povezava, preko katere lahko poteka več povezav. Podobno kot v ožičenih sistemih.

RFCOMM komunikacija se začne z stavkom `Connector.open()` in besedo, ki jo stavku podamo kot parameter. Vse možne besede, ki jih podajamo `Connector.open()` metodi so oblike:
{shema}:{cilj}{parametri}
Za uporabo RFCOMM je shema enaka `btspp`, cilj in parametri pa so različni, odvisno od tega ali je povezava strežnik ali odjemalec. Vse možne vrednosti parametrov pa so podane v tabeli na sliki 2.5. Vse druge vrednosti sprožijo napako `IllegalArgumentException`.

Vsi ti parametri so opcijski in ni nujno, da jih določimo. Če jih v povezovalni besedi izpustimo, so privzeto enaki `false`. Določene kombinacije niso mogoče, na primer če postavimo enkripcijo na `true` in avtentifikacijo na `false` nam `Connector.open()` vrne izjemo `BluetoothConnectionException`. Če avtentifikacija, enkripcija ali avtorizacija med izvajanjem ne uspejo, nam `Connector.open()` vrne enako izjemo.

IME	VREDNOSTI	STREŽNIK/ODJEMALEC
master	true, false	oba
authenticate	true, false	oba
encrypt	true, false	oba
authorize	true, false	strežnik
name	katerakoli beseda	strežnik

Slika 2.5: Možne vrednosti parametrov povezovalnih besed za RFCOMM.

Postavitev povezave

Za vzpostavitev strežnikove povezave moramo podati v `Connector.open()` besedo v obliki `"btspp://localhost:"+UUID`. Besedi lahko poljubno dodajamo še parametre, ki jih ločimo s podpičji.npr : ";master=true".
`Connector.open()` vrne objekt `StreamConnectionNotifier` in naredi se storitveni zapis. V podatkovno bazo storitvenih zapisov se shrani šele ko je klicana funkcija `acceptAndOpen()`.

Funkcija `acceptAndOpen()` je blokirajoča in čaka, dokler se odjemalec ne poveže na strežnik. Vrne objek `StreamConnection` s katerim lahko aplikacija bere in piše odjemalcu. Povezovalna beseda za odjemalca je podobna kot za strežnik, le da na mesto `localhost` vpišemo Bluetooth naslov naprave na katero se povezujemo in dodamo vrata ki pripadajo storitvi.
Npr. `"btspp://008003DD8901:1;master=true"`

Komunikacija poteka preko `InputStream` in `OutputStream` pri odjemalcu tako:

```
StreamConnection conn=(StreamConnection)Connector.open(connString)
InputStream is = conn.openInputStream();
OutputStream out = conn.openOutputStream();
out.write(byte[]);
in.read(byte[]);
```

Po koncu komunikacije je potrebno zapreti povezavo in vse odprte tokove.

2.3.3 OBEX

IrOBEX (ang. Infrared Object Exchange Protocol) je definiran z IrDA, kot alternativa za HTTP(ang. HyperText Transport Protocol) pri vgrajenih napravah. IrObex omogoča napravam da zahteve in odgvore razdelijo na manjše delčke. Z deljenjem zahtev in odgovorov na manjše delčke podatkov

omogoča da se podatki obdelajo takoj ko so dostavljeni, pošiljanje pa lahko tudi prekinemo.

IrOBEX je neodvisen od transporta. Deluje preko skoraj vseh ostalih transportnih slojev. Najprej so ga uporabljali za prenos preko infrardečih žarkov, sedaj pa se uporablja tudi pri TCP, serijskih in tudi RFCOMM povezavah. Ko govorimo o Bluetoothu se izgubi Ir in pravimo prenosu enostavno OBEX.

OBEX je eden izmed Bluetooth protokolov. Sloni na RFCOMM. SIG(ang. Special Interest Group) je ugotovil da je OBEX dober za grajenje novih Bluetooth profilov, zato so ustvarili profil GOEP, ki definira kako OBEX deluje z Bluetooth napravami. Za razliko od RFCOMM in TCP/IP, ki pošiljata samo bajte, OBEX pošilja strukturirane pakete. Tako se zahteve med seboj dobro ločijo.

OBEX se lahko uporablja v različne namene, npr. za izmenjavo elektronskih vizitk, sinhronizacijo vgrajenih naprav z osebnimi računalniki, tiskanje, itd.

Opis protokola

OBEX se razlikuje od ostalih Bluetooth protokolov. Grajen je na šestih temeljnih operacijah:

CONNECT(poveži) začne povezavo. Odjemalec pošlje strežniku zahtevo za povezavo(operacija CONNECT) in čaka na odgovor.

SETPATH(nastavi pot) operacija nastavi trenutno mapo.

GET(pridobi) ta operacija pridobi pakete preko povezave.

PUT(postavi) ta operacija pošlje pakete preko povezave.

ABORT(prekini) operacija lahko prekine operaciji PUT in GET preden se zaključita.

DISCONNECT(končaj povezavo) zaključi sejo.

Med ukazoma CONNECT in DISCONNECT lahko odjemalec pošlje katerokoli število zahtev SETPATH, GET, ABORT ali PUT. PUT in GET sta sestavljeni iz več manjših operacij, zato jih lahko prekinjamo. Pri vsaki zahtevi ali odgovoru se pošlje tudi glava. Glava je lahko:

NAME(ime) je ime objekta.

LENGTH(dolžina) je dolžina objekta.

TYPE(tip) je tip objekta.

COUNT(štetje) uporablja zahteva CONNECT za določitev števila objektov, ki se bodo pošiljali ali sprejemali.

DESCRIPTION(opis) je kratek opis objekta.

HTTP določa HTTP glavo.

BODY(telo) določa del objekta.

END OF BODY(konec telesa) določa zadnji del objekta.

TIME_ISO_8601 timestamp objekta.

TIME_4_BYTE timestamp objekta.

TARGET ciljna OBEX storitev.

WHO storitev, ki procesira zahtevo.

OBJECT_CLASS razred objekta.

APPLICATION_PARAMETER posebni parametri aplikacije.

OBEX specifikacija določa kako so glave kodirane. Npr. glava NAME mora biti Unicode beseda. BODY in END OF BODY se uporabljata za pošiljanje ali pridobivanje iz strežnika preko metod PUT in GET.

Specifikacija dopušča poleg zgoraj naštetih glav še 64 glav, ki jih lahko definirajo uporabniki. OBEX specifikacija definira še dve dodatni operaciji PUT-DELETE in CREATE-EMPTY. PUT-DELETE je operacija PUT z določeno glavo NAME in brez BODY. Ta operacija se uporablja pri sporočanju strežniku, da se mora datoteka s tem imenom izbrisati. CREATE-EMPTY operacija je tudi PUT operacija, ki vsebuje NAME in END OF BODY brez podatkov. Tako sporoča odjemalec strežniku naj ustvari datoteko z imenom iz NAME brez podatkov.

Primer seje

Vsaka OBEX seja se začne tako, da odjemalec pošlje strežniku zahtevo CONNECT. Če želi, lahko pošlje tudi dodatne glave. Ko strežnik sprejme zahtevo, obdela glave in se odloči za sprejetje ali zavrnitev povezave. Če sprejme povezavo, odgovori z OK odgovorom-SUCCESS(uspešno) koda odgovora. Če zahtevo zavrne, odgovori s HTTP kodo odgovora, v kateri pojasnjuje zakaj je prišlo do zavrnitve.

Med povezavo odjemalec pošilja različno število PUT in GET operacij. Ker obe potekata podobno, bomo opisali za primer le potek PUT zahteve.

Odjemalec pošlje datoteko strežniku z operacijo PUT. Če je datoteka velika, jo mora razdeliti na več manjših delov. Najprej pošlje začetno PUT zahtevo, ki vsebuje glavo NAME z imenom datoteke in BODY glavo, ki vsebuje prvi delček datoteke. Ko strežnik to sprejme, shrani podatke in odgovori s CONTINUE(nadaljuj). Nato odjemalec pošilja na enak način še ostale delčke datoteke s PUT operacijami in glavami BODY. Pri zadnjem delu datoteke pa odjemalec pošlje zahtevo PUT z glavo END OF BODY namesto BODY. Tako strežnik ve, da je prejel zadnji del datoteke in ko ga shrani, odgovori namesto CONTINUE z OK(SUCCESS). Tako odjemalec ve, da je bila datoteka uspešno prejeta.

Povezava se zaključi, ko odjemalec strežniku pošlje zahtevo DISCONNECT. Ko jo strežnik sprejme, sprosti vse vire in nato pošlje odgovor OK. Ko odjemalec to sprejme, se povezava zaključi.

OBEX API funkcije

OBEX API funkcije so različne od ostalih API funkcij, saj zaradi vsebovanja nižje-nivojskega vmesnika razvijalcem dovoljujejo večji nadzor nad vsako zahtevo in odgovorom.

Še vedno pa API funkcije skrivajo določene podrobnosti protokola. Skrbijo za translacijo glav v bajte. Skrivajo tudi podrobnosti ukaza CONNECT. API funkcije skrbijo za deljenje zahteve na manjše delčke, na več PUT ali GET zahtev. Tako lahko razvijalec pošlje celotno datoteko, API funkcije pa poskrbijo za ustrezno razdelitev v manjše delčke za pošiljanje.

OBEX API je grajen na GFC. Dodaja tri vmesnike, ki razširjajo `javax.microedition.io.Connection`. `Connector.open()` vrne objekt `javax.obex.ClientSession`, ko mu podamo odjemalčevo besedo za povezavo. Kadar želimo strežnik, nam vrne objekt tipa `javax.obex.SessionNotifier`. `javax.obex.Operation` se uporablja za PUT in GET zahteve. Zaradi teh treh novih vmesnikov definira API `javax.obex.Autenticator` in

`javax.obex.HeaderSet`. `Authenticator` je implementiran v aplikacijah, ki želijo avtentifikacijo. `HeaderSet` pa združuje več glav. Vse glave, razen BODY, END OF BODY, CONNECTION-ID, AUTHENTICATION_CHALLENGE in AUTHENTICATION_RESPONSE, lahko nastavljamo. API vsebuje tri nove razrede: `javax.obex.PasswordAuthentication` , `javax.obex.ResponseCodes` in `javax.obex.ServerRequestHandler`. Prvi vsebuje pare uporabniških imen in gesel za avtentifikacijo, drugi definira vse kode odgovorov, zadnji pa definira metode, ki se kličejo, ko strežnik dobi različne OBEX zahteve.

Besede, ki jih podamo `Connector.open()`, so oblike {transport}obex://{cilj}{parametri}

Kadar je transportni protokol TCP/IP, je začetni del besede `tcpobex://`. Ko odpremo odjemalčevo povezavo, sta cilj strežnikov IP in vrata. Ko odpiramo povezavo strežnika, pa je cilj le številka vrat. Če je RFCOMM transportni protokol, se beseda začne z `btgoep://`. Za povezave strežnika je cilj UUID storitve, za odjemalca pa Bluetooth naslov in RFCOMM številka kanala. Ko podamo besedo odjemalca, nam `Connector.open()`vrne objekt tipa `ClietnSession` in transportna povezava je ustvarjena. Za plastno povezavo pa je potrebno klicati `ClientSession.connect()`. Preden zapremo transportno povezavo moramo klicati `ClientSession.disconnect()`.

Če odpiramo povezavo strežnika, gre potek vzpostavitve nekoliko drugače. `Connector.open()` vrne objekt tipa `SessionNotifier`. Povezave pa lahko uporabnik sprejme s klicem `SessionNotifier.acceptAndOpen()`. `acceptAndOpen()` ima za argument `ServerRequestHandler` in opcijski `Authenticator`. `acceptAndOpen()` metoda vrne objekt tipa `Connection`, ki predstavlja nivo prenosne povezave.

Upravljanje OBEX glav

OBEX opravlja vse komunikacije z glavami. JABWT vsebuje različne metode za branje in pisanje glav skozi objekt tipa `HeaderSet`. Izjema so BODY, END OF BODY, AUTHENTICATION_CHALLENGE, AUTHENTICATION_RESPONSE in CONNECTION-ID, do katerih lahko razvijalci dostopajo preko posebnih metod.

Razvijalcem ni dovoljena implementacija lastne `HeaderSet` implementacije. `HeaderSet` se kreira s klicem funkcije `ClientSession.createHeaderSet()`. Nato lahko glave nastavljamo enostavno s klicem funkcije `setHeader()` in parametroma identifikacija glave in vrednost glave. Za branje glav pa kličemo

metodo `HeaderSet.getHeader()`. Če podamo nedefinirane parametre, dobimo napako `IllegalArgumentException`. 643 glav je namenjenih za uporabniško definirane glave. S funkcijo `getHeaderList()` dobimo tabelo števil, ki predstavlja identifikatorje glav. Nikdar ne vrne NULL. Če ni nobena glava definirana, nam vrne prazno tabelo. AUTHENTICATION_CHALLENGE in AUTHENTICATION_RESPONSE glavi sta dostopni preko `Authenticator` vmesnika. Glava CONNECTION-ID je dostopna s funkcijama `getConnectionID(), setConnectionID()`, pri čemer je CONNECTION-ID unikaten. Uporablja se za ločevanje storitev, ki jih nudi isti OBEX obveščevalni objekt. Z `getResponseCode()` dobimo kodo odgovora na poslano zahtevo. Ko strežnik dobi zahtevo, se avtomatično kliče ustrazna metoda za sprejeto zahtevo. Možne metode so onConnect(), onDelete(), onDisconnect(),onSetPath(), onPut() in onGet().

OBEX avtentifikacija

Deluje na principu izziva in odgovora. Za avtentifikacijo se pošlje na drugi konec OBEX povezave zahteva z glavo AUTHENTICATION_CHALLENGE. V odgovor se pošlje glava AUTHENTICATION_RESPONSE z geslom in uporabniškim imenom. Če se uporablja avtentifikacija, mora biti kreiran `Autenticator` s funkcijo `ClientSession.setAuthenticator()` ali `SessionNotifier.acceptAndOpen()`.

2.3.4 L2CAP

L2CAP je razdeljevalni sloj, ki dovoljuje različnim višjenivojskim protokolom in aplikacijam uporabo Bluetooth komunikacije. L2CAP uporablja HCI za komuniciranje z osnovnopasovnim nivojem v Bluetooth modulu. Vsi podatkovni Bluetooth komunikatorji uporabljajo L2CAP, z izjemo zvočnih komunikatorjev.

L2CAP ustvari kanal za povezavo, po katerem se prenašajo podatkovni paketi. Prenos je možen v obe smeri in med več destinacijami. Obstaja limit velikosti paketa, namenjenega za določeno destinacijo, ki mu pravimo MTU(maximum transmission unit - maximalna prenosna enota). Npr. paket, ki se prenaša k strežniku, ima lahko večji MTU kot paket, namenjen odjemalcu.

Osnovnopasovni nivo ustanovi ACL povezavo med napravama. Obstaja lahko natanko ena ACL povezava med napravama. ACL povezava in osnovnopasovni nivo nudita infrastrukturo, potrebno za podporo L2CAP. L2CAP paketi moraj biti pretvorjeni v enega ali več osnovnopasovnih paketov za prenos

preko ACL povezave. Obstajajo različne velikosti osnovnopasovnih paketov, vendar je največja velikost enaka 339 bajtov. To je veliko manj kot največji možni paket za L2CAP, ki je enak 65,535 bajtov. Zaradi abstrakcije kanalov in paketov L2CAP omogoča višjim nivojem lažjo uporabo Bluetooth komunikacije. L2CAP omogoča tudi enosmerne kanale, vendar jih JABWT ne podpira.

Uporaba API funkcij

Podobno kot pri OBEX in RFCOMM, uporablja L2CAP za čakanje na povezovanje odjemalčev objekt `L2CAPConnectionNotifier`. Notifier vrne objekt tipa `L2CAPConnection` za dostop do kanala med strežnikom in odjemalcem. Enak objekt vrne odjemalcu `Connector.open()`. Ker L2CAP pošilja pakete, ne more uporabljati tokov, tako kot RFCOMM (`StreamConnection`). Za L2CAP sta bili ustvarjeni metodi `send()`(pošlji) in `receive()`(sprejmi). Metoda `receive()` blokira, dokler ne prebere L2CAP paketa ali pa se kanal zapre. `L2CAPConnection` vmesnik vsebuje tudi metodo `ready()`, s katero lahko preverimo, če bo klic na `receive()` blokirajoč. Metoda `ready()` vrne `true`, če je L2CAP razpoložljiv za takojšnje branje s strani metode `receive()`, brez blokiranja.

Tudi L2CAP povezava se začne s klicem na `Connector.open()`, ki mu kot argument podamo povezovalno besedo. Struktura povezovalne besede je podobna kot pri RFCOMM in OBEX, le da se začne z `btl2cap://`. Primer povezovalne besede je `btl2cap://00560D03234C:1001;receiveMTU=512`. `1001` je v tem primeru PSM vrednost strežniške apliakcije. PSM je pridobljen iz storitvenega zapisa. PSM vrednost pove L2CAP sloju, na katero strežniško aplikacijo se želi odjemalec povezati. Višjenivojski protokoli kot so RFCOMM in SDP imajo permanentno dodeljeno PSM vrednost. PSM je aplikacijam dinamično dodeljen in je lahko ob ponovnem zagonu aplikacije vsakič drugačen. PSM vrednosti so interpretirane kot šestnajstiške vrednosti.

Povezovalna beseda lahko vsebuje parametre, kot smo jih določali pri OBEX in RFCOMM (enkripcija, autentifikacija, avtorizacija,...). Lahko pa določamo še dva dodatna parametra `receiveMTU`(največja velikost paketa v bajtih, ki jo bo sprejel odjemalec od strežnika) in `transmitMTU`(največji možni paket v bajtih, ki ga bo odjemalec poslal strežniku).

Največja možna velikost paketa, ki jo lahko sprejme Bluetooth sklad, je shranjena v lastnosti `bluetooth.l2cap.receiveMTU.max`. Aplikacija mora izbrati, če želi upravljati s tako velikimi paketi, kot jih omogoča Bluetooth sklad ali pa določi manjšo velikost paketa.

V splošnem je boljše, če se velikosti paketov za sprejemanje in pošiljanje ne določa v povezovalni besedi, razen če je to absolutno potrebno. Če niso določene v povezovalni besedi, so MTU vrednosti dodeljene avtomatično. Navadno je avtomatično nastavljena velikost MTU vrednosti na 672 definirana s strani L2CAP specifikacije. Pravila za MTU:

Prvo pravilo MTU: Vrednosti `receiveMTU` in `transmitMTU` ne smeta biti manjši, kot je `L2CAPConnection.MINIMUM_MTU`, ki je enak 48 ali večji kot maksimalna vrednost L2CAP paketa, ki je enaka 65,535.

Drugo pravilo MTU: Velikost `receiveMTU` mora biti manjša ali enaka kot največji možni paket, ki ga lahko sprejme Bluetooth sklad naprave, na kateri teče aplikacija. To vrednost lahko dobimo med izvajanjem s klicem sistemskih lastnosti.

Tretje pravilo MTU: Za prenos s `send(byte[] out)`, `out` ne sme biti večji kot `L2CAPConnection.getTransmitMTU()`. Če je večji, so bajti odvrženi, preden se L2CAP paket pošlje.

Četrto pravilo MTU: Za prejetje paketov z `receive(byte[] in)` se alocira bajtno polje velikosti `L2CAPConnection.getReceiveMTU()`. Če se uporablja manjši `in`, so odvečni bajti zavrženi.

Povezovanje aplikacije je lahko neuspešno zaradi neprimernih MTU vrednosti. To se zgodi v primeru, če pošlje aplikacija A večji paket(`transmitMTUa`), kot ga lahko sprejme aplikacija B(`receiveMTUb`). Glavni vzrok te napake je v tem, da L2CAP protokol ne omogoča poizvedbe o MTU na oddaljeni napravi pred vzpostavitvijo povezave. Temu se lahko izognemo tako, da MTU vrednosti ne nastavljamo ali pa določimo `transmitMTU` naprave A enako kot `receiveMTU` naprave B.

V Bluetooth specifikaciji 1.1 ni bilo L2CAP nadzora poteka. Osnovnopasovni sloj vsebuje nadzor poteka za ACL povezavo. Na žalost ta nadzor ni dovolj. L2CAP je razdeljevalni sloj, ki nudi več L2CAP kanalov, namenjenih več višjenivojskim aplikacijam in protokolom. Če prihajajo paketi hitreje, kot so lahko obdelani, se L2CAP medpomnilnik napolni. V tem primeru L2CAP ukrepa na dva načina: ali prekine vse prihajajoče pakete v ACL povezavi (tudi tiste, ki nimajo problemov) ali zavrže nekatere L2CAP pakete, ker zanje ni prostora v medpomnilniku. Drugi način ni primeren, ker vodi do prejemanja nepravilnih podatkov, medtem ko prvi lahko vodi do mrtvih zank. V Bluetooth specifikaciji 1.2 je implementiran nadzor poteka, vendar ni zagotovljeno, da vsak Bluetooth sklad implementira nadzor poteka.

Tudi pri L2CAP poteka uporaba API funkcij podobno kot pri prejšnjih protokolih. Začne se s stavkom `Connector.open()`, ki mu podamo ustrezno povezovalno besedo. Ta vrne objekt tipa `L2CAPConnectorNotifier` ali `L2CAPConnection`. V primeru strežnika aplikacija čaka na vhodne povezave pri stavku `acceptAndOpen()`. V primeru, da sta MTU vrednosti strežnika in odjemalca nekompatibilna, pride do napake `BluetoothConnectionException`. Pošiljanje podatkov med strežnikom in odjemalcem poteka s funkcijama `L2CAPConnection.send()`(za pošiljanje) ali `L2CAPConnection.receive()`(za sprejemanje).

L2CAP omogoča uporabo enakih funkcij tako strežniku kot odjemalcu. To pomeni, da imata oba na razpolago vse funkcije. Oba lahko kreirata objekt `L2CAPConnection` in pošiljata pakete kadarkoli želita.

Kontrola pretoka

Začetne verzije Bluetooth specifikacije niso vsebovale kontrole pretoka. V verziji 1.2 pa se je kontrola pretoka pojavila, vendar ni bila obvezna. Navadno Bluetooth protokoli in profili, ki uporabljajo L2CAP, vsebujejo lastne kontrole pretoka.

2.3.5 Odkrivanje sosednjih naprav

Vsaka Bluetooth naprava mora biti sposobna odkrivati ostale Bluetooth naprave v dosegu oddajnika. Sposobna mora biti tudi pridobivati storitve, ki jih nudijo le-te naprave, saj jih le tako lahko uporablja. Bluetooth specifikacija deli odkrivanje naprav in odkrivanje storitev na napravah v dva različna procesa. Odkrivanje naprav(device discovery) poišče vse Bluetooth naprave, z vključenim oddajnikom, ki so vidne in v dosegu oddajnika. Odkrivanje storitev pa vrne vse storitve na že najdeni sosednji napravi.

Naprave v bližini odgovorijo na poizvedbo(inquiry) s svojim Bluetooth naslovom in razredom naprave. Bluetooth naslov je 6-bajtna unikatna identiteta, dodeljena vsaki Bluetooth napravi s strani proizvajalca. Razred naprave opisuje tip Bluetooth naprave(npr. mobilni telefon, osebni računalnik,...).

Bluetooth SIG je definiral dva tipa poizvedb: omejena in splošna. Splošen pristop poišče vse naprave v dosegu našega oddajnika. Omejen pristop pa poišče vse naprave v okolici, ki so vidne le za omejen čas.

Vsaka Bluetooth naprava je lahko vidna, omejeno vidna ali splošno vidna. Če naprava ni vidna, ne more odgovoriti na nobeno poizvedbo. Če je omejeno vidna, odgovarja na splošne in omejene poizvedbe, če je splošno vidna, pa le

na splošne poizvedbe.

Odkrivanje naprav se začne z definiranjem tipa poizvedbe: splošno ali omejeno. API funkcije vrnejo aplikaciji vse najdene naprave preko dogodka `deviceDiscovered()`. Aplikacija pridobi naprave in razrede katerim pripadajo. Razred naprave določa tip naprave, ki odgovarja in storitve, ki jih naprava vsebuje. Razred naprave je sestavljen iz glavnega storitvenega razreda, glavnega razreda naprave in pomožnega razreda naprave. Glavni storitveni razred definira storitve, ki so na voljo na napravi. Glavne storitve, ki jih je Bluetooth SIG definiral:

- pozicioniranje,
- povezovanje,
- zajemanje,
- prenos objektov,
- audio,
- informacije,
- omejeno odkrivanje,
- izvajanje in
- telefonija.

Naprava lahko ima več glavnih storitvenih razredov. To pomeni, da lahko na napravi istočasno teče avdio in telefonska storitev. Vsaka naprava ima lahko le en glavni razred naprave. Glavni razredi naprav, ki jih je definiral SIG so:

- osebni računalniki (namizni, prenosnik,...),
- telefoni (mobilni telefon, modem, ...),
- LAN/mrežna dostopna točka,
- audio/video (zvočniki, stereo, slušalke,...),
- periferne naprave (miška, igralna palica, tipkovnica,...),
- naprave ki upravjajo s slikami (tiskalnik, optični čitalec, kamera, zaslon,...),

- razne naprave,
- žepne naprave,
- igrače in
- zdravstvene naprave.

Pomožni razred naprave je definiran na glavnem razredu naprave. To je le bolj natančen opis naprave. Npr. če je naprava osebni računalnik, nam pomožni razred naprave pove, za kateri tip računalnika gre(namizni, prenosni,...). Ta sistem omogoča razvijalcem, da iščejo storitve le na določenih tipih naprav, ki jih zanimajo.

Za omejevanje iskanja sosednjih naprav obstaja še en podoben način - obnovitev oddaljenih naprav. Kadar naprava izvaja poizvedbo, potrebuje od 8 do 10 sekund, da z verjetnostjo 95% odkrije vse sosednje naprave. Poizvedba je tudi potratna z energijo. JABWT omogoča obnovitev spiska naprav, ki so najbrž v okolici, brez poizvedbe. Tem napravam pravimo predefinirane. Obstajata dva tipa predefiniranih naprav: pre-known in cached. Pre-known so tiste naprave, s katerimi naprava pogosto sodeluje. Nastavljene so v BCC. To seveda še ne zagotavlja, da so te naprave dejansko v okolici. Cached naprave so tiste naprave, ki smo jih našli s prejšnjimi poizvedbami, ki so potekale na tej napravi in jih je lahko izvajala tudi kakšna druga aplikacija.

Uporaba API funkcij

Za programiranje potrebujemo podatke naše Bluetooth naprave. Za to je bil ustvarjen razred `LocalDevice`, ki nam omogoča dostop do lokalne Bluetooth naprave. Objekt tega tipa dobimo s klicem funkcije `LocalDevice.getLocalDevice()`. V primeru da Bluetooth sklad ali oddajnik ne delujeta pravilno, nam ta metoda vrne napako `BluetoothStateException`. Ko enkrat dobimo objekt tipa `LocalDevice`, lahko iz njega dobimo podatke, kot so prijateljsko ime(s klicem funkcije `getFriendlyName()`), Bluetooth naslov(s klicem funkcije `getBluetoothAddress()`), trenutni način vidnosti Bluetooth naprave (s klicem funkcije `getDiscoverable()`) in zapis razreda naprave (s klicem funkcije `getDeviceClass()`). Razred `DiscoveryAgent` vsebuje konstante za splošno vidnost(GIAC-General Inquiry Access Code), omejeno vidnost(LIAC - Limited Inquiry Access Code) in nevidnost (NOT_DISCOVERABLE) Bluetooth naprave.

Način vidnosti naprave lahko tudi spreminjamo s klicem funkcije `setDiscoverableMode()`. Ta funkcija nam vrne `true`, če je način vidnosti

uspešno spremenjen in `false`, če BCC zavrne zahtevo ali pa lokalna naprava ne podpira željenega načina vidnosti. V primeru da je naprava v stanju, kjer sprememba načina vidnosti ni vidna, nam funkcija vrne napako `BluetoothStateException`. V primeru konfliktnih zahtev po spremembi načina vidnosti, poskrbi za rešitev BCC.

Tudi lokalna naprava ima funkcijo `LocalDevice.getProperty()`, podobno kot `System.getProperty()`, s katero lahko dobimo podatke o Bluetooth napravi. Funkciji podamo parameter tipa `String`(beseda) z željeno lastnostjo, katere vrednost želimo izvedeti.

Spisek lastnosti, ki jih lahko beremo:

bluetooth.api.version Verzija JABWT.

bluetooth.master.switch Ali je dovoljeno preklapljanje med gospodar/suženj?

bluetooth.sd.attr.retrievable.max Max. št. atributov storitev pri storitvenem zapisu.

bluetooth.l2cap.recieveMTU.max Največja velikost sprejetja MTU v bajtih pri L2CAP.

bluetooth.connected.devices.max Največje možno število povezav.

bluetooth.sd.trans.max Največje število tekmovalnih transakcij pri iskanju storitev.

bluetooth.connected.inquiry.scan Ali lahko naprava odgovori na poizvedbo medtem, ko ima vzpostavljeno povezavo z drugo napravo?

bluetooth.connected.page.scan Ali lahko sprejme povezavo z oddaljeno napravo, če je že povezana z drugo napravo?

bluetooth.connected.inquiry Ali lahko naprava začne poizvedbo, če je že povezana na drugo napravo?

bluetooth.connected.page Ali lahko naprava vzpostavi povezavo z oddaljeno napravo, če je že povezana z drugo napravo?

Vsaka naprava ima samo en objekt tipa `DiscoveryAgent`, ki vsebuje metode za začetek iskanja sosednjih naprav in storitev na njih. Ta objekt uporablja drugi objekt `DiscoveryListener`, preko katerega prejema najdene naprave in storitve. Objekt tipa `DiscoveryAgent` dobimo s klicem metode `LocalDevice.getDiscoveryAgent()`.

Obnovitev spiska oddaljenih naprav dobimo z metodo `DiscoveryAgent.retrieveDevices()`. Ta metoda nam vrne spisek pre-known naprav, če uporabimo argument `DiscoveryAgent.PREKNOWN` in spisek cached naprav, če uporabimo argument `DiscoveryAgent.CACHED`.

Vsaka naprava, ki želi opraviti iskanje sosednjih naprav, mora implementirati `DiscoveryListener`. Ta vmesnik potrebuje implementacijo metod `deviceDiscovered()` in `inquiryCompleted()`.

Iskanje sosednjih naprav se začne s klicem funkcije `startInquiry()`, kateri podamo kot argument tip poizvedbe in implementacijo `DiscoveryListener` vmesnika. Aplikacija dobi sosednje naprave preko dogodkov v funkciji `deviceDiscovered()`. Ta funkcija se kliče vsakič, ko najdemo oddaljeno napravo in nam vrne objekt tipa `RemoteDevice`, ki vsebuje podatke oddaljene naprave(Bluetooth naslov, ime,...).

JABWT omogoča prekinitev poizvedb. Npr. če že najdemo iskano napravo, lahko prenehamo z iskanjem ostalih naprav v okolici. Prekinitev poizvedb naredimo s klicem metode `cancelInquiry()`. Za preprečitev prekinitve poizvedbe ene aplikacije s strani druge moramo funkciji podati kot parameter objekt
`DiscoveryListener`. Kadar se poizvedba konča, se kliče funkcija `inquiryCompleted()`, ki nam pove razlog zaključitve procesa.

Objekt `RemoteDevice` vsebuje informacije o oddaljeni napravi. Poleg že prej omenjenega načina pridobitve lahko tak objekt dobimo z metodo `RemoteDevice.getRemoteDevice()` ali z lastnim razredom, ki razširja razred `RemoteDevice`.

`RemoteDevice.getRemoteDevice()` vzame kot argument Bluetooth povezavo z oddaljeno napravo. Ko imamo objekt `RemoteDevice`, lahko kličemo metode `getBluetoothAddress()`, `isTrustedDevice()` in `getFriendlyName()`.

Potem ko je povezava vzpostavljena, lahko specificiramo Bluetooth varnost preko objekta `RemoteDevice` z metodami `authenticate(), encrypt()` in `authorize()`. Kakšna je nastavljena varnost, pa lahko preverimo s funkcijami `isAuthenticated(), isEncrypted()` in `isAuthorized()`. Obstaja še funkcija za preverjanje, če oddaljeni napravi zaupamo in sicer `isTrustedDevice()`.

Nepotrebnemu iskanju storitev oddaljene naprave se lahko izognemo s pomočjo objekta `DeviceClass`, ki nam ga vrne funkcija `deviceDiscovered()`. Ta objekt vsebuje funkcijo `getServiceClasses()`, ki vrne število, iz katerega po bitih razberemo, katere storitve vsebuje. V tabeli na sliki 2.6 je spisek bitov in storitev, katerim pripadajo.

Iskanje lahko omejimo tudi samo na določene tipe naprav. Pri tem si

RAZRED STORITVE	BITNA VREDNOST	ŠESTNAJSTIŠKA VREDNOST
Omejen način iskanja	13	0x2000
Pozicioniranje	16	0x10000
Povezovanje	17	0x20000
Izvajanje	18	0x40000
zajemanje	19	0x80000
Prenos podatkov	20	0x100000
Audio	21	0x200000
Telefonija	22	0x400000
Informacije	23	0x800000

Slika 2.6: Storitve in njihova bitna ter šestnajstiška vrednost.

pomagamo s funkcijo `getMajorDeviceClass()` in `getMinorDeviceClass()`. V tabeli na sliki 2.7 so podane vrednosti razredov v šestnastiškem sistemu in njihova imena.

GLAVNI RAZRED NAPRAVE	ŠESTNAJSTIŠKA VREDNOST
računalnik	0x100
telefon	0x200
LAN/mrežna dostopna točka	0x300
audio/video	0x400
periferna naprava	0x500
naprava za slike	0x600
žepna	0x700
igrača	0x800
zdravstvena	0x900
razno	0x000

Slika 2.7: Seznam razredov naprav in njihovih šestnajstiških vrednosti.

2.3.6 Odkrivanje storitev na sosednjih napravah

Ko najdemo vse naprave v dosegu Bluetooth oddajnika, moramo poiskati še vse storitve, ki potekajo na napravah, šele nato se lahko povežemo nanje. Na napravi lahko poteka le eno odkrivanje storitev naenkrat. Zahteva ni blokirajoča. Za vsako najdeno oddaljeno napravo poiščemo storitve, ki potekajo na le-tej. Naprava, ki išče storitve na ostalih napravah, vpraša oddaljeno napravo, če ima storitev definirano s storitvenim zapisom s specifičnimi atributi. Če ima oddaljena naprava tako storitev, vrne storitveni zapis, ki opisuje to storitev.

Storitveni zapis ima različne atribute, ki podajajo dodatne informacije za določeno storitev. Lahko vsebuje karkoli, tudi podatke o tem, kako se povezati na storitev.

Odkrivanje storitev poteka po principu strežnik-odjemalec. Odjemalec, ki išče storitve, pošlje zahtevo oddaljeni napravi(strežniku), ki preveri ali ima storitev, ki ustreza zahtevam(storitve išče v SDDB).

JABWT omogoča iskanje vseh storitvenih zapisov, ki ustrezajo določenim pogojem na oddaljeni Bluetooth napravi. Ko aplikacija išče storitev, zagotavlja set UUID-jev, ki jih iščemo. UUID je sekvenca bitov, ki unikatno identificira karakteristiko storitve. Nekateri UUID-ji so specificirani v Bluetooth specifikaciji. Ostali UUID-ji so definirani na "service-by-service" bazi. Ko je storitev najdena, nam JABWT omogoča pridobitev nabora atributov. Ko je storitev, ki ustreza specificiranim UUID-jem najdena, nam preko dogodka `serviceDiscovered()` vrne svoj storitveni zapis.

JABWT definira tudi blokirajočo metodo `selectService()`, ki opravi oboje, iskanje naprav in storitev na njih. Aplikacija določi le en UUID, `selectService()` pa vrne besedo za povezavo, ki jo podamo `Connector.open()` funkciji.

Navadno mora aplikacija na strežniku teči, da se odjemalec lahko poveže nanjo. Takim aplikacijam pravimo po JABWT specifikaciji "run-before-connect". Druga vrsta storitev pa je "connect-anytime"(poveži-kadarkoli).

Metode `open()`, `acceptAndOpen()` in `close()` so del GCF definirane od CLDC. Te metode poskrbijo, da so storitveni zapisi avtomatično narejeni, shranjeni in izbrisani iz SDDB na naslednji način:

- `Connector.open(String url)` ustvari nov storitveni zapis če `url` začne z: `btspp://localhost:`,`btgoep://localhost:` ali `btl2cap://localhost:`.
- Prvič ko se sporočilo `acceptAndOpen()` pošlje obveščevalnemu objektu, se doda kopija storitvenega zapisa v SDDB.
- Ko se `close()` sporočilo pošlje obveščevalnemu objektu, se storitveni zapis odstrani iz SDDB.

Registracija storitev

Vsaka storitev mora kreirati svoj storitveni zapis, če želi prejemati zunanje povezave. Navadno je dovolj storitveni zapis, ki se avtomatično kreira.

SPP specifikacija vsebuje predlogo za storitveni zapis. JABWT uporablja to predlogo za ustvarjanje novih storitvenih zapisov in vanj vstavi ustrezne

vrednosti za RFCOMM identifikator kanala. Tako nastane minimalen, a zadosten storitveni zapis. Bluetooth SDP uporablja za identifikatorje atributov vrednosti med 0 in $2^{16} - 1(65535)$.

Vsaka vrednost atributa je podatkovni element(DataElement), to je podatkovna struktua, ki vsebuje tip objekta in vrednost. Če odpremo povezavo s `Connector.open("btspp://localhost: 123E34f852e233275e3dd3b0318d76ec;name=ime")`, se kreira objekt tipa string z vrednostjo `ime`, ki se uporabi kot atribut za `ServiceName` atribut storitvenega zapisa. Nekaj identifikatorjev atributov storitvenega zapisa je podanih v tabeli na sliki 2.8.

IME	ID	OPIS
ServiceRecordHandle	0x0000	unikatna identifikacija storitve na napravi
ServiceClassIDList	0x0001	definira razred, ki opisuje storitev
ServiceID	0x0003	unikatno identificira storitev
ProtocolDescriptionList	0x0004	opisuje protokol za povezavo na storitev
ServiceInfoTimeToLive	0x0007	definira koliko časa bo storitev veljavna
ServiceAvailability	0x0008	relativna razpoložljivost storitve
BluetoothProfileDescriptionList	0x0009	specifikacija profilov storitve
DocumentationURL	0x000A	URL, ki kaže na dokumentacijo storitve
IconURL	0x000C	URL, ki kaže na ikono storitve
ServiceName	0x0100	ime storitve
ServiceDescription	0x0101	opis storitve

Slika 2.8: Seznam atributov, ki pripadajo storitvenemu zapisu.

DataElement je lahko tipa Null, Integer, Unsigned Integer, URL, UUID, String, Boolean, DATALT-alternativni DataElement ali DATASEQ-sekvenca DataElement.

Ko odjemalec išče ustrezno storitev, išče po seznamu ServiceClassIDList ustrezen UUID. Bluetooth Asigned Numbers definira tudi krajše 16-bitne UUID-je. Npr. za RFCOMM je 128-bitni UUID enak 0x0000000300001000800000805F9B34FB, 16-bitni pa 0x0003.

Strežniki, ki uporabljajo za komunikacijo L2CAP ali OBEX, rabijo drugačne storitvene zapise, kot smo jih do sedaj opisali. Tudi v teh primerih je JABWT odgovoren za avtomatično generiranje storitvenih zapisov. V tem primeru vsebuje drugačne podatke. Še vedno vsebuje atribute ServiceClassIDList, ProtocolDescriptionList in ServiceName. V ServiceClassIDList ni več SerialPort atributa. Iz ProtocolDescriptionList je odmaknjen RFCOMM. L2CAP strežniki komunicirajo direktno z L2CAP slojem v Bluetooth skladu, zato je

v ProtocolDescriptionList na zadnjem mestu, ki opisuje sekvenco nivojev protokolov preko katerih je treba, če želimo priti do strežniške aplikacije. Parameter PSM je potreben, ker je L2CAP razdeljevalni nivo, tako da lahko več aplikacij interaktira z L2CAP nivojem na strežniški napravi.

JABWT shrani v storitveni zapis vse obvezne storitvene atribute storitvenega zapisa za L2CAP, torej tvori minimalen, ampak zadosten storitveni zapis.

Tudi za OBEX se ustvari malenkost drugačen storitveni zapis kot pri SPP, in sicer ne vsebuje SerialPort in v ProtocolDescriptionList vsebuje storitveni zapis OBEX, kar pomeni da aplikacije komunicirajo direktno z OBEX nivojem v Bluetooth skladu.

Dodajanje storitvenih zapisov v SDDB

Ko se ustvari storitveni zapis, ga odjemalci še vedno ne vidijo. Odjemalci se na strežniško storitev lahko povežejo šele po klicu `acceptAndOpen()`. Zato JABWT doda storitev v SDDB le, ko se prvič kliče `acceptAndOpen()`. Ko je shranjen v bazo, lahko odjemalci najdejo storitev in se povežejo nanjo. Če dodajanje storitve v bazo ne uspe, pride do `ServiceRegistrationException`. Na enem obveščevalnem objektu je lahko odprtih več povezav različnih odjemalcev. Če nižjeležeči Bluetooth sistem tega ne podpira, vrne napako `BluetoothStateException`. Ko aplikacija ni več sposobna sprejeti zunanjih povezav jo JABWT odmakne iz SDDB ali onemogoči.

Uporaba API funkcij

Vse naprave, ki vedo naslov strežnikove naprave, lahko dostopajo do storitev, tudi če je način vidnosti nastavljen na nevidno. Iskanje storitev je neblokirajoča zahteva. Ko so storitve najdene, so podane funkciji, ki sproži dogodek. Na koncu iskanja se sproži dogodek, ki opozarja na to, da se je iskanje zaključilo. Veliko naprav podpira več iskanj storitev sočasno. Število možnih sočasnih iskanj storitev lahko dobimo iz lastnosti naprave "bluetooth.sd.trans.max".

Iskanje storitev se začne s klicem funkcije `DiscoveryAgent.searchServices()`, kateri podamo kot argumente spisek atributov, ki jih rabimo. Če mu podamo `null`, nam vrne privzete vrednosti. Drugi argument je spisek UUID-jev ki jih iščemo, tretji argument je oddaljena naprava, po kateri iščemo, zadnji pa je objekt `DiscoveryListener`, ki je obveščen, ko je najdena kakšna storitev. Kliče se `serviceDiscovered()`

metoda, ki dobi podan storitveni zapis in transakcijski ID.

Ko se iskanje storitev zaključi, se kliče metoda `serviceSearchCompleted()`. Podano dobi kodo razloga zaključitve procesa. Možne kode so SERVICE_SEARCH_COMPLETED(normalna zaključitev procesa in vsaj ena storitev najdena), SERVICE_SEARCH_TERMINATED(iskanje storitev je bilo prekinjeno s klicem na `cancelServiceSearch()`), SERVICE_SEARCH_ERROR(med iskanjem storitev je prišlo do napake), SERVICE_SEARCH_NO_RECORDS(nobena storitev ni bila najdena), SERVICE_SEARCH_DEVICE_NOT_REACHABLE(do oddaljene naprave ne moremo dostopati). Iskanje storitev lahko tudi prekinemo s klicem funkcije `cancelServiceSearch()`.

Ko odjemalec dobi storitveni zapis dane storitve, pokliče `getConnectionURL()`, ki mu vrne naslov storitve in nato se poveže na storitev. Vrednosti atributov lahko dobimo s klicem funkcije `getAttributeValue()`.

Pri ustvarjanju nove aplikacije je potrebno dodeliti unikaten UUID. Za generacijo na operacijskih sistemih Windows obstaja orodje Guidgen.txt, na Linux pa obstaja funkcija uuidgen.

2.3.7 Push registry

Večina mobilnih telefonov omogoča delovanje le ene aplikacije naenkrat. Velikokrat pa bi bilo potrebno, da bi v ozadju potekale aplikacije, ki bi enostavno samo čakale na zahteve po povezavi iz zunanjosti. Potrebne bi bile storitve, ki bi se vključile ob zahtevi po povezavi iz zunanjosti. Ta problem je bil rešen z Push Registry.

Push Registry omogoča aplikacijam, da registrirajo sprejemanje povezav tudi takrat, ko se ne izvajajo. Te obveznosti opravlja JAM(Java aplication manager- Java program za upravljanje z aplikacijami). JAM skrbi tudi za namestitev, odmestitev, pričetek, premor in zaustavitev Java MIDlet-ov na Java ME napravah.

Ko aplikacija prijavi zahtevo za Push Registry, pomeni da se želi zagnati, ko pride do zahteve po povezavi s karateristikami definiranimi v zahtevi. Ko Push Registry sprejme zahtevo po povezavi s strani oddaljene naprave, obvesti JAM. JAM zažene MIDlet, ki je specifično zahtevo prijavil. Ko se MIDlet zažene, lahko bere in piše preko povezave z oddaljeno napravo. Ko obdela zahtevo, se midlet konča in čaka da ga JAM zažene ob ponovni zahtevi po povezavi od oddaljene naprave.

Za registracijo MIDleta v Push Registy so potrebni podatki: povezovalna beseda, ki opisuje storitev, ki je na voljo, ime MIDleta, ki bo prevzel povezave

na storitev in spisek naprav, katerim je dovoljena ali ni dovoljena uporaba storitve. V Push Registry lahko prijavimo storitev ob namestitvi (definiramo spremenljivko MIDlet-push-1 v datoteki .jad) ali ob zagonu aplikacije(s klicem funkcije `registerConnection()`).

Ko se povezava registrira v Push Registry, se kreira tudi storitveni zapis na bazi povezovalne besede. Push Registry shrani storitveni zapis v SDDB.

2.4 Bluecove

Kot smo že prej povedali, JSR-197(JABWT za Java SE) še ni bil razvit. Zato smo morali najti neko drugo knjižnico, ki bi podpirala funkcije Bluetooth v okolju Java SE. Na izbiro je bilo več knjižnic, kot so Bluecove, Avetana, BlueSock... Odločili smo se za Bluecove, ker je odprtokodna. BlueCove je JSR-82 implementacija za okolje Java SE , ki trenutno deluje z Mac OS X, WIDCOMM, BlueSoleil in Microsoft Bluetooth skladom iz Windows XP SP2 in novejšimi. Do sedaj ne implementira še vseh API funkcij iz JSR-82. BlueCove deluje na vseh Java virtualnih strojih od verzije 1.1 dalje, na Windows Mobile, Windows XP, Windows Vista, Mac OS X. Kasneje so podprli še linuxov sklad BlueZ.

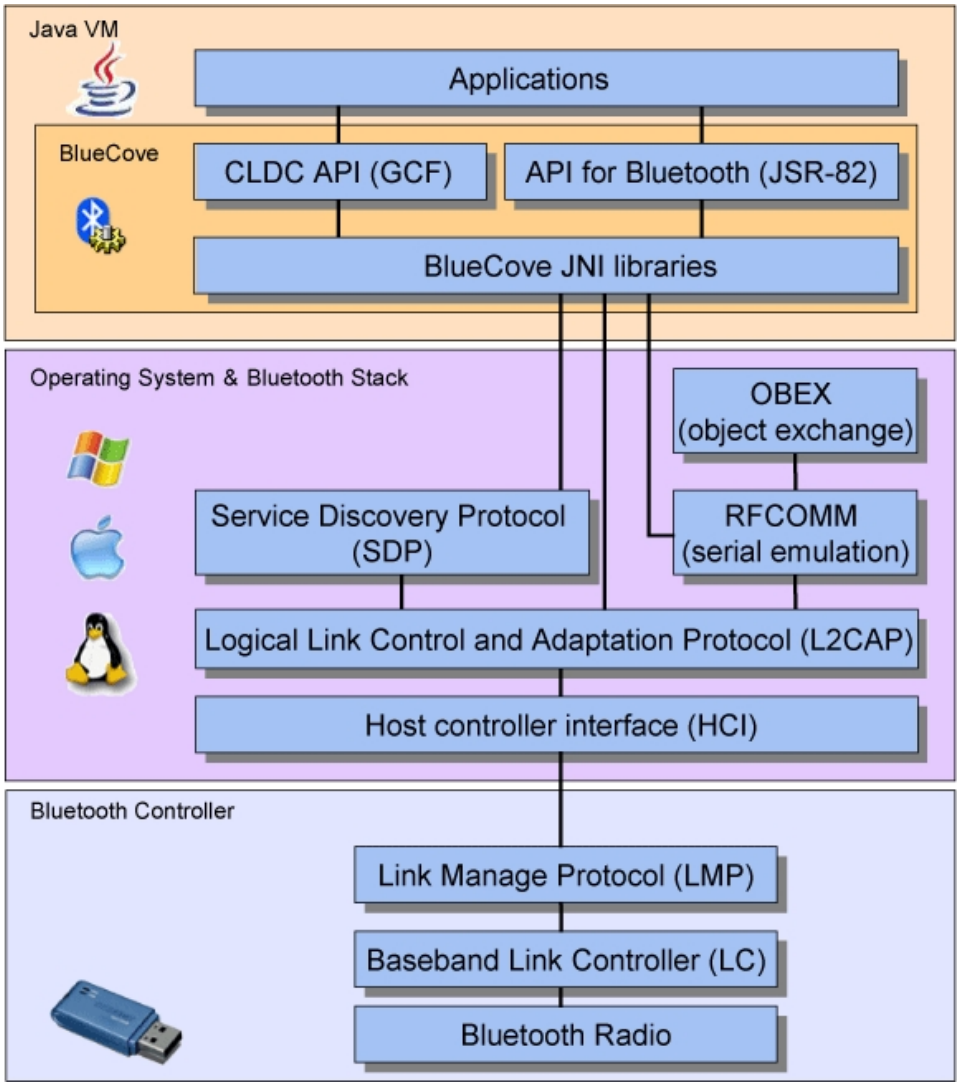
2.4.1 Omejitve

BlueCove ima nekatere omejitve. L2CAP podpira le na Windows WIDCOMM skladu, Linux BlueZ in Mac OS X sklad. Ker podpira Microsoft Bluetooth sklad le RFCOMM, podpira na tem skladu le ta protokol tudi BlueCove. Podprobnejše opisane omejitve v zvezi s skladi in Bluecove lahko najdete na spletni strani http://code.google.com/p/bluecove/wiki/stacks.

2.4.2 Konfiguracija ob zagonu

Z Java sistemskimi nastavitvami lahko izbiramo kateri Bluetooth sklad bomo uporabljali in inicializirali. Sistemsko lastnost “bluecove.stack“ določimo kot “widcomm“, “bluesoleil“ ali “winsock“. Samodejno je izbran sklad winsock, če obstaja.

Lahko določamo tudi lastnost “bluecove.stack.first“ za optimizacijo detekcije sklada. Če je -Dbluecove.stack.first=widcomm, potem je naložen najprej widcom sklad, če pa ga ni, bo bluecove prešel na “winsock“. Če je detektiranih več skladov so izbrani po sledečem zaporedju: “winsock“, “widcomm“, “bluesoleil“.



Slika 2.9: Diagram bluecove

Lahko določamo naslednje sistemske lastnosti:

- `bluecove.connect.timeout` - Potreben čas v milisekundah za vzpostavitev povezave RFCOMM ali L2CAP pred napako BluetoothConnectionException. Prednastavljeno je na 2 minuti. Samo WIDCOMM in OS X.
- `bluecove.obex.timeout` - Potreben čas v milisekundah v katerem aplikacija poskuša uspešno prenesti paket pred sprožitvijo InterruptedIOException napake. Prednastavljeno je na 2 minuti. Za vključitev te opcije je potrebno uporabiti
  `javax.microedition.io.Connector.open(String,int, true);`
- `bluecove.obex.mtu` - Možno je povečanje prenosne hitrosti s spremembo mtu na večjo vrednost. Prednastavljeno je 1024.
- `bluecove.bluez.class`
- `bluecove.inquiry.duration` - povpraševanje naprave v sekundah. Prednastavljeno je na 11s.
- `bluecove.inquiry.report_asap` - Če nastavimo na `true` za klic DiscoveryListener.deviceDiscovered brez čakanja na posodobitev razreda storitev.
- `bluecove.jsr82.psm_minimum_off` - omogoča PSM vrednosti manjše od 0x1001.
- `bluecove.connect.unreachable_retry` - poskusi ponovno povezati na MS sklad ob napaki
- `bluecove.debug.stdout` - onemogoči System.out.println debug
- `bluecove.debug.log4j` - onemogoči log redirekcijo na log4j

2.5 Zgradba Bluetooth aplikacij

Funkcionalnost JABWT se deli na tri glavne kategorije: Odkrivanje, komunikacija in upravljanje z napravami. Bluetooth aplikacije uporabljajo naslednje pakete:

1. javax.microedition.midlet.MIDlet,

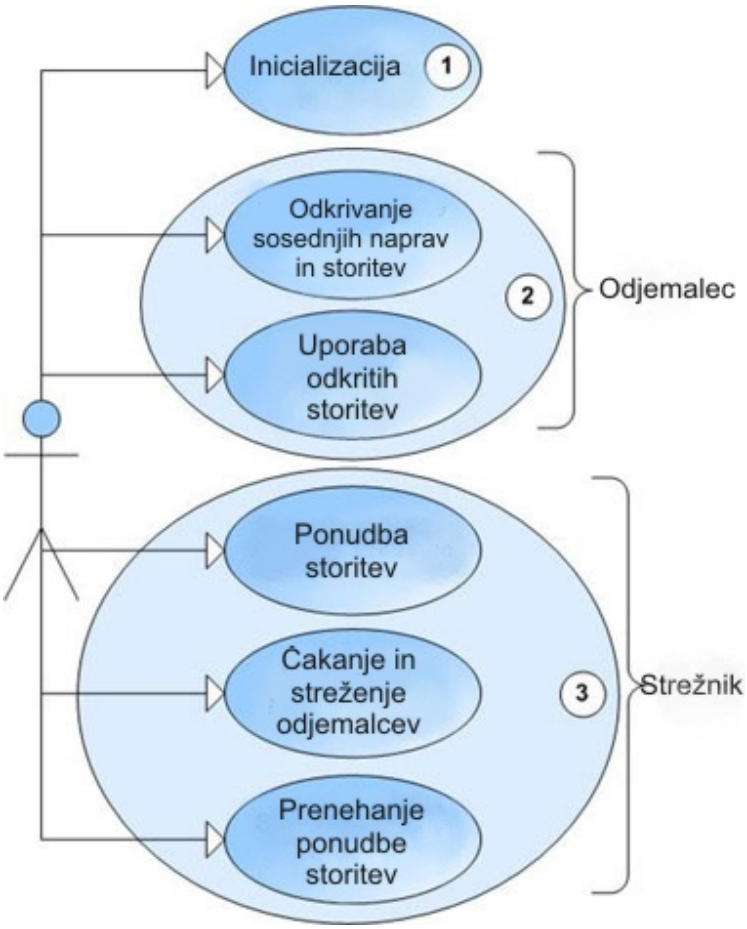
2. javax.bluetooth.DiscoveryAgent,
3. javax.bluetooth.DiscoveryListener,
4. javax.bluetooth.LocalDevice,
5. javax.bluetooth.RemoteDevice,
6. javax.bluetooth.ServiceRecord,
7. javax.bluetooth.L2CAPConnection,
8. javax.bluetooth.L2CAPConnectionNotifier,
9. javax.microedition.io.StreamConnection,
10. javax.microedition.io.StreamConnectionNotifier.

Bluetooth aplikacija je lahko strežnik ali odjemalec, lahko pa se obnaša kot točka peer-to-peer z obnašanjem obeh tipov(strežnik ali odjemalec). Slika 2.10 prikazuje uporabo Bluetooth aplikacij.

Bluetooth aplikacija poteka tako:

1. Inicializacija: vse Bluetooth aplikacije morajo najprej inicializirati Bluetooth sklad.
2. Odjemalec: Uporablja storitve oddaljene naprave. Najprej s SDP poišče naprave v bližini, nato pa storitve, ki jih te naprave nudijo.
3. Strežnik: Omogoča storitve drugim napravam. Najprej storitve registrira v Service Discovery Database(SDDB), nato pa čaka na prihajajoče povezave, sprejema in streže odjemalce. Ko storitev ni več potrebna, jo aplikacija odstrani iz SDDB.

Bluetooth sklad pri tem omogoča skladišče za storitvene zapise , ki dopušča dodajanje, popravljanje in brisanje svojih lastnih storitev ter povezave z aplikacijami oddaljenih naprav. Za SDP pa omogoča iskanje storitev shranjenih v SDDB.



Slika 2.10: Shema strežnik-odjemalec.

Poglavje 3

Predstavitev programa

V tem poglavju bomo predstavili praktični del diplomskega dela, to je aplikacija, ki temelji na tehnologijah, opisanih v prejšnjem poglavju.

3.1 Uporabljena orodja za razvoj aplikacije

Aplikacija je sestavljena iz dveh delov. Prvi del je bil implementiran v okolju Java ME in je namenjen mobilnim telefonom. Drugi del pa je bil sprogramiran v okolju Java SE in se uporablja na računalnikih. Oba dela smo razvili s pomočjo orodja Eclipse. Za upravljanje z Bluetooth tehnologijo je bilo potrebno dodati še knjižnico v kateri so implementirane funkcije za uporabo komunikacije Bluetooth(Bluecove in J2ME wireless toolkit, ki vključuje JSR-82). Testiranje aplikacije med razvojem in ob zaključku je na napravi lahko zelo težavno, saj naprave nimajo orodij za razhroščevanje. Pomagamo si lahko le z izpisi na zaslon. Poteku aplikacije je na tak način zelo težko slediti. Zato smo za razvoj aplikacije za mobilne telefone uporabili simulator naprave Sun Java Wireless Toolkit. Sun Java Wireless Toolkit emulira mobilno napravo, tako da lahko testiramo MIDlet-e preden jih dejansko naložimo na telefone. Ko je aplikacija pravilno delovala v emulatorju, smo jo preizkusili še na mobilnih napravah. To smo storili tako, da smo najprej naredili datoteko tipa jar in jo naložili na mobilno napravo.

Na žalost je tudi z uporabo predhodnega testiranja na emulatorju razvoj aplikacije za okolje Java ME zelo težaven, saj se mobilni telefoni različnih tipov in proizvajalcev med seboj zelo razlikujejo v delovanju in implementacijah Bluetooth protokolov. Tako delovanje aplikacije v emulatorju še ne zagotavlja delovanja na mobilnih telefonih.

Razvoj za osebne računalnike pa je potekal podobno kot razvoj ostalih Java

aplikacij, le da smo vključili dodatno knjižnico za upravljanje z tehnologijo Bluetooth - Bluecove. Kot smo omenili v razdelku 2.4 je takih knjižnic več, vendar smo se odločili za Bluecove, ker je odprtokodna.

3.2 Opis funkcionalnosti aplikacije

3.2.1 Pogoji za delovanje

Za uporabo aplikacije je potreben mobilni telefon, osebni računalnik ali katerakoli naprava, ki podpira Java ME ali Java SE aplikacije. Naprava mora vsebovati Bluetooth oddajnik in programsko opremo, ki upravlja z njim (npr.WIDCOMM, WINSOCK,...). Pogoji za delovanje aplikacije na napravah so vključen Bluetooth oddajnik in vidnost naprave do ostalih Bluetooth naprav.

3.2.2 Opis aplikacije

Slika 3.1: Primer izgleda aplikacije.

Namen aplikacije je ustvariti klepetalnico v katero se lahko vključujejo mobilni telefoni in računalniki, ki komunicirajo preko Bluetooth povezave. Vsaka naprava, ki se želi vključiti v klepetalnico mora imeti vklopljen Bluetooth oddajnik, nato poišče vse že odprte klepetalnice v dosegu svojega Bluetooth oddajnika. Ko se iskanje konča, se uporabnik lahko vključi v eno izmed najdenih klepetalnic ali ustvari svojo klepetalnico. Kdor odpre novo klepetalnico, avtomatično postane njen lastnik in ima zato posebne funkcije. Določa, kdo

lahko vstopi v klepetalnico in kdo ne. Uporabniki klepetalnice lahko nato pišejo vsem uporabnikom in berejo izjave vseh ostalih uporabnikov. Za uporabniško ime se uporablja ime Bluetooth naprave. Beležijo se tudi ostali dogodki, kot je vključitev in izključitev novih članov ter zaprtje klepetalnice. Za preglednost nad člani klepetalnice pa smo dodali še dodatno funkcijo, ki izpiše vse trenutne člane klepetalnice. Primer izgleda klepetalnice je prikazan na sliki 3.1.

3.2.3 Tokovi dogodkov

Slika 3.2 prikazuje potek aplikacije.

Ko uporabnik zažene aplikacijo, se takoj prične iskanje sosednjih naprav(slika 3.2 prvi korak), na katerih že poteka klepetalnica(v lasti jo ima lastnik naprave). Ko se iskanje konča, ima uporabnik na voljo(slika 3.2 drugi korak):

1. odpreti lastno klepetalnico,

2. izbrati eno od že obstoječih klepetalnic v okolici in vstopiti vanjo.

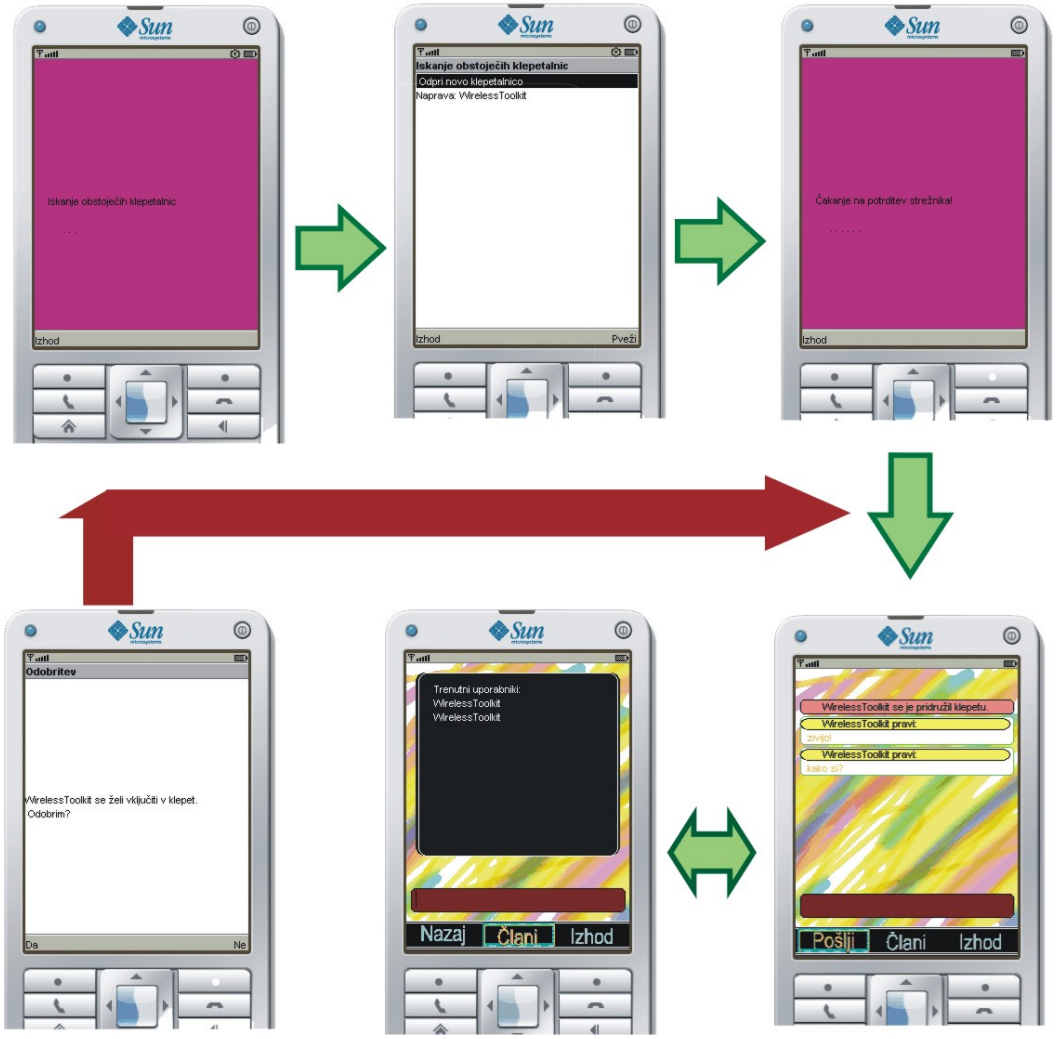
V primeru, da se odloči za prvo možnost se na mobilnih telefonih izpiše opozorilo za odpiranje strežniške povezave. Uporabnik mora dovoliti take povezave, sicer klepetalnica ne more sprejemati novih članov. Nato lahko uporabnik sprejema ali zavrača nove člane ter si dopisuje z že sprejetimi člani klepetalnice.

V primeru da se odloči za drugo možnost, se na mobilnih telefonih izpiše opozorilo za odpiranje odjemalčeve povezave. Uporabnik mora dovoliti take povezave, sicer se aplikacija konča, saj ne more pravilno delovati dalje. Nato uporabnik čaka na potrditev s strani strežnika(slika 3.2 tretji korak). Ko mu strežnik odobri vstop v klepetalnico, se mu prikaže klepetalnica in lahko si začne pisati z ostalimi uporabniki(slika 3.2 četrti korak).

V obeh primerih pa ima uporabnik na voljo še pregled nad vsemi uporabniki, ki so trenutno v klepetalnici. Do spiska uporabnikov pride tako, da izbere gumb “Člani“ in pojavi se izpis vseh trenutnih članov klepetalnice(slika 3.2 peti korak).

Nova sporočila uporabnik pošilja tako, da najprej vpiše tekst v polje in nato pošlje s klikom na gumb “Pošlji“. Poslano sporočilo se izpiše na zaslon pošiljatelju in vsem ostalim članom klepetalnice v obliki, kot je prikazana na sliki 3.2 v prvem koraku. Izpiše se ime pošiljatelja, spodaj pa sporočilo, ki ga je pošiljatelj poslal.

Za boljšo preglednost nad trenutnimi člani v klepetalnici se na zaslon izpisujejo tudi izredni dogodki, kot so vključitev novega člana v klepetalnico ali za-



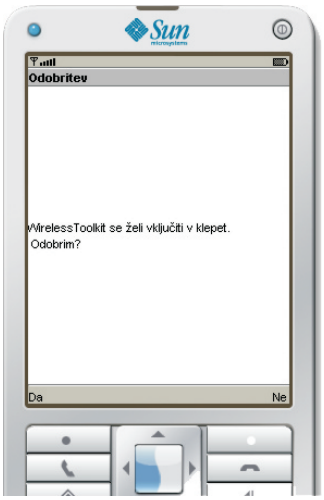
Slika 3.2: Shema poteka aplikacije za Bluetooth klepetalnico. Potek prestavlja korake uporabnika, ki se vključi v že obstoječo klepetalnico. Zelena puščica predstavlja prehode med stanji v aplikaciji. Rdeča puščica pa predstavlja zunanji dogodek(odobritev uključitve v klepetalnico) s strani lastnika klepetalnice, v katero se uporabnik želi vključiti.

pustitev klepetalnice s strani članov. Ti dogodki so v oblačkih obarvanih z rdečo barvo tako, da se ločijo od sporočil, ki jih pošiljajo uporabniki.

V primeru, da klepetalnico zapusti njen lastnik, se klepetalnica zapre in aplikacije vseh članov se zaključijo z opozorilom.

3.2.4 Varnost

Java API funkcije omogočajo tri tipe varnosti - avtentifikacija, enkripcija in avtorizacija. Uporabili smo vse tri. Varnost je pri brezžičnih komunikacijah zelo pomembna, saj bi drugače lahko prišlo do prisluškovanja s strani neželjenih naprav. Avtorizacija je za našo aplikacijo pomembna, saj mora tako lastnik klepetalnice vedno potrditi ali zavrniti vsakega novega uporabnika, ki se želi vključiti v klepetalnico. Le tako ima lahko popolni nadzor nad vsemi člani klepetalnice. Primer avtorizacije je na sliki 3.3.



Slika 3.3: Primer uporabe avtorizacije.

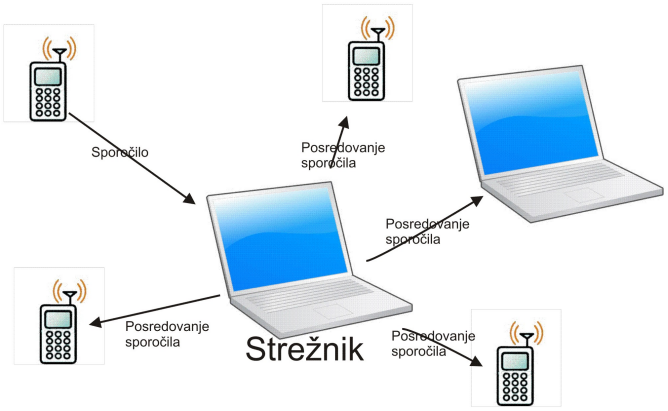
3.3 Potek razvoja aplikacije

V tem poglavju bomo opisali potek razvoja aplikacije. Preden smo se lotili konkretnega programiranja aplikacije, smo določili koncept le-te.

3.3.1 Koncept aplikacije

Kot je razvidno iz slike 3.4, smo si kot center sistema zamislili strežnik, ki se lahko nahaja na mobilnih telefonih ali osebnih računalnikih, nanj pa se povezujejo pripadajoči odjemalci - prav tako osebni računalniki ali mobilni telefoni. Strežnik je v aplikaciji lastnik klepetalnice, odjemalci pa so ostali člani klepetalnice. Vsako sporočilo, ki ga pošlje odjemalec (ali streznik), se preko strežnika prenese na vse ostale odjemalce. Strežnik posreduje odjemalcem tudi vse ostale podatke. V primeru ko se v klepetalnico vključi ali izključi član, strežnik to sporoči vsem odjemalcem. Spisek vseh odjemalcev ima shranjen strežnik. Če želi odjemalec pregled nad vsemi člani klepetalnice, mora poslati zahtevo strežniku in ta mu vrne spisek imen.

Obe aplikaciji(strežnik ali odjemalec) sta za uporabnike na videz enaki.



Slika 3.4: Shema odjemalci-strežnik.

3.3.2 Koraki pri zvedbi aplikacije

Postavitev okolja

Kot že prej omenjeno, smo aplikacijo razvijali s pomočjo orodja Eclipse. Ker se mobilni telefoni razlikujejo od osebnih računalnikov po zmogljivosti, je bilo

zanje ustvarjeno posebej prilagojeno, manj obsežno okolje - Java ME. Za to okolje pa se razvijajo posebne Java aplikacije, ki jim pravimo MIDlet. V projekt je bilo potrebno vključiti še dodatno knjižnico Java Sun Wireless Toolkit for CLDC, ki je potrebna za brezžične mobilne aplikacije. Vsebuje pa tudi emulator, na katerem lahko testiramo program.

Del aplikacije, namenjen osebnim računalnikom, smo razvijali s pomočjo orodja Eclipse. Razvoj je potekal na operacijskem sistemu Windows XP. Potrebovali smo druge knjižnice, prilagojene za okolje Java SE. Na voljo smo imeli več knjižnic, a smo izbrali odprtokodno Bluecove. Med Bluetooth skladi pa smo izbrali WIDCOMM, ker je med možnimi najbolj zanesljiv.

Izbira Bluetooth protokola

Postavitvi okolja za razvoj aplikacije je sledila izbira Bluetooth protokolov. Ker se v naši aplikaciji pošiljajo le navadne besede tipa `String`, smo se prvotno odločili za komunikacijo preko protokola RFCOMM. Ta je v primerjavi z L2CAP veliko lažji za uporabo, saj je višjenivojski in sam skrbi za preliv čakalnih vrst. OBEX smo takoj izločili iz možnih protokolov za naš primer, saj je namenjen pošiljanju objektov in bi bil za našo aplikacijo neprimeren. Z RFCOMM smo med razvojem prišli do večjih problemov, saj na mobilnih telefonih protokol ni deloval, kot bi moral, in kot je deloval na emulatorjih. Zato smo se odločili za nižjenivojski protokol L2CAP. Seveda smo potrebovali še protokol SDP za iskanje obstoječih strežnikov.

Na osebnih računalnikih smo se odločili za Bluetooth sklad WIDCOMM, saj podpira največ funkcij.

Razvoj strežnika

Strežnik je naprava, na kateri je odprta klepetalnica. Uporabnik odpre novo klepetalnico tako, da po iskanju že odprtih klepetanic izbere odprtje nove klepetalnice. Tako se poda stavku `Connector.open()` povezovalna beseda za strežnik. V našem primeru smo odprli strežniško aplikacijo, kot je prikazano na sliki 3.5.

```
L2CAPConnectionNotifier notifier = (
        L2CAPConnectionNotifier)Connector.open("btl2cap://localhost:" +
        "86b4d249fb8844d6a756ec265dd1f6a3;authorize=true;encrypt=true;"+
        "authenticate=true;name=BLUETOOTH_SERVER");
```

Slika 3.5: Stavek za odprtje strežniške povezave.

S tem, ko odpremo povezavo za strežnik, se storiteveni zapis shrani v SDDB bazo tako, da lahko ostale naprave zaznajo in uporabljajo to storitev.

Strežnik opravlja vse glavne funkcije. Skrbi za povezanost vseh naprav, ki so udeležene v klepetalnici. Vsaka naprava, ki nekaj napiše v klepetalnico, pošlje besedilo do strežnika(slika 3.6), strežnik besedilo izpiše na uporabnikovem zaslonu in pošlje besedilo še ostalim članom tako, da imajo na koncu vse naprave izpisan enak pogovor na zaslonu.

```
conn.send(highlightedMenu.currentString.getBytes());
```

Slika 3.6: Pošiljanje sporočila.

Strežnik pri stavku `acceptAndOpen()` čaka na zunanje povezave. Za vsakega novega člana, ki se vključi v klepetalnico, se odpre nova nit, ki skrbi za prejemanje sporočil. Ko nit sprejme sporočilo od pošiljatelja, ga pošlje vsem ostalim članom(slika 3.7). Spisek članov je shranjen v strukturi tipa `Vektor` na strežniku.

```
public void sendToAllClients(String out,String nameOfClient)
{

    for(int i = 0; i <listClientov.size(); i++)
    {
        if(listClientov.elementAt(i)!=conn)
        //  odjemalcu ki je poslal sporočilo ne posredujemo sporočila
        {
            try
            {
                ((L2CAPConnection)listClientov.elementAt(i)).send(nameOfClient.getBytes());
                ((L2CAPConnection)listClientov.elementAt(i)).send(out.getBytes());

            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

Slika 3.7: Pošiljanje sporočila vsem ostalim odjemalcem.

Odjemalec in strežnik si pošiljata nize, ki se začnejo z enim izmed definiranih začetkov. Te besede pomenijo tip zahteve. Možne besede, ki jih lahko pošlje odjemalec ali strežnik so :

;%left;% Ta ukaz pošlje strežnik vsem članom, ko se iz klepetalnice odjavi uporabnik.

;%us;% Ta ukaz pošlje strežnik odjemalcu, ko mu odgovarja na zahtevo po pridobitvi vseh članov klepetalnice.

;%join;% Ta ukaz pošlje strežnik vsem članom, ko se v klepetalnico prijavi nov uporabnik.

;%lefts;% Ta ukaz pošlje strežnik vsem članom, ko sam zapre klepetalnico.

;%users;% Ta ukaz pošlje odjemalec strežniku, ko želi dobiti spisek vseh članov klepetalnice.

Posebni nizi imajo za delom, ki pomeni tip zahteve še podatke, ki jih rabi aplikacija za izvršitev zahteve. Npr. z ukazom, da je v klepetalnico vstopil nov član, se pošlje še ime novega člana.

Registracija storitve

Registracija storitve je potrebna samo pri aplikacijah, ki so strežnik. Storiteveni zapis se avtomatično shrani v SDDB, ko strežniška aplikacija prvič pride do stavka `acceptAndOpen()` (slika 3.8). Ko je storitev registrirana, jo lahko zaznajo sosednje naprave s protokolom za iskanje storitev na sosednjih napravah (SDP) in se nato povežejo nanjo in vstopijo v klepetalnico.

```
while(true)
{
    if(BtChatMidlet.koncajNit == 1)
    // ob izhodu aplikacije se čakanje na odjemalce prekine
    {
        break;
    }
     conn = notifier.acceptAndOpen();

     //odpremo nit, ki streže novemu odjemalcu
     WaitingForConfirm wc = new WaitingForConfirm(dis, conn,listClientov,
             highlightedMenu,listRemoteDeviceov,localDevice,chServer);
}
```

Slika 3.8: Stavek acceptAndOpen() čaka na zunanje povezave.

Iskanje sosednjih naprav in storitev na le-teh

Ko se aplikacija zažene, prične takoj iskati odprte klepetalnice v dosegu svojega Bluetooth oddajnika. To stori s protokolom iskanja bližnjih naprav(SDP).

Najprej poišče vse naprave, ki so v bližini. Iskanje začnemo s stavkom `discoverAgent.startInquiry( DiscoveryAgent.GIAC,this );`. Ko protokol najde napravo, sproži dogodek, ki ga ujame funkcija na sliki 3.9. Za iskanje smo nastavili pogoj, da iščemo le naprave, ki so osebni računalniki ali mobilni telefoni. Tako smo skrajšali čas iskanja s takojšnjim izločanjem neustreznih naprav.

```
public void deviceDiscovered(RemoteDevice remoteDev, DeviceClass devClass)
{
    try
    {
        if( devClass.getMajorDeviceClass() == 0x0200 ||
                devClass.getMajorDeviceClass() == 0x0100)
        //samo ce je najdena naprava mobilni telefon ali osebni računalnik
        {
            remoteDevices.addElement(remoteDev);
        }
    }
   catch ( Exception e )
   {
        listDev.append("Error 3",null);
   }
}
```

Slika 3.9: Funkcija, ki ujame dogodek in se sproži ob novi najdeni napravi.

Ko aplikacija najde novo napravo, jo shrani v tabelo najdenih naprav.

Ko se konča iskanje sosednjih naprav, se kliče funkcija `inquircCompleted()`. Tako protokol SDP sporoči aplikaciji da se je iskanje sosednjih naprav zaključilo. Ko se iskanje zaključi, lahko začnemo z iskanjem storitev na napravah. Iščemo točno določeno storitev, katere UUID podamo kot argument(slika 3.10).

Za vsako napravo, na kateri najdemo storitev z UUID-jem u se sproži dogodek, ki ga ujame funkcija `servicesDiscovered()`(slika 3.11). Če je storitev aplikacija Bluetooth klepetalnica tipa strežnik, jo shranimo v tabelo, ki jo kasneje aplikacija ponudi med odprtimi klepetalnicami.

Po koncu iskanja se uporabniku izpišejo vse klepetalnice in možnost odprtja nove klepetalnice.

Razvoj odjemalca

Odjemalec je naprava, ki se vključuje v že odprto klepetalnico na oddaljeni napravi. Uporabnik izbere eno izmed že odprtih klepetanic, ki mu jih vrne

```
public int starServiceSearch(RemoteDevice remoteDev)
{
    try {
        int attrbs[] = { 0x0100,0x0200 };
        UUID[] u = new UUID[]{new UUID( "86b4d249fb8844d6a756ec265dd1f6a3", false )};
        discoverAgent.searchServices(attrbs, u, remoteDev, this);
    }
    catch (BluetoothStateException e) {
        listDev.append("Error 99",null);
        e.printStackTrace();
    }
    return 0;
}
```

Slika 3.10: Funkcija, ki začne iskati storitv z UUID u na oddaljeni napravi remoteDev.

```
public void servicesDiscovered(int transID, ServiceRecord[] serviceRecords)
{
    for(int i = 0; i < serviceRecords.length; i++)
    {
       if(((String)serviceRecords[i].getAttributeValue(0x0100).getValue())
                .equalsIgnoreCase("BLUETOOTH_SERVER")||
                ((String)serviceRecords[i].getAttributeValue(0x0200).getValue())
                .equalsIgnoreCase("BLUETOOTH_SERVER"))
       {
            try
            {
                listNaslovov.addElement(serviceRecords[i].getHostDevice());
                listConnUrl.addElement( serviceRecords[i].getConnectionURL(0, false));
                listDev.append( "Naprava: " +
                        serviceRecords[i].getHostDevice().getFriendlyName( false ),
                        null);
            }
            catch (Exception e)
            {
                listDev.append("Error 3",null);
                e.printStackTrace();
            }
       }
    }
}
```

Slika 3.11: Funkcija ki ujame dogodek, ki se sproži ob najdenih storitvah na oddaljeni napravi.

protokol SDP. Tako se poda stavku `Connector.open()` povezovalna beseda za odjemalca, ki jo dobimo iz storitvenega zapisa z metodo `getConnectionURL()`. Naslov storitve nam vrne protokol SDP v funkciji `serviceDiscovered()`, katera ima kot parameter storitveni zapis, ki vsebuje tudi naslov storitve.

Odjemalec lahko komunicira z ostalimi člani samo preko strežnika. Ko hoče poslati sporočilo vsem članom, ga v resnici najprej pošlje strežniku, ta pa ga posreduje naprej ostalim. Pri prejemanju sporočil pa gre proces v obratno smer. Ostali člani enako pošljejo sporočilo strežniku, ta pa ga pošlje ostalim in uporabniku.

MTU velikost

MTU parametra se pri razvoju aplikacije nismo dotikali. Pustili smo privzeto velikost. Tako smo se izognili morebitnim težavam z usklajenostjo velikosti in smo zadostili vsem pravilom MTU navedenim v poglavju o L2CAP protokolu.

Grafika

Ko je aplikacija že pravilno delovala, smo ji dodali še lepši izgled. Da bi se čim manj razlikovala koda med aplikacijo, namenjeno mobilnim telefonom in aplikacijo, namenjeno osebnim računalnikom smo za grafiko uporabili razred `java.awt.Canvas`. Za osebne računalnike smo naredili navadno okno tipa `JFrame`, za mobilne telefone pa obstaja poseben razred `Display`, ki nam omogoča dostop do zaslona mobilnega telefona. Podrobnega opisa razvoja grafike ne bomo podali, saj to ni bil namen diplomske naloge.

Koraki pri zvedbi aplikacije za osebne računalnike

Za osebne računalnike je ostala aplikacija po funkcionalnosti enaka. Potrebno je bilo spremeniti MIDlet v navadno java aplikacijo za okolje Java SE. Tako je aplikacija za osebne računalnike izhajala iz osnovnega razreda `JFrame`. Namesto JABWT pa smo morali uporabiti posebno knjižnico, namenjeno za okolje Java SE, to je Bluecove.

3.4 Razširitve

Aplikacija ima do sedaj le osnovne, nujno potrebne funkcije. Da bi bila bolj uporabna, bi lahko v prihodnosti dodali še nove funkcije:

- Možnost pošiljanja datotek vsem članom klepetalnice(npr. slike, melodije, elektronske vizitke,...).
- Lastnik klepetalnice bi moral imeti možnost izključitve člana iz klepetalnice.
- Ker je Bluetooth kratkega dosega, bi bilo dobro razširiti aplikacijo tako, da bi sporočila posredovali tudi odjemalci drugim odjemalcem. Gradilo bi se omrežje strežnikov. Tako bi lahko povečali največje število članov na več kot 7 in razdalja članov bi se z vsakim članom povečala za njegov radij.
- Aplikacija bi morala samodejno vključiti Bluetooth, vendar zaenkrat ta funkcija še ni podprta.
- Aplikacijo bi lahko prilagodili za različne operacijske sisteme in mobilne telefone.

Poglavje 4

Zaključek

V diplomskem delu smo opisali razvoj aplikacije Bluetooth klepetalnice za okolje Java ME in Java SE. V poglavju o tehnologijah smo predstavili potrebno znanje za razumevanje razvoja aplikacije.

V poglavju o praktičnem delu pa smo opisali postopek, kako smo prišli do rešitve in nekatere probleme do katerih smo prišli med razvojem.

Prišli smo do zaključka, da je razvoj za mobilne telefone zalo težaven, saj se implementacije različnih protokolov razlikujejo med proizvajalci in tipi mobilnih telefonov. Potrebno je razvijati aplikacijo za točno določen tip telefona. Opraviti je treba testiranja, ki so tudi zelo težavna, saj na mobilnih telefonih ni razhroščevalnikov in je sledenje programu oteženo.

Z mobilnimi telefoni smo imeli probleme tudi zaradi premajhnih pomnilnikov. Tako smo prišli do ugotovitve, da mobilni telefoni niso primerni za vlogo strežnika, saj ne morejo shranjevati velike količine podatkov in pri več povezavah delujejo zelo počasi. Tako postane klepetalnica neuporabna. Za učinkovito delovanje mora biti strežnik osebni računalnik.

Bluetooth je zaenkrat še v razvoju. API funkcije za okolje Java SE še niso popolne. Za uporabo določenih funkcij mora razvijalec poskrbeti sam. V Bluecove knjižnici obstajajo tudi hrošči, ki se še odpravljajo.

Ker pa se Bluetooth vedno bolj uveljavlja in razvijalci vse bolj pogrešajo mankajoče funkcije, verjamemo da bojo kmalu izdane tudi uradne, popolne API funkcije za okolje Java SE in takrat se bo Bluetooth uporabljal vse pogosteje.

Literatura

[1] Timothy J. Thompson, Paul J. Kline, C Bala Kumar, *Bluetooth application programming with the Java APIs essential edition*, United States of America, 2008.

[2] (2009) Bluecove. Dostopno na: http://code.google.com/p/bluecove/

[3] (2008) Bluecove JSR-82 project. Dostopno na: http://bluecove.org/

[4] (2008) Bluetooth stack - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Bluetooth_stack

[5] (2009) Using the Java APIs for Bluetooth Wireless Technology. Dostopno na: http://developers.sun.com/mobility/apis/articles/bluetoothintro/

[6] (2006) Package javax.microedition.midlet. Dostopno na: http://java.sun.com/javame/reference/apis/jsr037/javax/microedition/midlet/package-summary.html/

[7] Forum.Nokia.Com. Dostopno na:http://wiki.forum.nokia.com

[8] Wireless Application Programming with J2ME and Bluetooth. Dostopno na: http://developers.sun.com/mobility/midp/articles/bluetooth1/

[9] Bluetooth protocols - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Bluetooth_protocols

[10] Bluetooth - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Bluetooth

[11] Radio Baseband LMP HCI L2CAP RFCOMM SDP Profiles Logical Link Control and Adaptation Protocol.Dostopno na: http://www.palowireless.com/infotooth/tutorial/l2cap.asp