AS-0.1102 – projektisuunnitelma

JTTBot – IRC bot

Tekijät:

78634P Juha Keponen jkeponen@cc.hut.fi
65088R Tapio Partti tnolvi@cc.hut.fi
66709A Timo Romppanen taromppa@cc.hut.fi

Päivitetty
6.11.2008

Vaatimukset

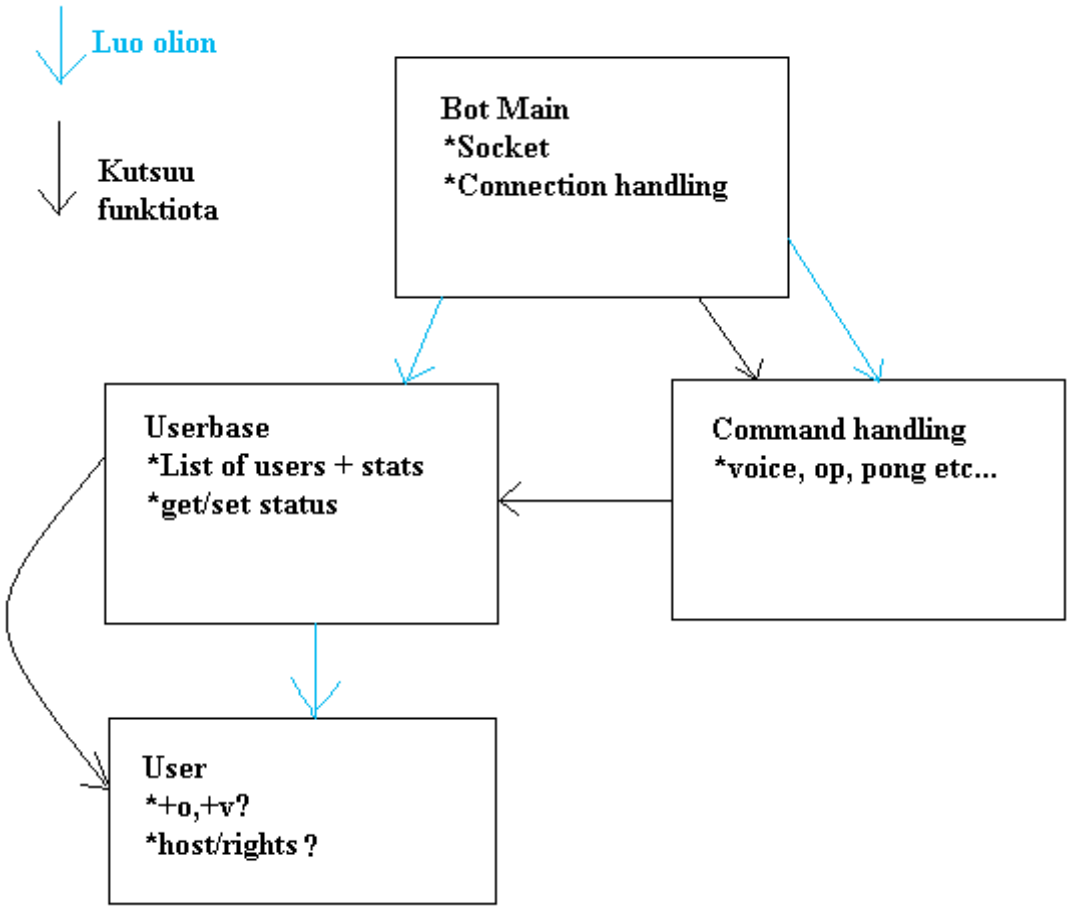
Vähimmäisvaatimuksina botin tulee pystyä yhdistymään RFC 1459 standardin mukaiselle irc-serverille sekä myös uudelleen yhdistymään mikäli yhteys katkeaa. Lisäksi botin täytyy tunnistaa yksittäiset käyttäjät ja osata toteuttaa yksinkertaisia komentoja, kuten antamaan operaattorin oikeudet kanavalle.

Lisäominaisuuksina botille voi ohjelmoida enemmän komentoja, tuen useammalle kanavalle sekä SOCKS4/5 palomuurille, mutta lähtökohtana on vähimmäisvaatimusten täyttäminen. Lisäkomentoina voisi olla muun muassa topicin vaihtaminen ja siihen lisääminen sekä esimerkiksi muistutuksen laittaminen. On myös mahdollista, jos into ja aika riittää, että toteutamme botille jonkinlaisen tekstitiedostoon perustuvan tietokannan käyttäjistä, joilla on oikeus saada opit tai voice. Käyttäjien tunnistautuminen perustuisi luultavasti hostiin ja se osaisi antaa oikeutetuille käyttäjille oikeudet heidän liittyessään kanavalle. Monikanavatuki sekä käyttäjien oikeuksien tallentaminen ovat tärkeysjärjestyksessä ennen muita.

Ohjelman arkkitehtuuri

Ircbottimme on suunniteltu minimivaatimusten perusteella ja se koostuu neljästä kuvassa luokasta (kuva 1): BotMain, CommandHandling, UserBase ja User. Pääohjelma luo käynnistyessään CommandHandling luokan, joka huolehtii annettujen komentojen käsittelystä eli käskee UserBase-luokkaa päivittämään tai antamaan käyttäjätietoja. Lisäksi se rakentaa IRC-palvelimelle lähetettävän tekstimuotoisen komennon ja kutsuu BotMain-luokkaa lähettämään sen. Ohjelman käynnistyessä ja yhteyden katketessa BotMain-luokassa rakennetaan TCP(socket) ja IRC-protokollan mukainen -yhteys komentoriviparametrina saatuun IRC-palvelimeen. Jos yhteys saadaan, luodaan UserBase-olio ja kutsutaan CommandHandling-luokan joinChannel -funktiota, joka lisää botin komentoriviparametrina saadulle kanavalle. Kun botti liittyy kanavalle, se pyytää IRC-palvelimelta listan käyttäjistä ja niiden tilasta sekä lisää nämä UserBase-luokassa olevaan tietorakenteeseen. Yksittäisen käyttäjän tiedon - mukaan lukien muun muassa käyttäjän host - säilötään UserBase-olioon. Jos toteutamme bottiin monikanavatuen, tulee UserBase-luokkaan useampia käyttäjälistoja, jolloin yksi lista edustaa yhtä kanavaa.
Kun botti saa edellä mainitut toiminnot suoritettua, se jää looppiin kuuntelemaan sockettia. Kaikki viestit parsitaan BotMain-luokassa, mutta komennot suoritetaan CommandHandling-luokan avulla. Tällä tavalla botti vastaa niin serveriltä tuleviin ping-viesteihin kuin botin käyttäjien pyyntöihin

saada esimerkiksi opit. Jos emme toteuta edellisessä luvussa mainittua käyttäjien oikeuksia säilövää tietorakennetta, täytyy käyttäjien tietää niin oppien kuin voicejen saamiseksi salasanat, jotka määritellään botille sen käynnistyksen yhteydessä. UserBase-luokka huolehtii kaikista käyttäjien tilaan liittyvistä asioista ja kaikki siihen tarkoitukseen tulevat funktiot ottavat siis parametrina vähintäänkin käyttäjän hostin. Käyttäjien tunnistus tapahtuu siis hostien perusteella, sillä nick:iin perustuva tunnistautuminen on huono ajatus sen helpon vaihdettavuuden takia.



Kuva 1: Luokkakaavio IrcBotista

BotMain-luokka sisältää main-funktion lisäksi funktiot ainakin komentojen parsimiseksi ja komentojenrivien lähettämiseksi palvelimelle. CommandHandlin-luokka sisältää alustavasti ainakin funktiot oppien ja voicejen antamiseen sekä poisottamiseen, ping viestiin vastaamiseen sekä kanavalta potkimiseen. Userbase sisältää ainakin seuraavat itsensä selittävät funktiot: addUser, delUser, addVoice, addOp, delVoice, delOp. Se, tarvitsemmeko esimerkiksi op- tai voice-statuksen kertovia funktioita riippuu siitä, tarvitsemmeko niitä loppujen lopuksi vai hoidetaanko esimerkiksi käyttäjien olemattomuuden seuraukset poikkeuksilla.

Käytämme Boost.ASIO kirjastoa soketin luontiin, ja näin ollen vältämme wrapper-luokan tekemisen. UserBase-luokassa User-oliot säilötään lista tietorakenteeseen, koska käyttäjiä joudutaan poistamaan myös keskeltä listaa niiden poistuessa kanavalta. Pidämme siis listaa tähän tarkoitukseen sopivimpana ja yksinkertaisimpana.

Tehtävien jako

Pyrimme jakamaan tehtävät kaikenkaikkiaan tasaisesti ottaen huomioon myös tämän suunnitelman ja loppuraportin. Juha ja Tapio tekivät suunnitelman suurimmalta osin, joten on oletettavaa, että Timo tekee enemmän loppuraporttia. Tämä myös siksi, että ohjelmointiosuuskin jakautuisi kohtalaisen tasaisesti. Itse ohjelman kirjoittamisen on ajateltu jakautuvan seuraavasti: Juha tekee pääohjelman, Tapio command handling -luokan ja Timo userbase ja user -luokat. Tarvittaessa autamme toki toisiamme ja jos jonkin luokan tekeminen osoittautuu odotettua työläämmäksi, voimme tehdä sitä yhdessäkin. Ennakolta arvioisimme, että BotMain-luokka on hankalinta toteuttaa ja niinpä muutkin kuin Juha voi tehdä siitä esimerkiksi komentojen parsimiseen liittyvää osuutta. Testaus jakautuu pääosin sen mukaan kuka mitäkin koodia kirjoittaa. Ohjelman eri osien yhteistyön testaus tapahtuu yhdessä tai ainakin kyseessä olevien luokkien tehneiden henkilöiden toimesta ja lopullinen toimintatestaus tehdään yhdessä. Ryhmän jäsenten henkilökohtaiseksi ajankäytöksi arvioidaan noin 50 tuntia.

Ohjelman testaus

Kuten edellisessä luvussa mainittiin, jokainen testaa omaa tuotostaan itsenäisesti ja kysyy tarpeen vaatiessa apua. Käyttäjätietohallintaa voi testata esimerkiksi tulostamalla käyttälistaa, kun sen näkee tarpeelliseksi sekä tulostamalla funktioiden paluuarvoja. Näin voidaan katsoa onko käyttäjien tiedot niinkuin niiden pitäisi. Apuna voi käyttää myös jotain yksinkertaista main-luokkaa, joka vain kutsuu UserBase -luokan funktioita sopivilla keinotekoisilla parametreilla.

Pääohjelmaa voi testata jokaisen pienen askeleen jälkeen. Jotta saadaan selville muun muassa onko ohjelmamme saanut TCP-yhteyden IRC-palvelimeen, vastaako se ping-viesteihin tai ovatko luomamme paketit oikein rakennettuja, kannattaa pakettiliikennettä tarkkailla wireshark-ohjelman avulla. Näin voimme varmistua, että ohjelmamme keskustelee halutulla tavalla IRC-palvelimen kanssa jo siinä vaiheessa kun bottimme ei ole edes IRC-kanavalla. Botin olemassaoloa ja käyttäytymistä IRC-kanavalla sekä valmiimman botin reaktioita komentoihin kannattaa tarkkailla ja kokeilla yksinkertaisesti liittymällä samalle kanavalle, jollain IRC asiakasohjelmalla. Koska botin pitää yrittää yhdistää IRC-palvelimelle uudelleen, jos yhteys siihen katkeaa, täytyy meidän saada yhteys myös katkeamaan. Tämä voidaan tehdä joko ohjelma-tasolla jättämättä vastaamatta ping-viesteihin tai rikkomalla socket. Yksi testausvaihtoehto tälle voisi olla myös palomuurin asettaminen estämään bottimme käyttämän portin liikenne.

CommandHandling luokan testaus on hieman hankalampaa etenkin ilman UserBase ja User -luokkia, mutta IRC-komentojen luomista voinee helposti kokeilla vertailemalla sen rakentamia komentorivejä joihinkin internetistä löytyviin samaa tekeviin esimerkkeihin. Komentojen oikeellisuutta voidaan testata myös syöttämällä niitä käsipelillä telnetin kautta. Koska BotMain-luokan tekemisessä menee luultavasti pisimpään, lienee järkevää testata muiden luokkien toimintaa ensin keskenään. CommandHandling ja UserBase -luokkien yhteistoimintaa voidaan testata helposti jälleen kerran luomalla jokin yksinkertainen main-luokka, joka kutsuu CommandHandling -luokan funktioita sopivilla parametreilla. Tämän jälkeen voidaan jälleen katsoa onko UserBase ja User olioiden tila halutunlainen.

Keskeneräisen ohjelman testauksessa keskitytään osatoimintojen testaukseen aloittaen pienemmistä kokonaisuuksista ja jatkamalla aina vain suuremmilla sitä mukaa kun koodia valmistuu. Lopullisen ohjelman testauksessa on yritettävä ottaa huomioon kaikki mahdolliset tilanteet, joihin botti voi joutua. Näitä ovat muun muassa eri komentojen antaminen botille, kun botti ja sen käyttäjät ovat eri ”tilassa” tai komentojen antaminen sopivasti väärillä parametreilla.

Projektin aikataulu

Koska projektin aikataulu on tiukka, yritämme saada erilliset osat toimimaan ja testattua erikseen vajaassa 3 viikossa, jonka jälkeen käytämme noin viikon osien yhteenliittämiseen ja yhteistoiminnan testaamiseen. Viimeistelytestaukseen ja botin korjaamiseen käytetään toivottavasti vähemmän kuin viikko. Samaan aikaan kuin ohjelmaa viimeistellään, voi ainakin yksi ryhmämme jäsen ryhtyä tekemään loppuraporttia. Jos asiat sujuvat nopeammin kuin on suunniteltu, saatamme lopuksi lisäillä bottiin ensimmäisessä osassa mainittuja ominaisuuksia. Tärkein tavoitteemme on kuitenkin saada botti toimivaksi ennen joulukuun puoliväliä.

Viittaukset

Aiomme käyttää Boost.ASIO kirjastoa tietoliikenneyhteyden muodostamiseen eli socketin rakentamiseen (http://www.boost.org/doc/libs/1_36_0/doc/html/boost_asio.html, viitattu 6.11.2008). Muutoin käytämme C++:n natiivi kirjastoja.

Testauksen apuna käytämme WireShark-ohjelmaa, http://www.wireshark.org/, viitattu 6.11.2008.

Tietoviitteinä käytämme Boost.ASIO kirjaston dokumenttien lisäksi ainakin seuraavia lähteitä:

J. Oikarinen, D.Reed, Internet Relay Chat Protocol, 1993, RFC 1459, http://www.faqs.org/rfcs/rfc1459.html, viitattu 6.11.2008.