TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Juho Hämäläinen

Internet Relay Chat -asiakasohjelma Symbian S60 -ympäristössä

Työn ohjaaja: Lehtori Tony Torp
Työn teettäjä: Digia Oyj, valvojana diplomi-insinööri Janne T. Koskinen
Tampere 2008

TIIVISTELMÄ

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka
Juho Hämäläinen: Internet Relay Chat -asiakasohjelma Symbian S60 -ympäristössä
Tutkintotyö, 42 sivua
Työn ohjaaja: Lehtori Tony Torp
Työn teettäjä: Digia Oyj, valvojana diplomi-insinööri Janne T. Koskinen
Kesäkuu 2008
Hakusanat: Symbian OS, asiakas-palvelin-arkkitehtuuri, Internet Relay Chat, TCP/IP

Symbian OS on erittäin laajalle levinnyt mobiililaitteissa käytettävä sovellusympäristö. Symbian OS on toteutettu lähes kokonaan C++-kielellä, ja se on voimakkaasti asiakas-palvelin-mallin mukainen. Sovellukset ajetaan yleensä yhdessä säikeessä, jolloin rinnakkaisuus toteutetaan aktiiviolioiden avulla. Internet Relay Chat on tekstipohjainen TCP/IP-kerroksen päällä toimiva asynkroninen joukkokeskusteluprotokolla. Protokollan yksinkertaisen luonteen takia se on hyvä pohja yksinkertaiselle tietoverkkoa käyttävälle sovellukselle. Symbianin verkko-ohjelmointiin käytetään Socket Server -rajapintaa.

Tämän työn tavoitteena on tutustua Symbian OS -ympäristöön ja toteuttaa Internet Relay Chat -asiakasohjelma. Työ tehtiin kahdessa osassa. Ensin toteutettiin asiakasohjelma, minkä jälkeen kirjoitettiin toteutuksesta raportti.

Työ onnistui hyvin ja asetetut tavoitteet saavutettiin. Toteutus ei täysin seuraa oliomallia ja sille löytyi myös vaihtoehtoinen oliomaisempi esimerkki. Vaihtoehtoisesta toteutuksesta ei kuitenkaan välttämättä ole suurta hyötyä, sillä myös toteutunut sovellus on helpohkosti laajennettavissa.

ABSTRACT

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software Engineering
Juho Hämäläinen: Internet Relay Chat client application in Symbian S60 environment
Engineering thesis, 42 pages
Thesis supervisor: Tony Torp (lecturer)
Commissioning company: Digia Plc. Supervisor Janne T. Koskinen (MSc)
June 2008
Keywords: Symbian OS, Client-Server architecture, Internet Relay Chat, TCP/IP

Symbian OS is widely spread Software Framework that is used in mobile devices. Symbian OS is implemented almost completely in C++ language and it is strongly Client-Server oriented. Applications are usually run in single thread and multi-threading is accomplished using Active Objects. Internet Relay Chat is text based asynchronic multi-user chat protocol on TCP/IP-layer. Since the protocol is quite simple it is good base for simple network based application. In Symbian Socket Server API is used to implement network communication.

The purpose of this thesis is to get aquainted with Symbian OS Framework and implement Internet Relay Chat client application. Thesis was done in two parts. First client application was implemented and afterwards a report was written to describe the implementation.

Thesis succeeded well and set goals were met. The implementation is not fully object oriented and an alternative more object oriented example was found. Still, the alternative isn't necessarily more usefull since the used implementation is also quite easily expandable.

ALKUSANAT

Tämä on insinöörityö Tampereen ammattikorkeakoulun tietotekniikan koulutusohjelmaan. Se on tehty vuosien 2007 ja 2008 aikana Digia Oyj:lle. Työn ohjaajana on toiminut Team Manager Janne T. Koskinen ja valvojana lehtori Tony Torp. Työn tarkoituksena on oppia Symbian OS -ympäristöä ja toteuttaa Internet Relay Chat -asiakassovellus.

Tahdon kiittää työni ohjaajaa Janne T. Koskista erityisesti opastuksesta toteutuksen kanssa sekä kirjallisen osan suunnittelussa. Kiitokset työtovereilleni Matti Kosolalle ja Janne Kekille työstä käyttöliittymän toteutuksessa. Kiitos myös vaimolleni Tiina Hämäläiselle tuesta ja kärsivällisyydestä työn eri vaiheissa.

Tampereella 2.6.2008

Juho Hämäläinen

SISÄLLYS

TERMIT JA SYMBOLIT

IRC Internet Relay Chat

Berkeley Sockets (BSD Socket API) BSD Unixille kehitetty socket-rajapinta

TCP/IP TCP over IP, IP-verkon päällä toimiva yhteydellinen tietoliikenneprotokolla

IP Internet Protocol huolehtii datagrammien luomisesta ja siirtämisestä verkon lävitse

TCP Transmission Control Protocol, yhteydellinen tietoliikenneprotokolla

Socket Rajapinta prosessin ja TCP/IP-rajapinnan välillä

Socket Server Symbianin verkkorajapintaa toteuttava palvelu

API Application Programmin Interface, rajapinta jonka kautta kirjastoja voidaan käyttää

Active Scheduler Aktiiviajoittaja, Symbianin palvelu joka mahdollistaa asynkronisen toiminnan prosessissa

Active Object Aktiiviolio, aktiiviajoittajan suorittamia olioita

S60 Graafinen puhelinkäyttöliittymä ja -alusta

GUI Graphical User Interface, graafinen käyttöliittymä

DLL Dynamik Link Library, jaettu kirjasto

DEF DEF-tiedosto pitää kirjaa Symbianin kirjastojen jaetuista metodeista

1. JOHDANTO

Työn tehtävänä oli ohjelmoida perustoiminnallisuudet toteuttava Internet Relay Chat -asiakasohjelma. Sovellus koostuu moottori- ja käyttöliittymäosista. Asiakasohjelman moottorin toteutus on tämän työn tärkein painopiste. Moottori koostuu verkkorajapintatoteutuksesta Socket Serverin kanssa sekä protokollan parsimisen ja käsittelyn toteuttavasta kirjastosta. Tavoitteena on harjoitella Symbian OS -ohjelmointia ja toteuttaa yleiskäyttöinen IRC-asiakaskirjasto, jonka käyttöä varten toteutetaan myös yksinkertainen käyttöliittymä S60-ympäristössä.

IRC-asiakasohjelma toimii hyvin perinteisesti asiakas-palvelin-arkkitehtuurin mukaisesti. Asiakasohjelma ottaa yhteyden palvelinten muodostamaan verkkoon, jonka kautta käyttäjät viestivät keskenään. Kappaleessa 2 kerrotaan asiakas-palvelinmallista ja sovelluksen arkkitehtuurista.

Työn tarkoituksena oli myös opetella Symbian OS -sovelluskehitystä. Symbianin Socket Server -palvelu mahdollistaa tiedonsiirron verkon yli ja luo perustan sovelluksen toiminnallisuudelle. Olennaista toteutuksessa oli myös aktiiviolioiden sekä muiden Symbianilla käytettävien ohjelmointimallien oikeaoppinen käyttö. Kappale 3 esittelee Symbian OS -ympäristöä sekä toteutuksen kannalta olennaisia osia sen kirjastoista ja palveluista.

IRC on melko vanha ja kohtuullisen yksinkertainen protokolla, minkä vuoksi perustoiminnallisuudet omaavan sovelluksen toteutus on melko triviaalia. Protokollan yksinkertaisuuden vuoksi se toimi hyvänä lähtökohtana verkkosovelluksen toteutusta silmälläpitäen. IRC-verkosta ja sen rakenteesta kerrotaan kappaleessa 4.

Kappale 5 esittelee työn toteutuksen. Kappaleessa 6 on esitelty tässä työssä käytetylle toteutukselle vaihtoehtoinen malli sekä jatkokehitysajatuksia. Kappaleessa 7 käsitellään työn lopputuloksia sekä pohdintaa toteutuksen onnistumisesta.

2. ASIAKASSOVELLUS HAJAUTETUSSA JÄRJESTELMÄSSÄ

Yhä useammat sovellukset toimivat tietoverkon välityksellä keskenään. Modernit tietoverkot ja mobiililaitteet mahdollistavat joustavien verkkojen rakentamisen.

Vaikka asiakas-palvelin-arkkitehtuuri mielletään yleensä tietoverkkoihin kuuluvaksi malliksi, käytetään sitä myös monissa muissa yhteyksissä. Symbian OS:n rakenne perustuu vahvasti asiakas-palvelin-arkkitehtuuriin. Monet järjestelmän palvelut toimivat palvelinsäikeinä, joihin useat asiakassäikeet voivat olla yhteydessä samanaikaisesti.

Sovellusten ja palveluiden jako asiakas-palvelin-mallin mukaisesti mahdollistaa rinnakkaisuuden eri komponenttien välillä. Lisäksi jako mahdollistaa paremman laajennettavuuden, suorituskyvyn ja turvallisuuden.

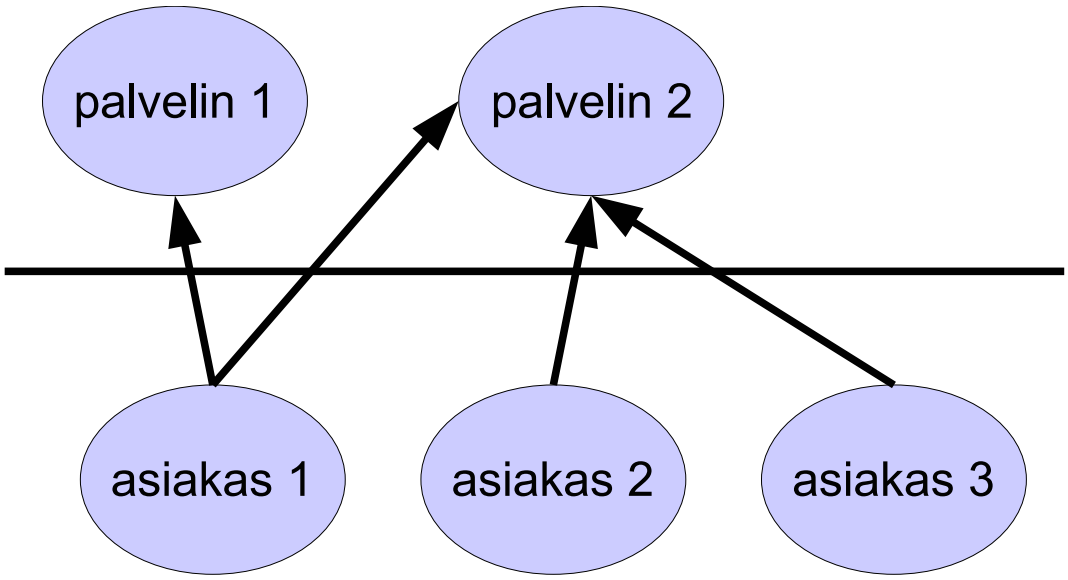
2.1 Asiakas-palvelin-arkkitehtuuri

Asiakas-palvelin-arkkitehtuuri on tällä hetkellä ehkä kaikkein yleisimmin käytetty arkkitehtuuriratkaisu. [7] Kun tietyn arkkitehtuuritason resurssin hallinta (palvelin) kapseloidaan, resurssin käyttäjän (asiakas) ei tarvitse huolehtia resurssin käyttöön liittyvistä teknisistä ongelmista. Asiakas voi käyttää jotain resurssia palvelimelta vapaasti riippumatta muista asiakkaista, kuten kuvassa 2.1) näkyy. [7]

Myös Symbian OS toimii hyvin pitkälle asiakas-palvelin-arkkitehtuurin mukaisesti. Usein asiakkaan ja palvelimen välinen vuorovaikutus tapahtuu istunnon aikana. Palvelimet odottavat yleensä passiivisina asiakkaan yhteydenottoa, jolloin istunto muodostetaan ja uutta tapahtumaa aletaan suorittaa. Asiakas ja palvelin toimivat selkeästi erillään toisistaan. Palvelin voi olla esimerkiksi verkkoyhteyden päässä, mutta joka tapauksessa se suoritetaan omassa säikeessään erillään asiakkaasta. Varsinkin Symbian OS:ssä asiakas-palvelin-arkkitehtuurin yksi merkittävimmistä eduista on selkeä työnjako. [7]

2.2 Asiakas-palvelin-arkkitehtuuri Symbian OS -ympäristössä

Käytännössä kaikki asynkroniset palvelut Symbian OS:ssä on toteutettu palvelimina asiakas-palvelin-arkkitehtuurin kautta. Asiakas-palvelin-arkkitehtuurin käyttö



Kuva 2.1: Asiakas-palvelin-arkkitehtuuri [7]

palvelujen toteuttamiseen mahdollistaa laajennettavuuden, suorituskyvyn, turvallisuuden sekä asynkronisuuden.

Palveluita on helppo laajentaa käyttämällä liitännäisiä. Usein tämä voidaan vielä tehdä ajon aikana, jolloin laite voi esimerkiksi käyttää uuden tyyppistä tiedostojärjestelmää, joka on kehitetty myöhemmin kuin tiedostopalvelin, joka sitä käyttää.

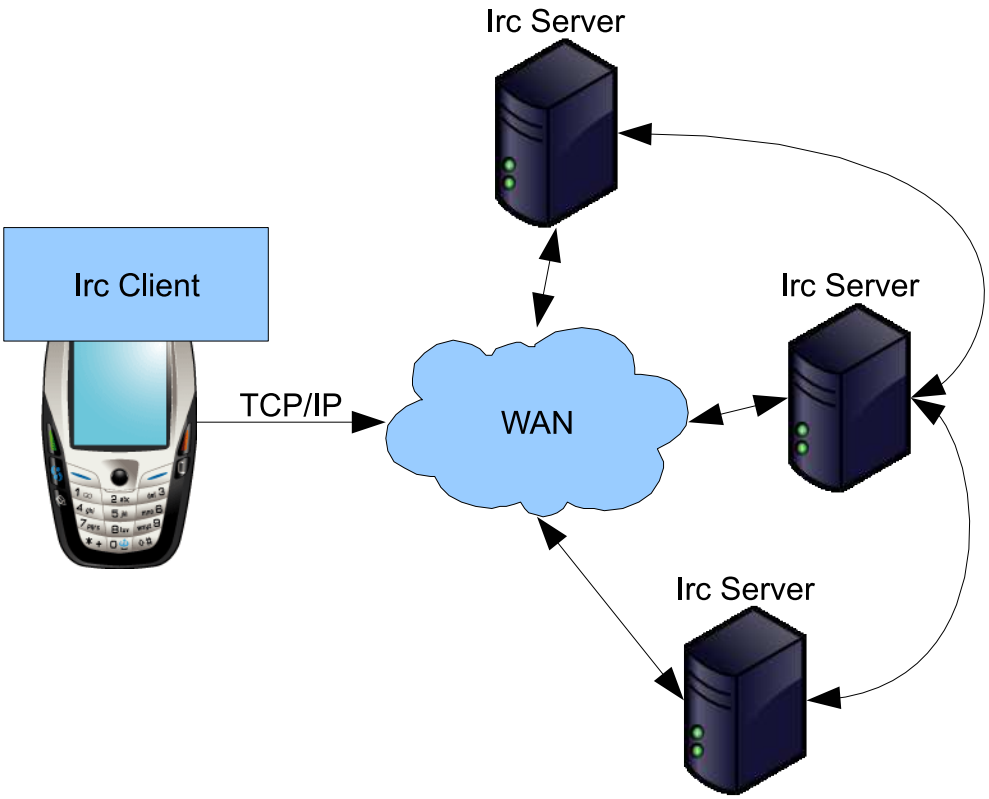
Suorituskykyyn vaikuttaa se, että yksi palvelin voi palvella useita asiakkaita samanaikaisesti. Palvelin voi myös rajoittaa tai estää uusien asiakkaiden pyyntöjä, jos resurssit ovat vähissä.

Koska palvelimet ja niiden asiakkaat yleensä sijaitsevat eri prosesseissa ja keskustelevat viestejä käyttäen, on palveluiden tarjoaminen turvallisempaa. Huonosti käyttäytyvä asiakasprosessi ei välttämättä saa suurta vahinkoa aikaiseksi palvelimeen, jota se käyttää.

Asynkroninen toiminta saadaan aikaiseksi, kun palvelimet käyttävät aktiiviolio-rajapintaa ilmaisemaan operaatioiden suorituksesta asiakkaalle. Aktiiviolioiden käyttö palvelimien kanssa on hyödyllistä. Näin asiakas voi olla pysäytetyssä tilassa palvelun suorituksen aikana, eikä hänen tarvitse tiedustella suorituksen tilaa jatkuvasti. Asynkroninen toiminta helpottaa virran säästämisessä, mikä on todella tärkeää mobiililaitteessa. [6]

2.3 Sovelluksen arkkitehtuuri

Sovelluksen nimeksi tuli iIrc, eli induction Irc. Se toimii S60 3rd edition -laitteissa. Sovellus käyttää Symbian OS:n sockettirajapintaa keskustellessaan TCP/IP-verkossa olevien palvelimien kanssa. Yhteydet on kuvattu kuvassa 2.2.



Kuva 2.2: Irc-ohjelman järjestelmäkuvaus

3. SYMBIAN OS

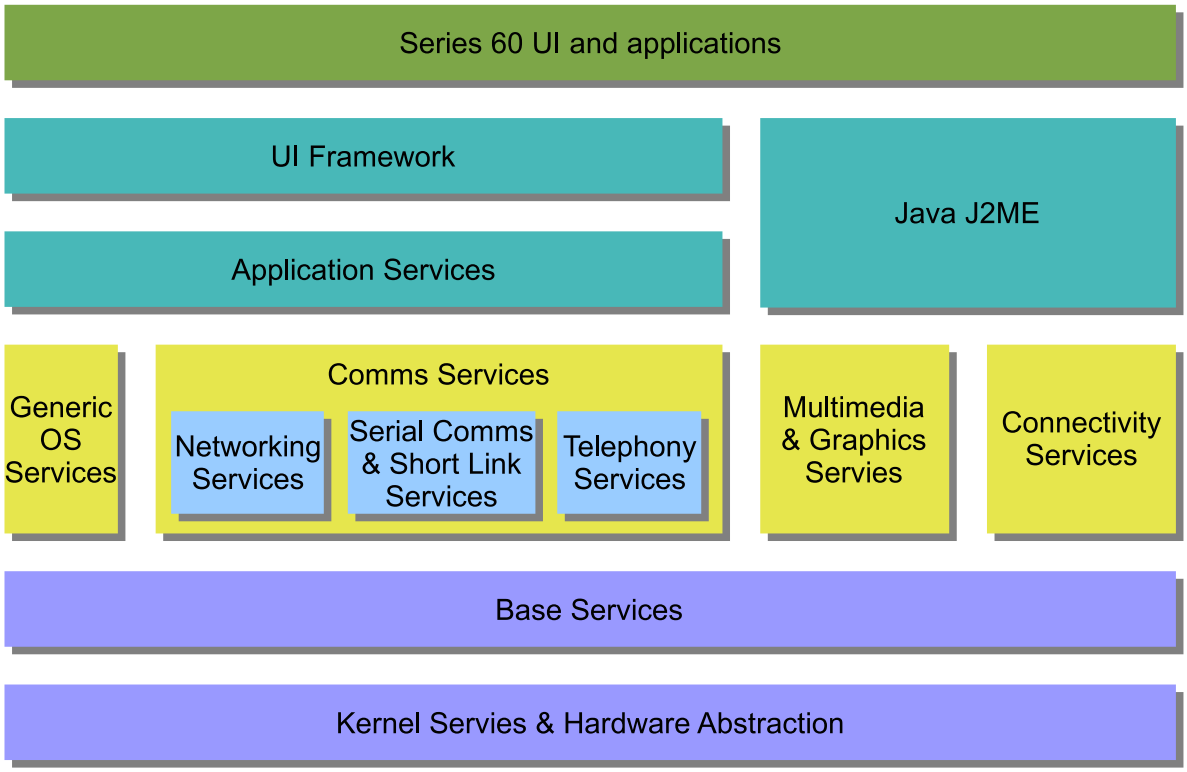
Symbian OS on ROM- ja mikrokernelipohjainen vähän virtaa kuluttava ympäristö. Mikrokerneliluonteensa vuoksi hyvin pieni osa palveluista ajetaan ytimen tasolla ja monet käyttöjärjestelmäkomponentit käyttäjäoikeuksin palvelinprosesseina. Symbianissa jokainen applikaatio ajetaan omassa prosessissa, eikä sillä ole näkyvyyttä kuin omaan muistialueeseensa. Symbian on suunniteltu käyttämään yksisäikeistä sovelluskehitystä. [10]

Symbian on myös komponenttipohjainen ja suunniteltu toimimaan erilaisilla alustoilla ja resursseilla. Yksi tekijä näiden toteuttamisessa on oliomallinen rakenne. Lähes kaikki komponentit ovatkin C++:aa alusta loppuun. [10]

Ensimmäinen kahdeksanbittinen Symbian OS -versio tehtiin 1980-luvun alkupuolella, jolloin sen nimi oli vielä EPOC. Kuusitoistabittinen EPOC16 julkaistiin saman vuosikymmenen lopussa. EPOC32 julkaisun jälkeen on bittisyyttä ilmaiseva loppuosa jätetty pois käytöstä. EPOCista tuli Symbian platform vuonna 2000 ja vuotta myöhemmin nimeä viilattiin nykyiseen muotoonsa.

3.1 Käyttöjärjestelmän rakenne

Symbian OS on jaettu viiten eri osaan, ydin (kernel), peruspalvelut (base), järjestelmä (system), applikaatiopalvelut, ja käyttöliittymäkehys. Jokainen osa koostuu vielä yhdestä tai useammasta alijärjestelmästä. Kuvasta 3.1 näemme Symbian OS:n osat, alimpana ydin ja päällimmäisenä käyttöliittymäkehys. Kaikkein ylimpänä, kuudentena, on vielä käyttöliittymäkehyksen toteutus, tässä S60-käyttöliittymä. [8]

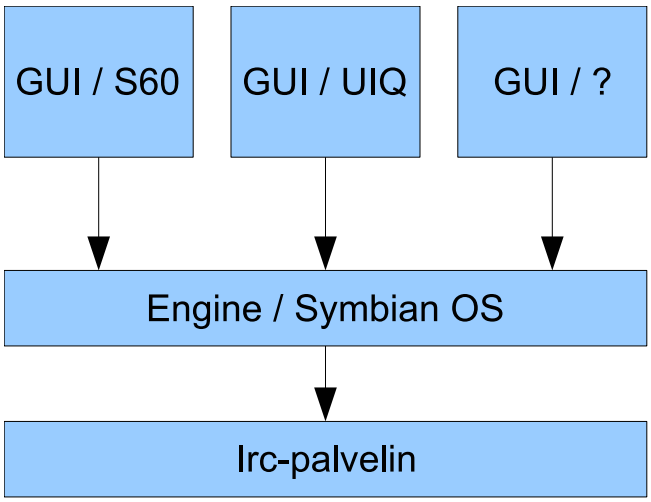


Kuva 3.1: Symbian OS-järjestelmän rakenne

3.2 Käyttöliittymä ja moottori

Ohjelmat jaetaan yleensä korkeammalla tasolla kahteen osaan, käyttöliittymään ja moottoriin. Tässä jaottelussa moottori tekee varsinaisen työn ja datan prosessoinnin, jonka käyttäytymistä ja tilan seurantaa käyttöliittymästä hallitaan. Kuva 3.2 esittää iIrc-sovelluksen karkean tason jakautumisen käyttöliittymään ja moottoriin. Käyttöliittymän kautta voidaan katsella eri kanavien keskusteluhistoriaa ja palvelimen viestihistoriaa, mutta kaikki näytettävä data luodaan moottorissa. Myös sovelluksen asetusten lataamisen ja tallentamisen hoitaa moottori. Käyttöliittymän kautta asetukset voidaan muuttaa käyttäjäystävällisesti. [1]

Tästä sovelluksen jaottelusta käyttöliittymään ja moottoriin on monia etuja ja



Kuva 3.2: Sovelluksen jakaminen käyttöliittymään ja moottoriin

käytännössä vain pieniä haittoja. Haitatkin voidaan minimoida hyvällä suunnittelulla. Jaottelun suurimpia hyötyjä on sovelluksen toteutuksen eriyttäminen kahteen haaraan. Kun moottorin tarjoama rajapinta ja palvelut on tarkasti määritelty ja suunniteltu, voidaan sovelluksen osia parhaassa tapauksessa kehittää hyvin itsenäisesti ja tehokkaasti rinnakkain. Lisäksi moottorista tulee parhaimmillaan hyvin uudelleenkäytettävä. Kun sovelluksen moottori ei tarvitse toimiakseen käyttöliittymäkomponentteja, voidaan mottoria parhaimmassa tapauksessa käyttää suoraan esimerkiksi toisessa käyttöliittymäkehyksessä (vrt. S60 ja UIQ) ja uuteen sovellukseen tarvitsee toteuttaa vain oma käyttöliittymä.

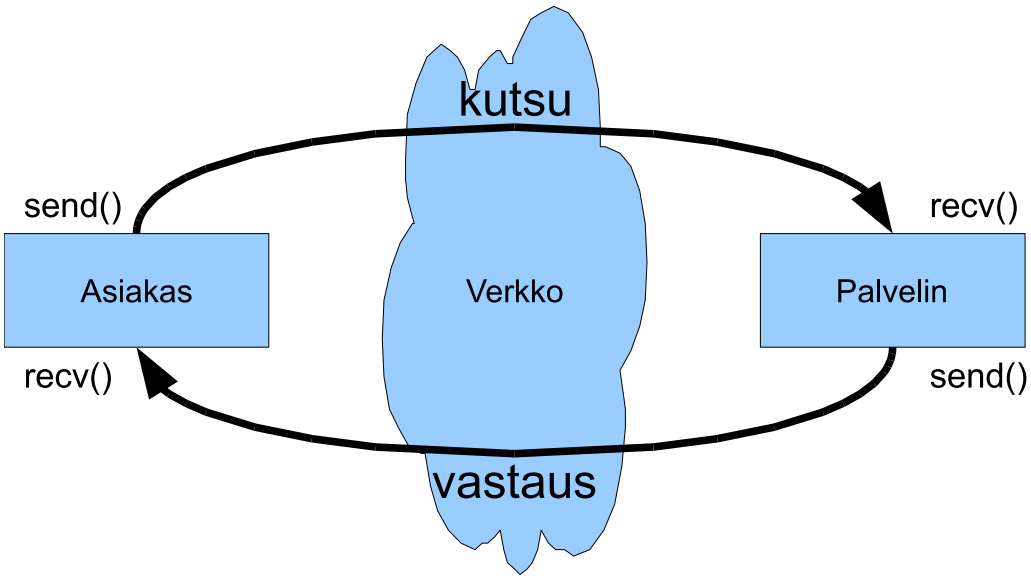
Ongelmana jaottelussa on Symbianin DLL-tiedostojen rakenne. Symbianin DLL-tiedosto täytyy suunnitella tarkasti jäädytyksen vuoksi. Kun kirjasto jäädytetään, irroitetaan siitä kaikki ylimääräinen osa. Jäädytyksen jälkeen listataan erityiseen DEF-tiedostoon tiedot siitä, missä järjestyksessä kirjaston ulkoistamat metodit löytyvät, eikä varsinaisesta kirjastosta metodien nimiä enää saa selville. Kun jäädytys on kerran tehty, ei periaatteessa enää voi tehdä muutoksia olemassaoleviin ulkoistettuihin metodeihin, järjestää olemassaolevia metodeja eri järjestykseen tai luoda uusia metodeja muualle kuin tiedoston loppupäähän. Siksi rajapinnan suunnittelussa tulee olla tarkkana, koska jäädytettyjen funktioiden muuttaminen vaikeuttaa myös käyttöliittymän ylläpitämistä.

3.3 Socketit ja SocketServer

Socketit tarjoavat korkeamman tason pääsyn kommunikaatioprotokolliin. Sockets API on alunperin kehitetty toteuttamaan TCP/IP-kommunikaatio BSD UNIX:ssa ja siitä on sen jälkeen tullut standardi rajapinta monissa ympäristöissä, kuten UNIX, Windows sekä Symbian OS. Kuvassa 3.3 esitetään yleinen vuorovaikutustilanne soc-

ketien kanssa toimiessa. Asiakas aloittaa yhteyden palvelimeen, ja lähettää ja vastaanottaa dataa avatun socketin kautta.

Symbianin Sockets API on hieman geneerisempi ja mahdollistaa muidenkin verkkotyyppien ja protokollien käytön. Rajapintaa voidaankin käyttää tiedonsiirtoon esimerkiksi infrapuna-, Bluetooth- ja WAP -protokollien avulla. [6]



Kuva 3.3: Asiakkaan ja palvelimen yhteistoiminta[15]

Sockets API:n toteutus Symbianissa on kuitenkin melko matalalla tasolla. Tästä syystä tarvitsee toteuttaa kohtuullisen paljon toiminnallisuutta itse, jotta tietoa pääsee siirtämään. Tarkemmin socket-tiedonsiirron toteutuksesta kerrotaan luvussa 5.

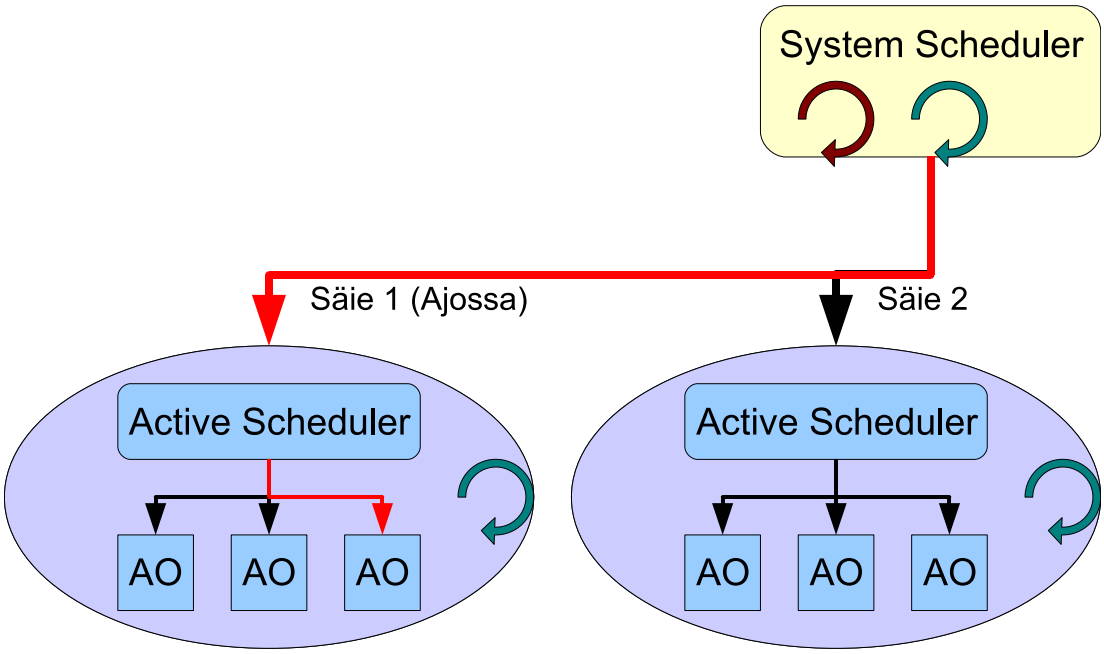
3.4 Aktiivioliot mahdollistavat asynkronisen toiminnan

Symbian OS on luonteeltaan vahvasti asynkroninen. Monet ohjelmat kutsuvat erilaisia järjestelmäpalveluita ja jäävät odottamaan asynkronista vastausta. Yleisiä järjestelmäpalveluita ovat esimerkiksi tiedostopalvelin tiedostoa avattaessa ja suljettaessa, verkkoliikennepalvelin, kun tietoa siirretään verkon yli tai ikkunamanageri, jolta odotetaan käyttäjän syötettä.

Esimerkiksi IRC-sovellus voi odottaa verkkoliikennettä tiedonsiirtopalveluilta samalla, kun käyttäjä selailee käyttöliittymää. Tämä on yksi tapa toteuttaa rinnakkaisuutta, joka Symbian OS:ssä on hoidettu aktiivioliioiden avulla.

Rinnakkainen toiminta voidaan toteuttaa myös esimerkiksi säikeillä, mutta aktiivioliot ovat Symbianissa suositellumpia. Koska käyttöjärjestelmä on suunniteltu mahdollisimman pieni resurssien kulutus päämääränä, saadaan tällä tavalla mahdollisesti pienempi tehontarve, kun kukin itsenäinen prosessi voidaan tarvittaessa

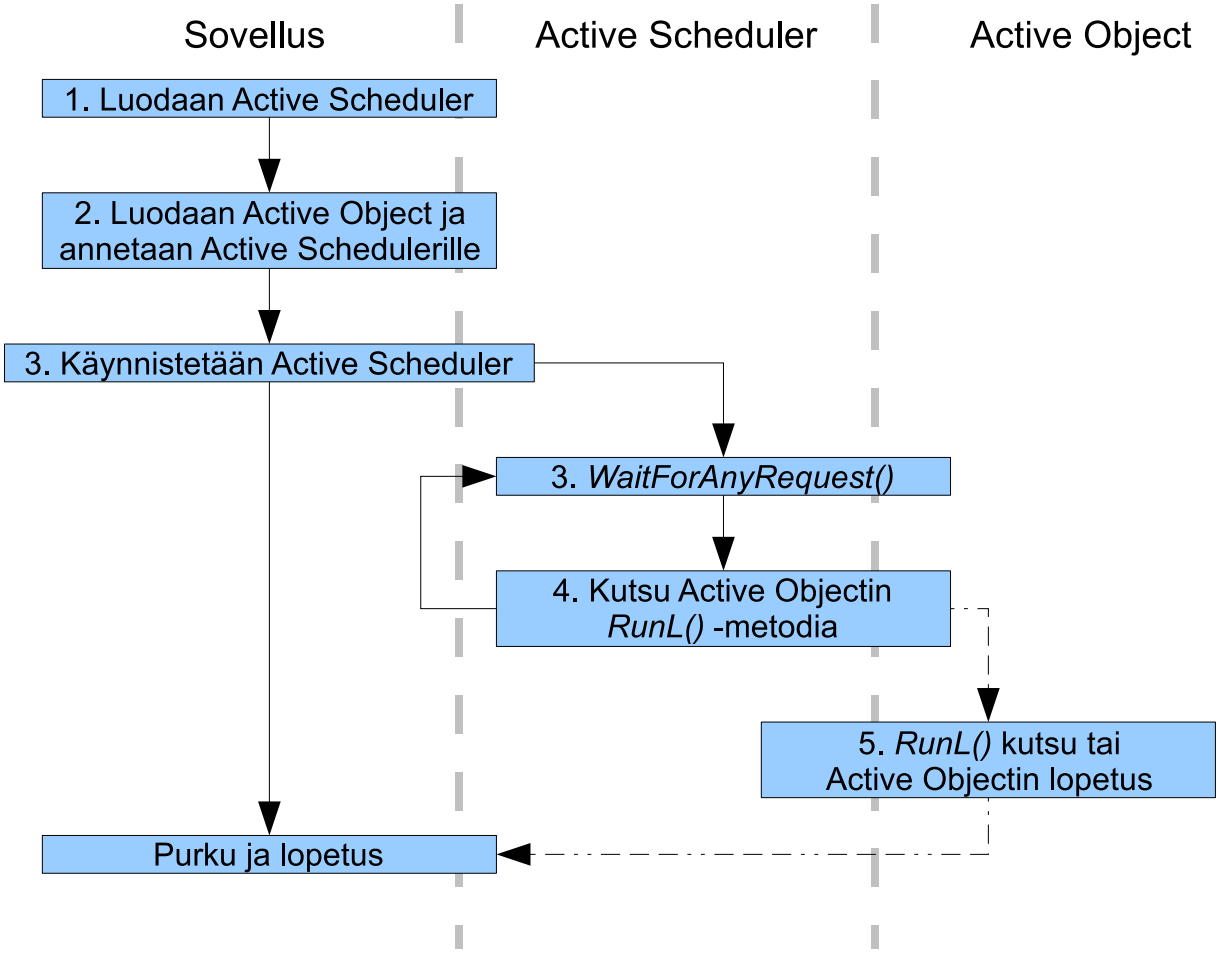
pysäyttää. Symbianissa jokainen sovellus ja palvelu on säie, jolla on omat resurssit. Aktiivioliorajapinta mahdollistaa moniajon yhdessä säikeessä. [10]



Kuva 3.4: Active Scheduler

Kun lähdetään toteuttamaan aktiivioliota, peritään ensin CActive-luokka, johon kuuluvat asynkroniset kutsut, ohjelman käsittelijä suoritetulle kutsulle ja toteutus, mikäli kutsu peruutetaan. Esimerkiksi kutsu voi olla ajaa tietokantakysely ja käsittelijä voi kutsua toiminnallisuutta, joka esittää saadut tulokset käyttäjälle. Aktiivioliolioiden suoritusta hallinnoi aktiiviajoittaja, joita on yksisäikeisissä sovelluksissa yksi. Aktiiviajoittaja suorittaa aktiivioliolioiden tapahtumia niiden prioriteettien määräämässä järjestyksessä.

Aktiiviolion elinkaari on esitetty kuvassa 3.5.



Kuva 3.5: Active Objectin elinkaari

1. Active Scheduler luodaan ja asennetaan. Yksi ajoittaja voi käsitellä yhden tai useamman aktiiviolion tapahtumat.

2. Ennen kuin aktiiviajoittaja käynnistetään, täytyy luoda ainakin yksi aktiiviolio ja sen *Request()*-metodia kutsua.

3. Kun aktiiviajoittaja käynnistetään, se kutsuu *WaitForAnyRequest()* -metodia. Sovelluksen säie odottaa signaalia aktiiviolion toiminnon päättymistä.

4. Aktiiviajoittaja käy läpi jokaisen aktiiviolion tilatiedot ja löytäessään olion jonka suoritus on valmis kutsuu ajoittaja löydetyn olion *RunL()* -metodia.

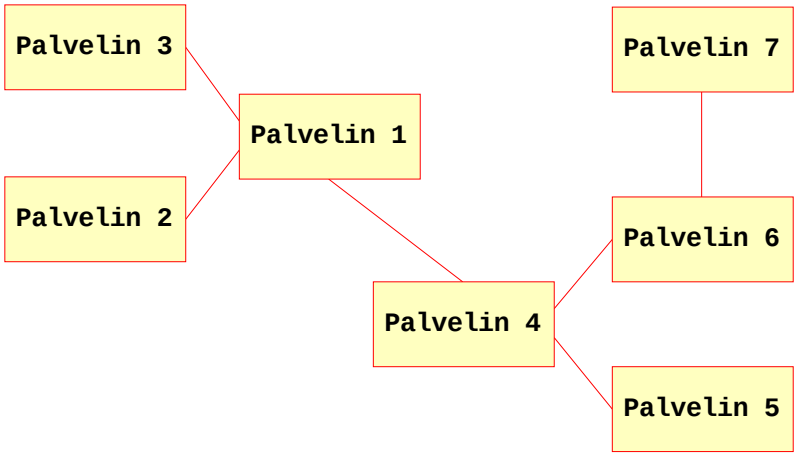
5. *RunL()* -metodi voi toistaa *Request()*-kutsun tai pysäyttää aktiiviajoittajan.

4. INTERNET RELAY CHAT

Internet Relay Chat on avoin tekstimuotoinen joukkokeskusteluprotokolla, joka toimii TCP/IP-verkon yli. Ensimmäisen version kehitti Jarkko Oikarinen vuonna 1988. IRC on alkujaan ollut yksi merkittävimmistä verkkokeskusteluprotokollista. Vaikka sen merkitys nykyään on pienentynyt muiden verkkokeskusteluohjelmien yleistyttyä, on sillä edelleen hyvin vankka käyttäjäkunta esimerkiksi tietotekniikkaharrastajien joukossa. Maailmalla pyörii tuhansia IRC-verkkoja, joista suurimpia ovat EFnet, IRCnet ja QuakeNet. [11]

IRC-protokollasta ei ole vielä julkaistu virallista spesifikaatiota, mutta ensimmäinen löyhä esitys protokollan toiminnasta julkaistiin jo vuonna 1993. Sitä on myöhemmin laajennettu, mutta siitä huolimatta protokolla on melko dynaaminen, eivätkä kaikki palvelin- tai asiakassovellukset toteuta koko protokollaa samoin tai täydellisesti. Lisäksi on olemassa myös spesifikaatioon kuulumattomia laajennuksia, esimerkiksi Microsoftin suljettu kaupallinen IRCX-protokolla. Protokollan viestien merkistön koodausta ei ole määritelty, mikä ennestään laajentaa mahdollisten tulkintojen määrää. [11]

4.1 Palvelimet ja asiakkaat muodostavat verkon



Kuva 4.1: IRC-verkon rakenne

IRC-verkko koostuu palvelimista, jotka ovat yhteydessä toisiinsa (kuva 4.1) sekä näihin palvelimiin yhdistyneistä asiakkaista (kuva 4.2). Verkko muodostaa puurakenteen, jossa asiakkaat ovat yhdistyneet yhteen palvelimeen ja palvelimet ovat yhteydessä toisiinsa muodostaen suurempia yhtenäisiä verkkoja. Yhteen palvelimeen yhdistynyt asiakas näkee vain oman palvelimensa. Joissain tapauksissa yhteys eri osien välillä voi olla myös salattu SSL:llä, mutta se on melko harvinaista. Koska protokolla on tekstipohjainen on mahdollista kirjautua IRC-verkkoon vaikka tavallisella telnet-ohjelmalla. Vaikka IRC on kehitetty käyttäen TCP/IP-protokollaa, ei se varsinaisesti ole vaatimus sen toiminnan kannalta. [12]

4.2 Käyttäjät ja ryhmät

IRC-verkko on sisäisesti jaettu erilaisiin ryhmiin, joilla rajoitetaan sekä käyttäjien näkyvyyttä että viestien välitystä eri käyttäjille. Ryhmistä puhuttaessa käytetään kuitenkin yleensä termiä kanava (Channel).

#kanava Jaettu kanava (Distributed Channel)

&kanava Rajoitettu kanava (Server Channel)

Ensimmäinen on jaettu kanava (Distributed Channel), joka on kaikkien verkon palvelimien tiedossa. Tällaiselle kanavalle lähetetty viesti välitetään myös muihin palvelimiin yhteydessä oleville asiakkaille, jotka kuuluvat kyseiselle kanavalle. Jaetun kanavan tunnistaa kanavan nimeä edeltävästä '#'-merkistä.

Toinen kanavatyyppi on rajoitettu kanava (Server Channel). Rajoitetun kanavan näkyvyys on vain samalla palvelimella oleville asiakkaille, eikä viestejä välitetä toisille palvelimille. Rajoitetun kanavan etuliite on '&'-merkki.

1. Käyttäjä (User)
2. Kanavaoperaattori (Channel Operator)
3. Verkko-operaattori (IRC Operator)

Käyttäjät voivat liittyä ja poistua kanavilta, keskustella toisien käyttäjien kanssa, hallita yleisiä itseensä kohdistuvia asetuksia ja tehdä kyselyjä toisista käyttäjistä. Yleensä ensimmäinen käyttäjä uudella kanavalla on myös automaattisesti kanavan kanavaoperaattori.

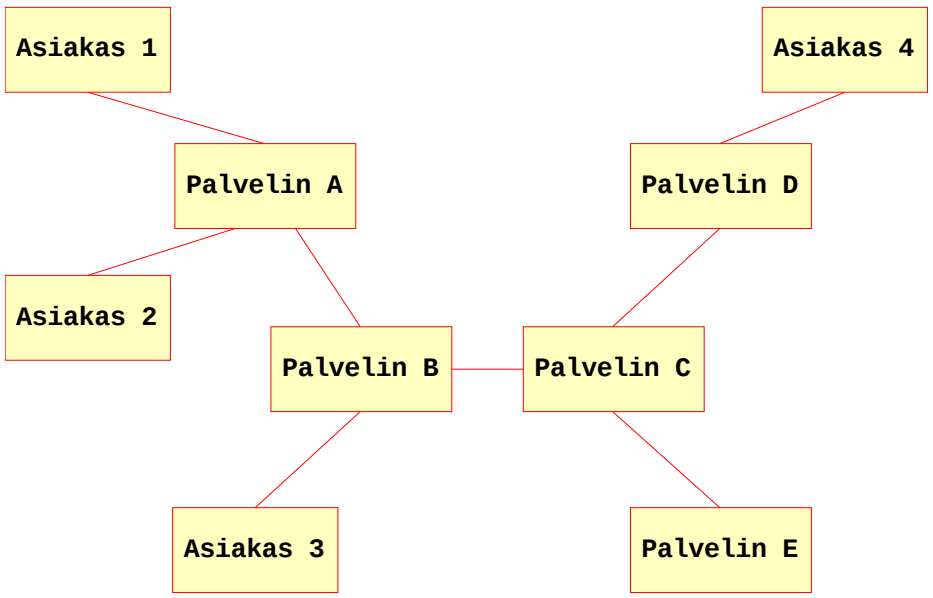
Kanavaoperaattorit hallitsevat kanavien ylläpitoa. He voivat muokata kanavan asetuksia ja rajoituksia ja yleensä myös hoitavat yleisen järjestyksen ja kanavan sääntöjen noudattamista.

Verkon palvelimien ylläpitotehtäviä hoitaa käyttäjäryhmä verkko-operaattorit. Verkko-operaattorit yleensä kehittävät IRC-verkkoa ja valvovat, että sääntöjä noudatetaan. Verkko-operaattoreita on kahta tyyppiä, paikallisia ja globaaleja. Paikalliset operaattorit pystyvät hallitsemaan vain heille osoitettuja palvelimia, kun taas globaalit operaattorit voivat hallita kaikkia verkon palvelimia.

4.3 Asiakkaat viestivät palvelinten kanssa

Tietoa siirretään verkossa asiakkaan ja palvelimen sekä palvelimen ja palvelimen välillä. Asiakkaat viestivät toisilleen tai ryhmille vain palvelimen kautta. Palvelinten välinen liikenne koostuu suurimmaksi osaksi eri asiakkaiden viestien välittämiseksi palvelimien kautta, mikäli asiakkaat ovat liittyneinä verkkoon eri palvelimien kautta.

Kun kaksi asiakasta keskustelee suoraan toistensa kanssa, eivät muut asiakkaat näe viestiä. Viesti välitetään asiakkaalta toiselle käyttäen pienintä määrää palvelimia viestin siirtämiseen. Esimerkiksi asiakkaan **1** lähettäessä viestin asiakkaalle **2**, viestin näkevät vain palvelin **A** ja asiakas **1**. Mikäli asiakas **3** lähettäisi viestin asiakkaalle **4**, viesti kulkisi palvelimien **B**, **C**, ja **D** kautta asiakkaalle **4**. Palvelinten ja asiakkaiden yhteydet nähdään kuvassa 4.2.



Kuva 4.2: Asiakkaiden ja palvelimien yhteydet [12]

Seuraavat esimerkit pohjautuvat kuvaan 4.2. Asiakkaat **1** ja **2** kuuluvat kanavalle **&yksityinen**. Koska kanavan näkyvyys on vain palvelimelle **A**, voisi myös palvelimella **B** olla kanava nimeltään **&yksityinen**, jolla olisi eri asiakkaat. Mikäli asiakkaat **1**, **3**, ja **4** kuuluisivat kanavalle **#porukka**, kanavalle lähetetyt viestit siirrettäisiin kaikkien palvelimien **A**, **B**, **C**, ja **D** kautta eri asiakkaille, asiakas **2** ei viestiä saisi.

4.4 Protokolla siirtää tiedon

Asiakkaan ja palvelimen viestintä on asynkronista. Jokainen viesti koostuu korkeintaan kolmesta osasta: etuliite (ei pakollinen), komento ja komennon parametrit. Parametreja taas voi olla yhdessä viestissä enintään viisitoista. Osat on erotettu toisistaan yhdellä tai useammalla välilyönnillä. Viestien maksimipituus on 512 merkkiä lopetusmerkkien CR-LF kanssa, jolloin komennolle ja mahdollisille parametreille jää 510 merkkiä.

```
PRIVMSG #kanava :Mennään Matin kanssa tänään keikalle, onko muita tulossa? (1)

PRIVMSG Matti :Tottahan toki! Nähdään illalla. (2)

:Matti PRIVMSG Teppo :Moi! Lähdetkö keikalle tänään? (3)
```

Esimerkkinä edellä viestin lähetys. Viesti voidaan lähettää eri ryhmille tai yhdelle käyttäjälle. Yleisimpiä tapauksia ovat viestin lähetys samalle kanavalle kuuluville

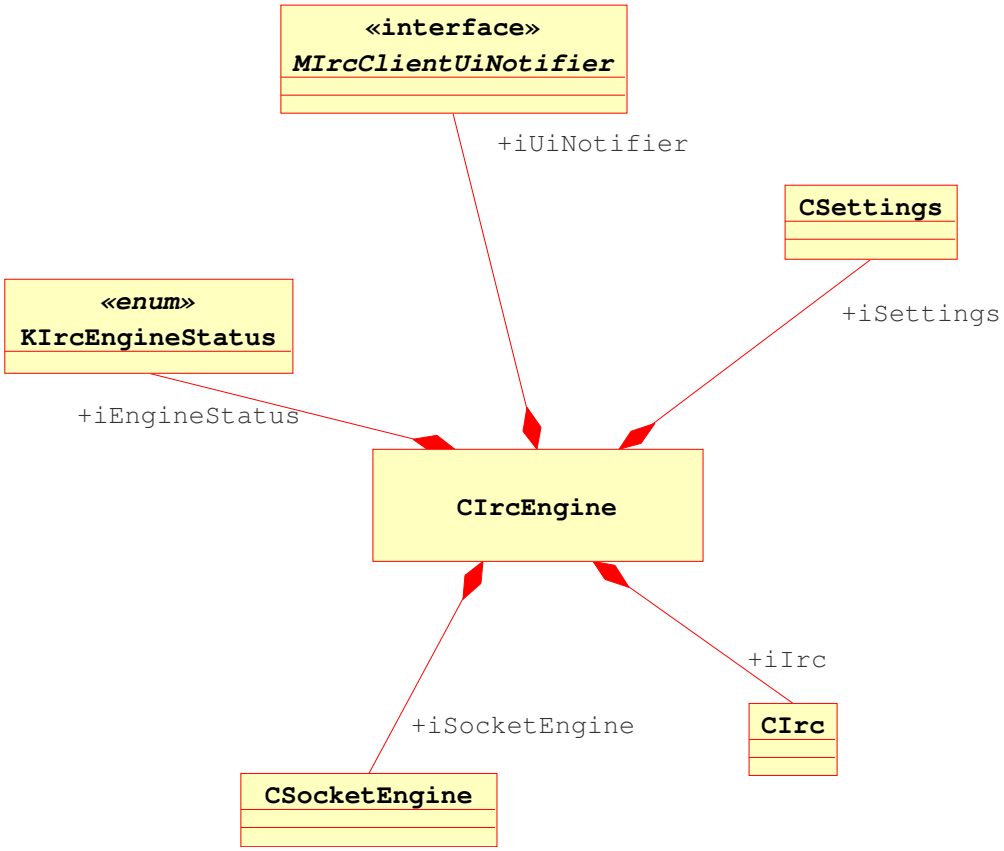
(kohta 1) tai yhdelle tietylle käyttäjälle (kohta 2). Kolmannella rivillä (kohta 3) näkyy asiakasohjelman vastaanottama viesti. Palvelimelle lähetettyjä viestejä ei kaiuteta tai niihin muutenkaan reagoida. Asiakas voi vain toivoa, että viesti on mennyt perille. Liitteessä A on lyhyt esimerkki telnet-istunnosta IRC-palvelimen kanssa.

5. MOOTTORIN TOTEUTUS

Mobiililaitteiden sovelluskehityksessä on monia asioita, joita tulee huomioida myös toteutusvaiheessa. Vaikka laitteiden suorituskyky on kasvanut, ovat laitteiden suorittimet verrattain hitaita ja muistia vähän. Laitteet ovat myös pitkään päällä, jolloin pitää olla erityisen tarkkana muistin käytön kanssa. Muistia ei saa vuotaa eikä ohjelman suoritus saisi lukita tai uudelleenkäynnistää laitetta. Myös laitteen fyysiset rajoitukset tulee huomioida, näyttöruutu on melko pieni ja useimmissa laitteissa tekstin syöttö on vaivalloista ahtaan näppäimistön vuoksi. [1]

5.1 Viestien käsittely

IrcEngine-luokka on käytännössä IRC-moottorin keskeisin ja käyttöliittymälle näkyvin osa. IrcEnginessä on käyttöliittymälle näkyvä API. Luokan sisältämät luokat ja niiden suhteet näkyvät kuvasta 5.1. IrcEngine-luokka itsessään toteuttaa suurimman osan datan käsittelystä.



Kuva 5.1: Yleiskuvaus IrcEnginen luokista ja julkinen API

IrcEnginen tehtävänä on yhdistää eri osien toiminnallisuus ja suorittaa suurin osa ohjelman logiikasta. IrcEnginessä on toteutettu eri IRC-protokollan komentojen käsittelijät sekä vastaavasti komennot, joilla viestitään palvelimelle päin. Lisäksi IrcEngine käynnistää ja sulkee socket-yhteyden käyttäen SocketEngineä. Asetusten hallinnan hoitaa CSettings-luokka, joka käytännössä paljastetaan suoraan toteutuksen yksinkertaistamiseksi. Käyttöliittymäkomponentteihin päin lähtevät komennot on esitelty abstraktissa luokassa MIrcClientUINotifier.

Viestintä palvelimen ja moottorin välillä on asynkronista, jolloin kaikki moottorin metodit ovat vain pyyntöjä suorittaa jokin komento. Mahdollinen vastaus tulee myöhemmin, toteutus ei millään tavoin pyri yhdistämään pyyntöjä mahdollisiin vastauksiin. Tämä on otettu huomioon myös käyttöliittymän suunnittelussa. Esimerkiksi kanavien luonti tapahtuu automaattisesti vasta kun palvelin ensimmäisen

kerran lähettää viestin koskien uutta kanavaa.

Yleisin viesti varsinkin pitkän session aikana on IRC-verkon heräte-viesti, PING. Asiakasohjelman tulee vastata jokaiseen PING-viestiin tietyn ajan sisällä, jotta palvelin ylläpitää yhteyttä. Muussa tapauksessa palvelin sulkee yhteyden asiakkaaseen ja ilmoittaa siitä Ping timeout -viestillä muille.

5.2 Protokollan hallinta

Kuten luvusta 4 näemme, IRC-protokolla koostuu viesteistä. IrcEngine purkaa staattista apuluokkaa CIrc käyttäen vastaanotetut protokollan viestit ja käsittelee ne asianmukaisesti.

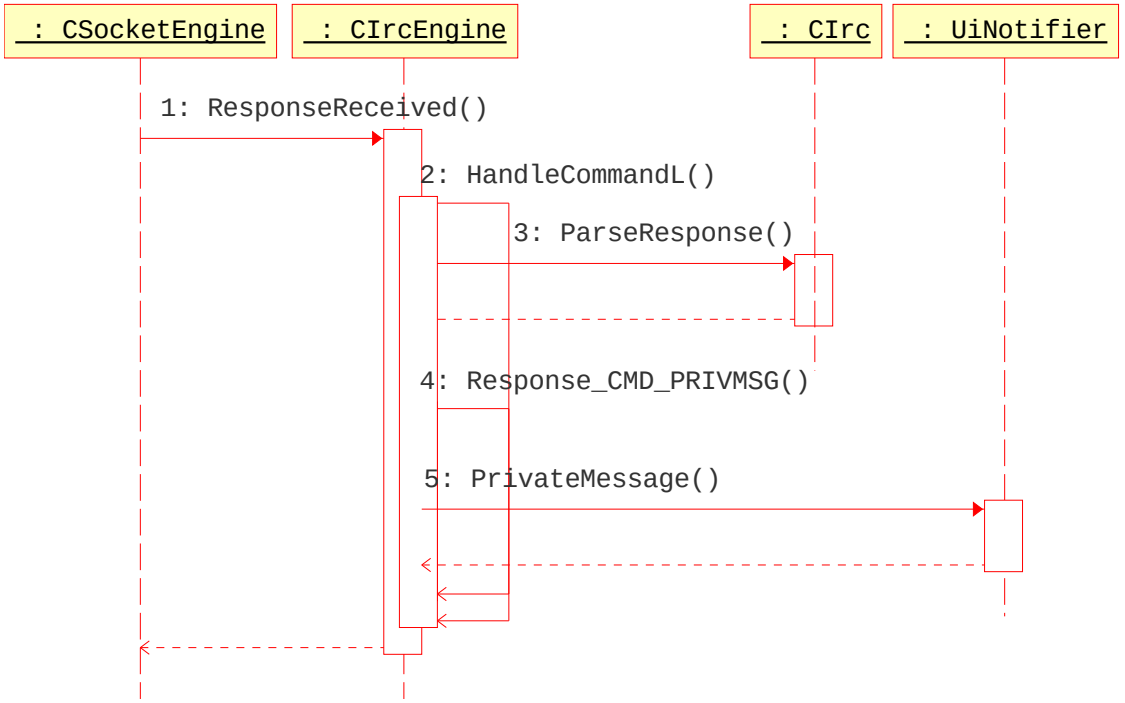
IRC-protokollan erilaiset vakiot, komentojen nimet sekä vastauksien numerokoodit, on varastoitu CIrc-luokan sisälle. Toteutettujen ominaisuuksien nimet sekä numerokoodit on listattu staattisesti suoraan luokkaan, mutta niiden seuraaminen on tehty mahdollisimman helpoksi listaamalla ne loogisesti luokan otsikkotiedostossa.

```
enum KResponseType
{
    RPL_TOPIC = 332,
    RPL_TOPIC_TIMESTAMP = 333,
    RPL_NAMREPLY = 353,
    RPL_ENDOFNAMES = 366,
    RPL_ENDOFMOTD = 376,
    ERR_UNKNOWNCOMMAND = 421,
    CMD_PRIVMSG = 1000, // 0
    CMD_PING,
    CMD_PONG,
    CMD_JOIN,
    CMD_PART,
[...]
    UNKNOWN_RESPONSE = -1
};
```

Nimellisten komentojen (käytännössä kaikki CMD-alkuiset) merkkijonot on tallennettu RArray-taulukkoon. Indeksointi on toteutettu niin, että jokainen komento-merkkijono on enumeraation kokonaisluku, josta on vähennetty tuhat.

```
void ConstructL()
{
    iResponseTypeStr.Append(_L8("PRIVMSG")); // 0
    iResponseTypeStr.Append(_L8("PING"));
    iResponseTypeStr.Append(_L8("PONG"));
    iResponseTypeStr.Append(_L8("JOIN"));
    iResponseTypeStr.Append(_L8("PART"));
[...]
};
```

Kun CSocketEngine vastaanottaa tietoa socket-serveriltä, lähettää se ensin tämän merkkijonon IrcEnginelle. IrcEngine kutsuu CIrc-luokan *ParseResponse()*-metodia. *ParseResponse()*-metodi jakaa saamansa merkkijonon IRC-protokollan viestin osiin, komento-osaan, etuliitteeseen, ja mahdollisiin komennon parametreihin. Esimerkkinä käyttäjän **matti** lähettämä yksityisviesti käyttäjälle **teppo**.



Kuva 5.2: Palvelimelta saapuvan PRIVMSG-tyyppisen viestin käsittely

1. Kun IRC-palvelimelta saapuu laitteelle päin uutta dataa, SocketEnginen SocketReader-aktiivioliota kutsutaan ja sille välitetään socketiin tullut data. SocketReader välittää tiedon edellen eteenpäin SocketEngine-luokalle, joka kutsuu IrcEnginen *ResponseReceived()*-metodia.

2. IrcEnginen *ResponseReceived()*-metodi jakaa mahdollisen datan sisältämät useat viestit (eroteltuina \r\n) ja kutsuu jokaisen viestin kohdalla *HandleCommandL()*-metodia.

3. *HandleCommandL()*-metodissa luodaan deskriptorit, joihin palvelimelta saatu data paloitellaan käyttäen CIrc-luokan *ParseResponse()*-metodia. *ParseResponse()* myös selvittää viestin tyypin. Tässä tapauksessa viesti on tyyppiä PRIVMSG.

4. Kun viesti on paloiteltu tarvittaviin palasiin, kutsutaan viestin tyypin mukaista käsittelijämetodia, *Response_CMD_PRIVMSG()*. *Response_CMD_PRIVMSG()* purkaa edelleen viestin osia, selvittää lähettävän käyttäjän käyttäjänimen etuliitteestä, kanavan nimen parametreistä sekä viestiosan parametreistä.

5. Tarvittavat parametrit on nyt purettu viestistä ja kutsutaan käyttöliittymäkomponentin *PrivateMessage()*-metodia. Käyttöliittymä huolehtii tämän jälkeen viestin näyttämisestä ja tarvittaessa kanavanäkymän luonnista.

```
:matti!~matti@client.maailmalla.fi PRIVMSG teppo :Hei!
```

Viestin osat ovat

etuliite matti! matti@client.maailmalla.fi

komento PRIVMSG

parametrit teppo :Hei!

Parametreistä ensimmäinen osoittaa viestin kuuluvan käyttäjälle tunnuksella teppo. Tämä tieto on olennaista käytännössä palvelinten välisessä liikenteessä, kun käyttäjät ovat eri palvelimien kautta samalla kanavalla. Toinen parametri sisältää yksityisviestin sisällön.

Vastaanotettujen komentojen parsimisen lisäksi CIrc-luokalla on myös toinen tärkeä tehtävä. Siihen on toteutettu palvelimelle lähetettävien viestien "valmistusohjeet", jolloin CIrcEngine voi vain kutsua tarvittavan komennon tuottavaa metodia sopivilla parametreilla ja CIrc tuottaa valmiin viestin joka voidaan välittää sellaisenaan CSocketEnginelle. Vastausviestien kokoamisen toteutus on huomattavasti suoraviivaisempi.

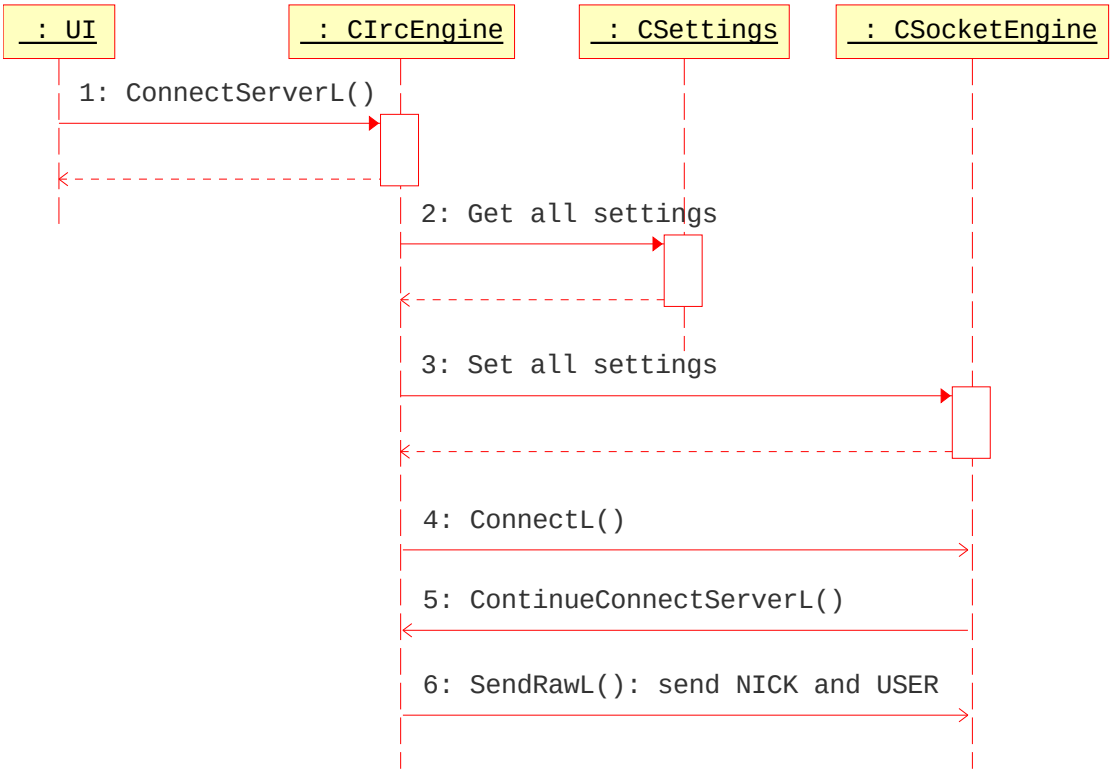
```
void CIrc::TopicString(TDes8& aString,
                       TDesC8& aChannel,
                       TDesC8& aNewTopic)
{
    aString.Append( iResponseTypeStr[cmd_pos(CMD_TOPIC)] );
    aString.Append(_L8(" "));
    aString.Append(aChannel);
    aString.Append(_L8(" :"));
    aString.Append(aNewTopic);
}
```

Käytännössä "valmistusohjeet"ovat vain metodeita, jotka järjestelevät olemassaolevia merkkijonoja yhdeksi pitkäksi merkkijonoksi.

5.3 Verkkoyhteys

CSocketEngine toteuttaa moottorin tietoliikennetoiminnot. Kaikki data, joka siirretään verkon yli, tapahtuu SocketEnginessä. Luokan toteutuksessa on käytetty aktiiviolioita ja Socket Serveriä. TCP/IP-protokollaa käytetään asiakas-palvelin-mallin mukaisesti. Symbian-järjestelmässä socket-palvelin tarjoaa geneerisen pääsyn sockettien käyttöön. Palvelin on riippumaton siirtoprotokollasta, jolloin varsinainen tiedonsiirto voi tapahtua esimerkiksi GPRS- tai Bluetooth-yhteyden yli. Käytännössä

käyttöliittymä kysyy käyttäjältä, mitä tiedonsiirtoväylää halutaan käyttää socket-yhteyttä muodostettaessa.



Kuva 5.3: Yhteyden muodostus palvelimelle CSocketEngineä käyttäen

Kuvasta 5.3 voimme nähdä, kuinka yhteys palvelimelle muodostetaan.

1. Käyttöliittymästä kutsutaan CIrcEnginen yhteyden muodostamismetodia, *ConnectServerL()*.

2. CIrcEngine pyytää CSettings-oliolta tarvittavat asetukset, palvelimen osoitteen ja portin.

3. Haetut asetukset välitetään CSocketEngine-oliolle kutsumalla metodeja SetServerName() ja SetPort().

4. Kutsutaan asynkronista funktiota ConnectL(), joka selvittää palvelimen IP-osoitteen ja käynnistää aktiiviolion. Aktiiviolion käynnistyttyä ensimmäiseksi aloitetaan yhteyden muodostus palvelimelle.

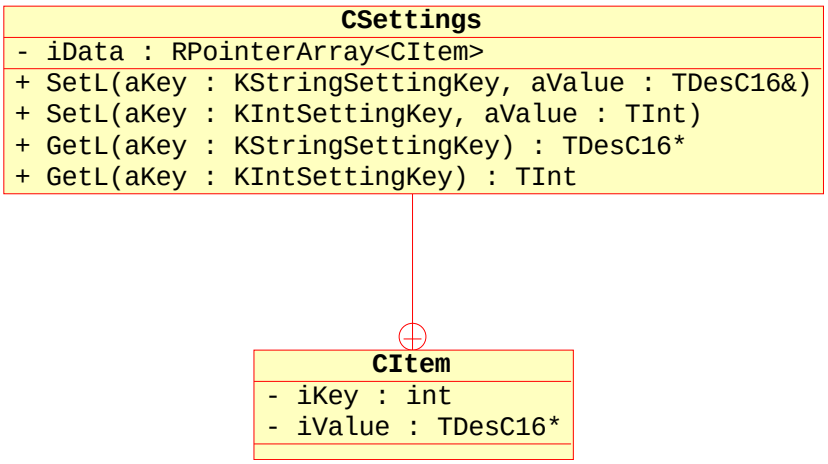
5. Kun yhteys on muodostunut, kutsutaan CIrcEnginen ContinueConnectServerL()-funktiota, joka hoitaa IRC-palvelimelle kirjautumiseen tarvittavien viestien muodostuksen.

6. Kirjautumisviestit lähetetään palvelimelle. Tämän jälkeen (mikäli palvelin hyväksyy kirjautumisen) on yhteys valmis.

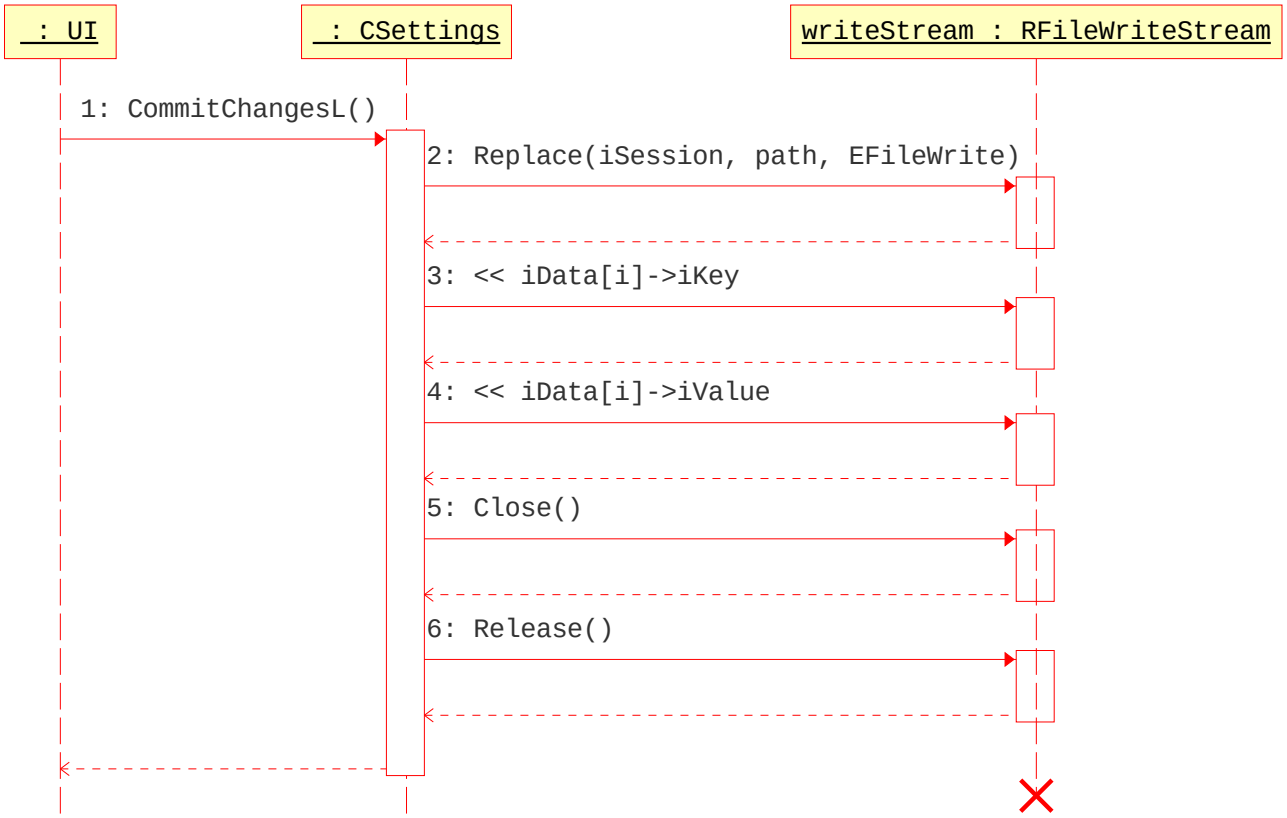
5.4 Asetusten hallinta

Tiedon tallennus ja hakeminen uudestaan myöhemmin on hyvin olennaista useimmissa sovelluksissa. Symbian OS:sta löytyykin useita erilaisia tapoja säilöä tietoa. Bittivirtoja (Stream) ja säiliöitä (Store) voidaan käyttää tiedon tallentamiseen. Luku- ja kirjoitusvirroilla voidaan tallentaa ja lukea dataa. Säiliö on kokoelma datavirtoja, jossa jokaisella virralla on oma yksilöllinen tunnus (ID). Tiedostoja tuetaan RFile APIn kautta, joka mahdollistaa tiedostojen luonnin ja avaamisen sekä binäärideskriptoreiden lukemisen sekä tallentamisen. Tämän lisäksi virtoja voi ulkoistaa tiedostoon. Streaming APIn käyttö datan tallentamiseen ja lukemiseen on usein helpointa. [6]

CSettings toimii tiedostopalvelimen kanssa. Se varastoi ohjelman tarvitsemat parametrit pitkäaikaiseen säilytykseen.



Kuva 5.4: CSettings-luokka varastoi asetukset CItem-olioihin

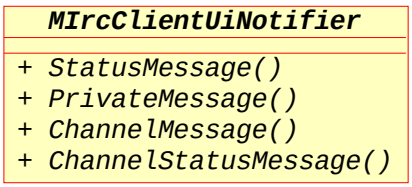


Kuva 5.5: *RFileWriteStream*in käyttö asetusten tallennuksessa

1. Käyttöliittymän puolelta pyydetään CSettings-oliota tallentamaan muistissa olevat asetukset.
2. Luodaan RFileWriteStream-olio.
3. Tallennetaan asetukset avain-arvo-pareina writeStream-virtaan, ensin avain
4. ja heti perään avainta vastaava arvo.
5. Suljetaan tiedosto.
6. Vapautetaan kahva tiedostopalvelimeen.

5.5 Yhteys käyttöliittymään

Koska CIrcEnginen toimintaperiaate on asynkroninen, säilytetään sama asynkroninen luonne myös sovelluksen graafisen osan toteutuksessa ja käyttäytymisessä. MIrcClientUiNotifier on CIrcEnginen tapa keskustella käyttöliittymälle päin. Jotta moottoria voidaan käyttää, tulee käyttöliittymän puolella toteuttaa puhtaasti virtuaalinen luokka MIrcClientUiNotifier. Kuva 5.6.



Kuva 5.6: Puhtaasti virtuaalisen MIrcClientUiNotifier-luokan kuvaus

StatusMessage() Kutsutaan, kun palvelimelta tuleva viesti sisältää jotain yleistä yhteyteen liittyvää.

PrivateMessage() Yksityinen viesti käyttäjälle (yksi-yhdelle-viesti, kappale 4).

ChannelMessage() Yleinen viesti käyttäjäryhmälle (yksi-monelle-viesti, kappale 4).

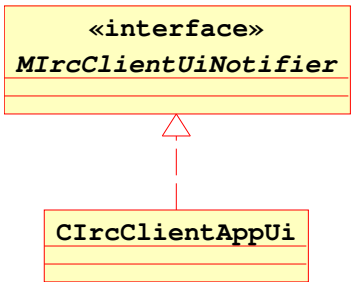
ChannelStatusMessage() Tietyn kanavan toimintaan liittyviä viestejä, esimerkiksi käyttäjän liittyminen kanavalle tai kanavan otsikon muutos.

6. GRAAFINEN KÄYTTÖLIITTYMÄ

Sovellus pyrittiin toteuttamaan niin, että moottori ja käyttöliittymä olisivat selvästi erillisiä komponentteja. Silloin käyttöliittymä voitaisiin tarvittaessa toteuttaa helposti mille tahansa sovelluskehykselle. Mobiililaitteessa toimivan käyttöliittymän suunnittelu on erityisen haasteellista ja sen vuoksi myös tärkeää. Hallintalaitteet ovat rajalliset ja yleensä pienet ja ne on sijoitettu ahtaasti. Käyttöliittymä on parhaimmillaan, kun sitä käytettäessä ei tarvitse etsiä toimintoja tai tutkiä käyttöohjekirjaa toiminnon suorittamiseksi, vaan ainakin yleisimmät toiminnot löytyvät loogisesti käyttäjän olettamista paikoista.

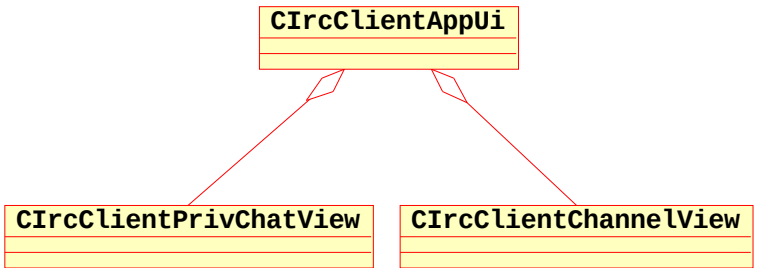
6.1 Käyttöliittymän luokkatason rakenne

Graafisessa käyttöliittymässä toteutettiin virtuaalinen luokka MIrcEngineNotifier (kuva 6.1), joka annetaan moottorille alustuksen yhteydessä.



Kuva 6.1: Käyttöliittymäluokka toteuttaa vahtirajapinnan

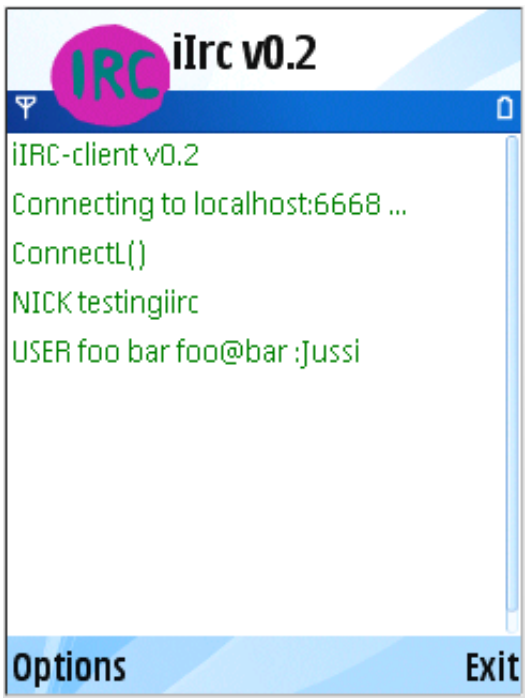
CIrcClientAppUi-luokka sisältää kaikki näkymät kanaviin ja yksityiskeskusteluihin (kuva 6.2).



Kuva 6.2: CIrcClientAppUi sisältää näkymät

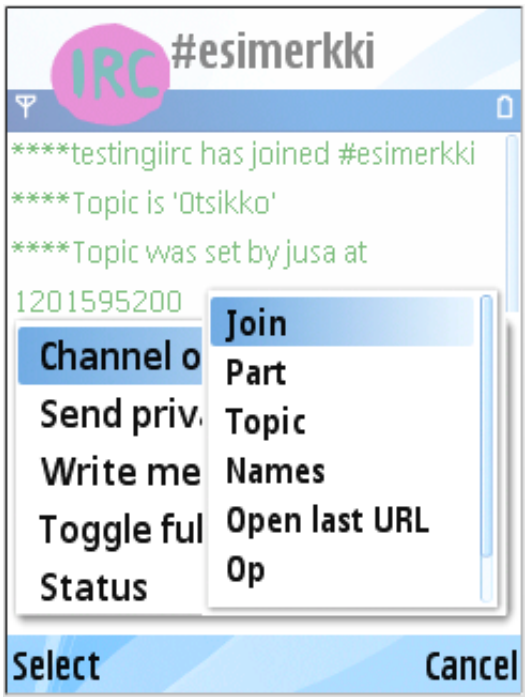
6.2 Ulkoasu ja käytettävyys

Sovelluksen yleisilme näkyy kuvassa 6.3, jossa sovellus on avaamassa yhteyttä IRC-palvelimelle.



Kuva 6.3: iIrc muodostaa yhteyttä palvelimelle

Koska puhelimen tekstin syöttö- ja hallintalaitteet ovat melko rajoittuneet ja sovelluksen käytettävä pinta-ala on rajallinen normaaliin pöytäkoneeseen verrattuna, on käyttöliittymän oltava mahdollisimman intuitiivinen käyttää saatavilla olevilla hallintalaitteilla. Suurin osa sovelluksen toiminnoista löytyy valikon takaa, mutta tärkeimmät, eli tekstin syöttö ja kanavan vaihtaminen on pyritty tekemään mahdollisimman helpoiksi. Sovelluksen valikkorakenne keskustelunäkymässä näkyy kuvassa 6.4.



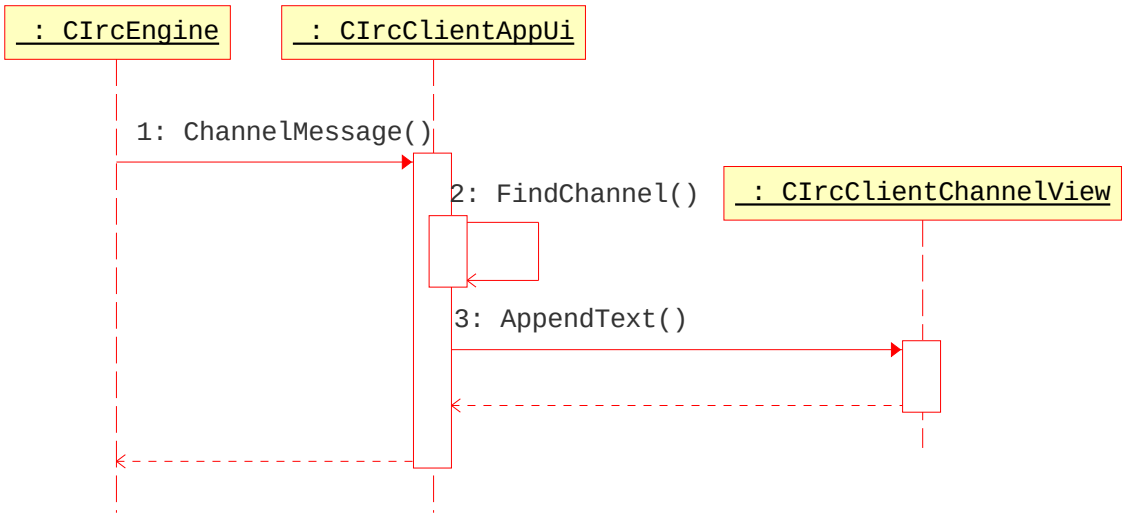
Kuva 6.4: Sovelluksen valikkorakenne keskustelunäkymässä

Helppokäyttöisyyden kannalta eräs merkittävä ongelma oli tekstin syötön kohtuuton vaiva, kun joka kerta viestiä kirjoittaakseen piti valita toiminto valikon kautta. Ongelma ratkaistiin toteuttamalla oma versio sovelluksen näppäintapahtumat käsittelevälle metodille. Sovelluksen pääluokka perii CAknViewAppUi-luokan. Oletuksena näppäintapahtumat käsitellään edellä mainitun luokan *HandleKeyEventL()*-metodissa, joka on private-tyyppinen. Kun metodi ylikirjoitettiin omalla toteutuksella, on mahdollista kaapata kaikki perusaakkosia tuottavat näppäinkoodit ja avata viestin kirjoitusdialogi. Samalla voidaan myös nuolinäppäimet vasemmalle ja oikealle valjastaa kanavanäkymän vaihtamiselle. Näinkin yksinkertainen muutos perustoiminnallisuuteen helpottaa huomattavasti tekstin syöttöä ja tekee sovelluksen käyttämisestä suoraviivaisempaa. Vähemmän tarvittavat toiminnallisuudet voidaan jättää valikoiden kautta käytettäviksi. Tekstin syöttödialogi on kuvassa 6.5.

Kuva 6.5: Sovellus on kaapannut näppäinkomennon ja avannut tekstinsyöttödialogin

6.3 Asynkronisen kirjaston käyttö

IRC-protokolla ja IrcEngine ovat luonteeltaan asynkronisia. Tämän vuoksi myös käyttöliittymän pitää toimia tämän luonteen mukaisesti. Kun käyttöliittymästä pyydetään jollekin kanavalle liittymistä, ei asiakas voi mitenkään tietää, onko pyyntöön vastattu tai onko se edes mennyt perille. Sen vuoksi ei myöskään kanavanäkymää kannata avata ennen kuin moottori kertoo kanavan luomisesta. Tämän vuoksi kaikki kanavanäkymät luodaan vasta kun moottorilta tulee viesti kanavalle, jota ei ole olemassa, samoin kanavanäkymä suljetaan, kun moottori välittää kanavalta poistumis -viestin käyttöliittymälle. Viestien välittämiseen käytetään käyttöliittymän perimän *MIrcEngineNotifier*-luokan metodeja. Tyypillinen näkymän luomisen tapahtumaketju on kuvattu kuvassa 6.6.



Kuva 6.6: Kanavanäkymän luominen

1. Kun moottorille on tullut kanavaviesti, se välitetään käyttöliittymälle *ChannelMessage()*-metodilla.

2. Käyttöliittymä menee luodut kanavanäkymät läpi ja etsii viestille sopivaa näkymää. Kun olemassa olevaa näkymää ei löydy, luodaan uusi näkymä ja lisätään se käyttöliittymän kanavanäkymälistaan.

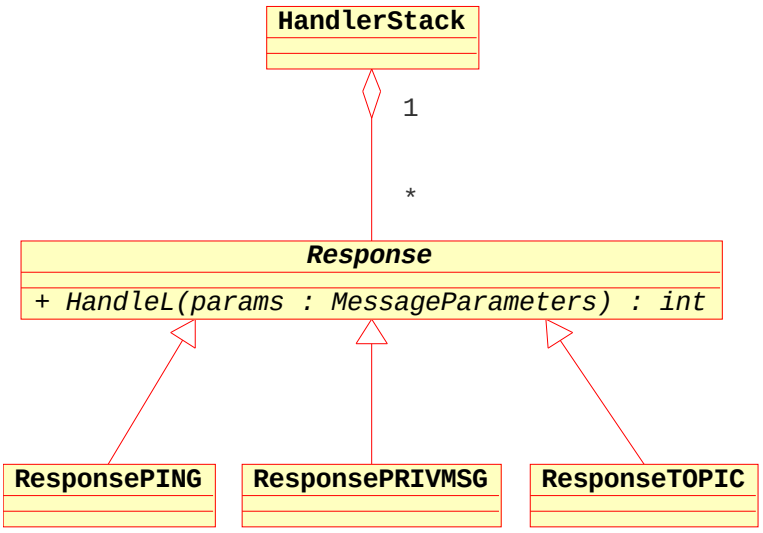
3. *FindChannel()* palauttaa osoittimen näkymään, jonka *AppendText()*-metodia kutsutaan ja välitetään kanavaviestin sisältämä data.

7. POHDINTA

Työn kulkua ja tuloksia on aina hyvä pohtia paitsi työtä tehdessä, myös sen jälkeen. Vaikka toteutuksen suunnittelisi kuinka hyvin, vasta työn jälkeen näkee kaikki ne mahdollisuudet ja yksityiskohdat, joita ei aluksi osannut ottaa huomioon. Sovelluksen toteuttamiselle varatun ajan puitteissa ei myöskään voi tehdä kaikkea, jolloin jää toteuttamatta paljon hyödyllisiä ominaisuuksia.

7.1 Toteutusvaihtoehdot

Toteutunut implementaatio on viestien käsittelyn kannalta melko kaukana olio-ohjelmoinnista. Käytännössä esimerkiksi CIrc-luokka on vain luokka täynnä staattisia metodeja merkkijonojen muokkaamiseen. Yksi mahdollisuus olisi ollut toteuttaa viestien käsittely käyttäen olioita.



Kuva 7.1: Oliokäsittelijöiden luokkakaavio

Kuvassa 7.1 näkyy vaihtoehtoisen toteutuksen luokkajako. Käsittely olisi voitu toteuttaa myös niin, että olisi ollut yksi abstrakti kantaluokka, esimerkiksi *Response*, joka saisi rakennuksen yhteydessään parametrinä esimerkiksi CIrcEnginen osoittimen. Kantaluokalla olisi kaksi abstraktia metodia *HandleL()* ja *GetType()*. Response-luokan toteuttavat käsittelijät olisivat jossain säiliössä, esimerkisi RArrayssä. Kun uusi viesti saapuisi CIrcEngineen, iteroitaisiin säiliö läpi kunnes jokin

käsittelijäolio suostuisi käsittelemään viestin. Käsiteltyään viestin asianomaisesti, käsittelijä kutsuisi joko käyttöliittymän metodeita tai lähettäisi dataa SocketEnginelle.

7.2 Jatkokehitysajatukset

Vaikka perustoiminnallisuuden toteuttava sovellus saatiinkin valmiiksi, on siinä vielä erittäin paljon kehittämismahdollisuuksia. Yksi olennainen liian vähälle huomiolle jäänyt ominaisuus sovelluksen moottorissa on virheiden käsittely. Esimerkiksi verkkoyhteyden katkeaminen kesken yhteyden saa sovelluksen tilaan, jossa se joko pysähtyy tai kaatuu. Myöskään yhteyden muodostaminen ei onnistu, mikäli ensimmäinen yritys epäonnistuu. Sovelluksen tilaa ei palauteta kunnolla.

Toteutunut parseri kaipaisi myös lisäkehitystä. Tällä hetkellä se on hyvin yksinkertainen ja herkkä epästandardille viestin muotoilulle. Parserin toteutuksen voisi tehdä kokonaan uudestaan, ja olisi hyvä selvittää perusteellisesti, kuinka Symbianissa voidaan käyttää säännöllisiä lauseita (regular expression). Säännöllisten lauseiden avulla viestin parsiminen olisi huomattavasti yksinkertaisempi operaatio. Protokollan toteutus on myös hyvin puutteellinen. Sovellus kaipaisi vielä paljon lisää toiminnallisuutta jotta siitä saisi paremmin käytettävän. Myös SSL-yhteyden (Secure Socket Layer) käyttömahdollisuutta voisi olla hyvä tutkia. Suojatun yhteyden avulla olisi mahdollista käyttää sovellusta myös suljettujen salattujen verkkojen kanssa turvallisesti.

Käyttöliittymän toteutus on myös hyvin yksinkertainen ja ominaisuuksiltaan vähäinen. Yksi näkyvimpiä puutteita on keskustelunäkymien tekstin muotoilun ja värityksen puute. Sovellukselle saisikin paljon lisää sekä käytettävyyttä että ulkonäköä kunnollisen värillisen tekstin muotoilun avulla. Käyttöliittymän valikkorakenteessa sekä asetuksien muokkaamisessa on myös paljon epäloogisuuksia ja vähälle jäänyttä käytettävyyssuunnittelua, joita miettimällä ja kehittämällä käyttökokemuksesta saisi sujuvamman.

8. YHTEENVETO JA JOHTOPÄÄTÖKSET

Työn toteutusosa valmistui pääosin aikataulun mukaisesti. Sovellus sisältää perustoiminnallisuuden, jotta sillä voi yhdistää ja keskustella IRC-verkon välityksellä. Suurimmat haasteet Symbian OS -ympäristössä olivat aktiiviolioiden ja deskriptoreiden kanssa. Symbianissa verkkoliikenteen data välitetään kahdeksanbittisissä deskriptoreissa, kun käyttöliittymäkomponenteissa käytetään kuusitoistabittisiä deskriptoreita. Varsinkin alkuvaiheessa konversioiden ja tekstin välitys deskriptoreiden kanssa tuotti ongelmia. Hieman vääränlainen deskriptoreiden käyttö johti yllättäviin ongelmiin ja virheisiin sovelluksessa. Esimerkiksi sovelluksen ensimmäinen kokeilu todellisessa laitteessa johti siihen, ettei ohjelma lähtenyt käyntiin vaan kaatui heti käynnistyksen yhteydessä. Syyksi paljastui liian suuri pinomuistin käyttö, jolloin se loppui melkein heti ohjelman käynnistyttyä.

Sovellus onnistui hyvin. Erityisesti hieman alkuperäisen toteutuksen jälkeen uudelleen kirjoitettu asetuksien tallennukseen käytetty luokka CSettings osoitti, että työtä tehdessä on kertynyt paljon tietoa ja taitoa. Nyt olisi hyvä kirjoittaa sovellus uudestaan puhtaalta pöydältä, kun ongelmakohdat ratkaisuvaihtoehtoineen ovat tiedossa. Toteutunut rakenne on kohtuullisen helposti laajennettavissa uusilla protokollan käskyjen käsittelijöillä.

Toteutuksen lopullinen muoto ei ole kovin olio-ohjelmointimallin mukainen, mutta on toimiva ja helpohkosti laajennettavissa oleva. Tietotyypit valittiin sopivasti, jotta ohjelmaan on kohtuullisen helppo lisätä uusia käsittelijöitä häiritsemättä vanhaa toteutusta. Oliomallisella toteutuksella ei kuitenkaan välttämättä saavuteta merkittäviä etuja nyt tehtyyn malliin nähden. Toteutuksen edetessä harkittiin myös oliomaisempaa toteutusta. Sitä ei kuitenkaan lähdetty tekemään, mutta dokumentoitiin lyhyesti vaihtoehtoiseksi toteutustavaksi. Uuden ohjelmointiympäristön ja erilaisten käytäntöjen vuoksi toteutus osoittautui yllättävän haastavaksi, mutta järjestelmällisen tiedon etsimisen ja hyvän ohjauksen ansiosta haasteista selvittiin.

Kaiken kaikkiaan sovelluksen suunnittelu ja toteutus sujui hyvin ja aikataulun mukaan. Työn aikana kertyi hyvä perusosaaminen Symbian OS -ympäristöstä ja ohjelmointimalleista. Kuten aina uutta asiaa tehtäessä ja tutkittaessa, vastaan tuli yllättäviä ja haasteellisia tekijöitä. Haasteet kuitenkin ylitettiin ja ideoita ja ajatuksia toteutuksen parantamiseksi löytyi huomattavasti.

LÄHTEET

[1] Niskanen, Pekka. 2005. Symbian-ohjelmointi: tehokas hallinta, nn. painos. Helsinki, Readme.fi. 216 s.

[2] Mutton, Paul. 2004. Irc hacks, 1. painos. Sebastopol, CA: O'Reilly. 409 s.

[3] Kaario, Kimmo. 2002. TCP/IP-verkot, 1. painos. Jyväskylä, Docendo. 396 s.

[4] Pyssysalo, Tino, et al. 2003. Programming for the series 60 platform and Symbian OS, 1. painos. Helsinki, Digia. 521 s.

[5] Haikala, Ilkka, Märijärvi, Jukka. 2004. Ohjelmistotuotanto, nn. painos. Helsinki, Talentum. 440 s.

[6] Edwards, L., Barker, R., et al. 2004. Developing Series 60 Applications: A Guide for Symbian OS C++ Developers, 1. painos. USA, Addison-Wesley. 749 s.

[7] Koskimies, Kai, Mikkonen, Tommi. 2005. Ohjelmistoarkkitehtuurit, 1. painos. Helsinki, Talentum. 250 s.

[8] Symbian OS 9.3 and S60 3.2 Architectures

[9] Symbian OS Overview

[10] Symbian OS Essentials

[11] Wikipedia, http://en.wikipedia.org/wiki/Internet_Relay_Chat, lainattu 29.8.2007

[12] Oikarinen, J., Reed, D. 1993. RFC 1459: Internet Relay Chat Protocol <http://www.ietf.org/rfc/rfc1459.txt>

[13] Kalt, C. 2000. RFC 2810: Internet Relay Chat: Architecture <http://www.ietf.org/rfc/rfc2810.txt>

[14] Kalt, C. 2000. RFC 2812: Internet Relay Chat: Client Protocol <http://www.ietf.org/rfc/rfc2812.txt>

[15] Beej's Guide to Network Programming http://beej.us/guide/bgnet/output/html/singlepag
lainattu 23.4.2008

A. LIITE: TELNET-ISTUNTO IRC-PALVELIMEN KANSSA

Esimerkissä on käyttäjän antamat syötteet merkattu rivin alussa olevalla » -merkillä.

```
testaaja@testikone ~ $ telnet irc.jyu.fi 6667
Trying 130.234.6.250...
Connected to irc.jyu.fi.
Escape character is '^]'.
:irc.jyu.fi 020 * :Please wait while we process your connection.
>> NICK esi-merkki
>> USER foo bar foo@bar :esi-merkki
:irc.jyu.fi 001 esi-merkki :Welcome to the Internet Relay Network
esi-merkki!-foo@malli-ip-osoite.fi
:irc.jyu.fi 002 esi-merkki :Your host is irc.jyu.fi, running
version 2.11.1p1
:irc.jyu.fi 003 esi-merkki :This server was created Sun Dec 17
2006 at 23:04:49 EET
:irc.jyu.fi 004 esi-merkki irc.jyu.fi 2.11.1p1 aoOirw
abeiIklmnoOpqrRstv
:irc.jyu.fi 005 esi-merkki RFC2812 PREFIX=(ov)@+ CHANTYPES=#&!+
MODES=3 CHANLIMIT=#&!+:21 NICKLEN=15 TOPICLEN=160 KICKLEN=160
MAXLIST=beIR:42 CHANNELLEN=50 IDCHAN=!:5
CHANMODES=beIR,k,l,imnpstaqr :are supported by this server
:irc.jyu.fi 005 esi-merkki PENALTY FNC EXCEPTS=e INVEX=I
CASEMAPPING=ascii NETWORK=IRCnet :are supported by this server
:irc.jyu.fi 042 esi-merkki 246DAAH8S :your unique ID
:irc.jyu.fi 251 esi-merkki :There are 87959 users and 6 services
on 44 servers
:irc.jyu.fi 252 esi-merkki 167 :operators online
:irc.jyu.fi 253 esi-merkki 1 :unknown connections
:irc.jyu.fi 254 esi-merkki 46245 :channels formed
:irc.jyu.fi 255 esi-merkki :I have 961 users, 0 services and 1 servers
:irc.jyu.fi 265 esi-merkki 961 1017 :Current local users 961, max 1017
:irc.jyu.fi 266 esi-merkki 87959 100987 :Current global users 87959,
```

```
max 100987
:irc.jyu.fi 375 esi-merkki :- irc.jyu.fi Message of the Day -
:irc.jyu.fi 372 esi-merkki :- 14/12/2006 19:21

[päivän viesti leikattu pois]

:irc.jyu.fi 376 esi-merkki :End of MOTD command.
:irc.jyu.fi 484 esi-merkki :Your connection is restricted!
:esi-merkki MODE esi-merkki :+r
>> JOIN :#kanava
:esi-merkki!-foo@malli-ip-osoite.fi JOIN :#kanava
:irc.jyu.fi 353 esi-merkki = #kanava :esi-merkki @Testaaja
:irc.jyu.fi 366 esi-merkki #kanava :End of NAMES list.
:Testaaja!~testaaja@malli-ip-osoite.fi JOIN :#kanava
:Testaaja!~testaaja@malli-ip-osoite.fi PRIVMSG #kanava :Hei!
>> PRIVMSG #kanava :Tervehdys vain!
:Testaaja!~testaaja@malli-ip-osoite.fi PRIVMSG #kanava :Eiköhän tämä
riitä esimerkistä?
>> PRIVMSG #kanava :Toki... Näkemiin!
>> PART :#kanava
:esi-merkki!-foo@malli-ip-osoite.fi PART #kanava :
>> QUIT
ERROR :Closing Link: esi-merkki[-foo@malli-ip-osoite.fi] ("")
Connection closed by foreign host.
```