Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Vesa Ruohonen

Tekstiviestien välittäminen Linux-sovelluksella Sijaishäly-järjestelmässä

Insinöörityö 14.4.2010

Työn ohjaaja: kehitysjohtaja Kari Häkkinen
Ohjaava opettaja: yliopettaja Matti Puska

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Vesa Ruohonen Tekstiviestien välittäminen Linux-sovelluksella Sijaishälyjärestelmässä 40 sivua 14.4.2010
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	kehitysjohtaja Kari Häkkinen yliopettaja Matti Puska

Insinöörityössä oli tavoitteena suunnitella ja toteuttaa tekstiviestien välitin Hakupajalle. Työssä laitteistona käytettiin minikannettavaa, johon oli asennettu Linux-käyttöjärjestelmä ja kaksi GSM-modeemia. Projektin päämääränä oli kehittää korvaaja Hakupajan tilapäisen henkilöstön hallintapalvelussa käytetyille tekstiviestivälittimille.

Välittimeen valittiin sopivat ohjelmistokomponentit ja tarpeelliset mutta puuttuvat ominaisuudet ohjelmoitiin projektia varten. Modeemien ohjaukseen valittiin SMS Server Tools 3 -ohjelmisto ja viestien käsittelyyn ohjelmoitiin erillinen Python-sovellus. Lisäksi testausympäristöä varten ohjelmoitiin palvelinsimulaattori.

Projektissa tehtyä prototyyppiä testattiin ja se saatiin täyttämään työlle annetut vaatimukset. SMS Server Tools 3 -ohjelmistopaketti osoittautui tehokkaaksi modeemirajapinnaksi eikä sen käytössä tai asennuksessa esiintynyt ongelmia. Python-sovellus, jota käytettiin viestien käsittelyssä, toimi hyvin, ja Python-kieli osoittautui hyväksi valinnaksi nopeaan ohjelmistokehitykseen. Ohjelmiston siirrettävyydessä ei myöskään esiintynyt ongelmia, kun sitä testattiin muissa ympäristöissä.

Projekti osoitti, että työhön annetulla laitteistolla voidaan rakentaa edullinen tekstiviestivälitin varsin lyhyessä ajassa. Testit osoittivat myös, että minikannettavan tehot riittivät hyvin annettuun tehtävään.

Hakusanat	Linux, GSM, SMS, minikannettava, Python

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	Vesa Ruohonen
Title	Relaying SMS messages with Linux application in work force management service
Number of Pages	40
Date	14 April 2010
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Kari Häkkinen, Development Director Matti Puska, Principal Lecturer

In this final-year project the task was to plan and develop an SMS gateway for Hakupaja. The equipment used in this project was a netbook employing Linux operating system with two GSM modems attached. The goal of the project was to develop a replacement for the existing SMS gateways in Hakupaja's work force management service.

Applicable software components were selected for the gateway, and the other necessary but missing functions were programmed for the project. SMS Server Tools 3 software was selected as a modem driver and a separate Python application was programmed to handle messages. Additionally, a server simulator was programmed for the testing environment.

The prototype made for the project was tested and it managed to meet the functional requirements. The SMS Server Tools 3 software package turned out to be an effective modem interface without any problems in usage or installation. The Python application used in message handling worked well and the Python language turned out to be a good choice for fast software development. Also, there were no problems with portability when the software was tested in other environments.

The project showed that a cheap SMS gateway can be built using the given hardware in quite a short period of time. The tests verified also that a netbook performed well enough for the task.

Keywords	Linux, GSM, SMS, netbook, Python

Lyhenteet, käsitteet ja määritelmät

ASCII	American Standard Code for Information Interchange, tietokoneissa käytetty amerikkalainen merkistöstandardi, joka sisältää vain englannin kielen kirjaimet, numerot ja joitain erikoismerkkejä.
AT	Hayes command set, modeemilaitteiden ohjaukseen suunniteltu komentokokoelma. GSM-laitteet tukevat komentokokoelman muotoilun mukaisia käskyjä tekstiviestien käsittelyssä.
C	Matalan abstraktiotason proseduraalinen ohjelmointikieli.
Eclipse	Avoimen lähdekoodin sovelluskehitysympäristö.
GSM	Groupe Spécial Mobile, euroopassa kehitetty matkapuhelinjärjestelmä, joka on nykyisin käytössä maailmanlaajuisesti.
HTTP	Hypertext Transfer Protocol, internetsivujen tiedonsiirrossa käytettävä protokolla.
ISO 8859-1	Vanha eurooppalainen merkistöstandardi. Tunnetaan epävirallisesti myös ISO Latin-1 -merkistönä.
Kernel	Käyttöjärjestelmän ydin.
Linux	Avoimen lähdekoodin käyttöjärjestelmä.
Merkistö	Määritelty esitystapa, jolla binääridatasta luodaan merkkejä.
Python	Korkean tason tulkkaava oliopohjainen ohjelmointikieli.
Rails	Ruby-ohjelmointikieltä käyttävä web-sovellusalusta, joka tunnetaan myös nimellä ”Ruby on Rails”.
Ruby	Korkean abstraktiotason tulkkaava oliopohjainen ohjelmointikieli.
SMS	Short Message Service, tekstiviestipalvelu.
USB	Universal Serial Bus, oheislaitteiden käyttöön suunniteltu sarjaliikennearkkitehtuuri.
USC2	Universal Character Set, vanha kansainvälinen merkistöstandardi, joka on UTF-16 yhteensopiva. Käyttää 16-bittistä esitystapaa.
UTF-8	Unicode Transformation Format, kansainvälinen merkistöstandardi.

Sisällys

1 Johdanto

Sähköisessä viestinnässä tekstiviesti on osoittautunut yhdeksi tehokkaimmista ja luotettavimmista tavoista tavoittaa haluttu henkilö. Vaikka lähes jokaisen voi tavoittaa nykyisin matkapuhelimeen soittamalla, tekstiviestiä käytettäessä vastaanottajan ei tarvitse edes olla paikalla samaan aikaan, koska se jää kuitenkin odottamaan lukijaansa.

On olemassa myös tilanteita, jolloin halutaan tavoittaa tietyt kriteerit täyttävä henkilö joukosta, mutta yhteydenottajaa ei kiinnosta, kuka joukosta poimitaan. Esimerkiksi kun tilataan taksi, asiakasta ei varsinaisesti kiinnosta, mille yksikölle soitto menee, kunhan se on tarpeeksi lähellä ja vapaana. Tarvitaan jokin luotettava ja helppo tapa valita sopiva vastaanottaja. Jos välityksessä voidaan käyttää tekstiviestejä puhelinsoittojen sijaan, koko järjestelmän automaation voi viedä vielä pidemmälle.

Myös sijaisia hakiessa pitää löytää sopiva kohde joukosta, mutta hakuun voidaan käyttää tekstiviestejä. Tässä insinöörityössä perehdytään Hakupajan Sijaishäly-palveluun, joka on henkilöstön hallintaan suunniteltu järjestelmä. Sijaishäly-järjestelmän keskeinen idea on tavoittaa haluttu ryhmä sijaisia tekstiviesteillä ja välittää tekstiviesteinä saapuneet vastaukset takaisin työnantajan sähköpostiin. Vaikka palvelu on osoittautunut varsin onnistuneeksi, sen teknisessä toteutuksessa olisi parantamisen varaa.

Tämän insinöörityön tarkoituksena oli suunnitella ja toteuttaa Linux-palvelimella toimiva välitysjärjestelmä, jolla voitaisiin korvata käytössä oleva järjestelmä. Tavoitteena oli käyttää halpaa laitteistoa.

Välitysjärjestelmän suunnittelu aloitettiin marraskuussa 2009, ja tavoitteena oli saada aikaan toimiva prototyyppi. Arviona oli, että määrityksien mukainen prototyyppi saataisiin ulos ennen maaliskuuta 2010. Työtä tultaisiin tekemään täysipäiväisesti etätyönä 40 tuntia viikossa.

2 Toimeksianto

2.1 Työn määrittely

Työnantajana toimi Hakupaja, joka on suomalainen IT-alan pienyritys. Hakupajalla on vain muutama vakituinen työntekijä. Sen tarjoamiin palveluihin kuuluvat konsultointi, henkilöstön koulutus ja tietotekniset ratkaisut. [1.]

Tehtäväksi annettiin Linux-palvelimella toimivan välittimen suunnittelu ja toteutus Sijaishäly-järjestelmään. Käytössä oleva järjestelmä on Windows-pohjainen, jossa käytetään Nokian ohjelmistoa ja laitteistoa. Se on osoittautunut kalliiksi ja rajoittuneeksi ratkaisuksi tähän tarkoitukseen.

Uuden välitysjärjestelmän tulee toimia halvoilla laitteilla. Sen täytyy tukea useiden eri valmistajien modeemeja sekä puhelimia. Järjestelmän pitää myös pystyä käsittelemään useampaa GSM-laitetta (Groupe Spécial Mobile) samanaikaisesti.

Palvelinlaitteen ei tarvitse olla kovin tehokas, mutta sen täytyy olla toimintavarma. Laitteen pitää toimia itsenäisesti ilman aktiivista ylläpitoa ja kestää esimerkiksi ajoittaisia virtakatkoksia. Minikannettava täyttää kaikki laitteelle asetetut vaatimukset, ja se on lisäksi vielä halpa. Kannettavaa käytettäessä myös modeemeista, joilla ei ole omaa virtalähdettä, tulee varmennettuja.

Työn tavoitteena on tehdä prototyyppi ohjelmistosta, joka hakee viestejä Sijaishäly-järjestelmän tietokannasta. Viestien haku tapahtuu HTTP-protokollaa (Hypertext Transfer Protocol) käyttäen, ja haetut viestit lähetetään tekstiviesteinä eteenpäin. Saapuneet tekstiviestit lähetetään vastaavasti Sijaishäly-järjestelmään.

Ohjelmisto hakee uusia viestejä Sijaishäly-järjestelmästä määrävälein. Määräväliä on voitava muuttaa järjestelmän kuormituksen mukaan.

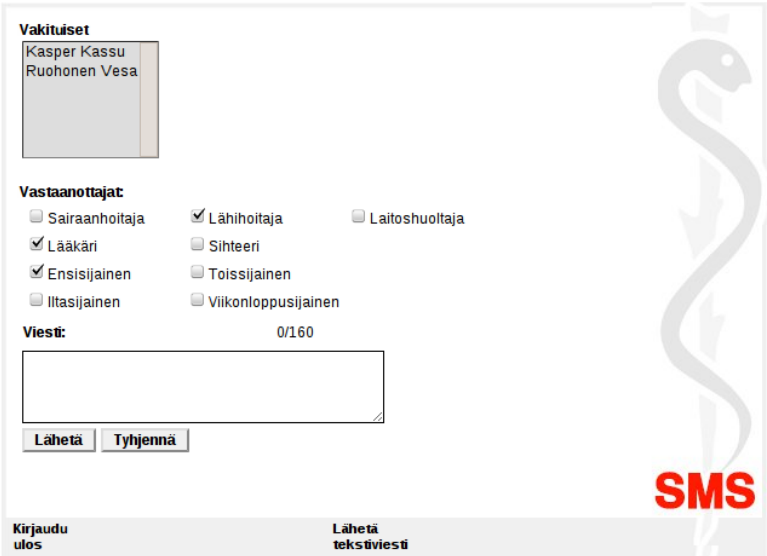
2.2 Sijaishäly-palvelu

Hakupajan Sijaishäly-palvelu on suunniteltu helpottamaan sijaisuuksien järjestämistä yrityksissä. Palvelun tarkoituksena on taata, että asiakkaalla olisi aina tarvittava työvoima käytössä myös hyvin lyhyellä varoitusajalla. [2.]

Tavanomaisesti tilapäistyövoimaa haetaan puhelinsoitoilla tai sähköpostilla. Jos sijaisia tarvitaan usein, puhelinsoittoihin kuluun paljon aikaa. Sähköposti on taas liian epäluotettava sijaisten hakuun lyhyellä varoitusajalla. Toki hakuun voidaan käyttää tekstiviestejä, mutta silti hakijan pitää tietää, keitä sijaisuuksiin haetaan. [3.]

Sijaiset ja heidän soveltuvuutensa eri tehtäviin voidaan rekisteröidä, jolloin sopivien henkilöiden haku helpottuu. Rekistereistä huolimatta yhteydenottoihin kuluu silti paljon aikaa. Tehokkaampi ratkaisu on käyttää rekisteriä ja automatisoida yhteydenotot.

Sijaishäly-palvelu tarjoaa asiakkaalle sekä keskitetyn rekisterin tilapäistyövoimasta että automaattisen viestien välityksen. Palvelu toimii hyvin yksinkertaisella kuvan 1 mukaisella selainpohjaisella käyttöliittymällä.



Kuva 1. Esimerkki Sijaishäly-palvelun selainpohjaisesta käyttöliittymästä

3 Suunnittelu

3.1 Suunnitteluperiaatteet

Ohjelmiston suoritus ei saa kaatua sisäisiin virheisiin. Monimutkainen toteutus johtaa helposti ennalta arvaamattomiin ongelmiin. Tarpeetonta monimutkaisuutta pyritään siis välttämään.

Kaikki kirjoitettava ohjelmakoodi tulee kommentoida asiallisesti, jotta ohjelmaa voidaan ylläpitää kehityksen jälkeen. Ohjelmakoodissa tulee käyttää selkeitä ja kuvaavia muuttujanimiä. Ohjelmalohkojen tarkoitus ja toiminta pitää selvitä niiden kuvauksesta.

Isot kokonaisuudet on syytä jakaa pieniksi osiksi. Niitä on helpompi suunnitella ja toteuttaa, mutta jos osat ovat toisistaan riippuvaisia, niiden synkronoinnista täytyy huolehtia.

Suunnittelussa tulee käyttää Linux-järjestelmän ohjelmointiperiaatteita. Esimerkiksi taustalla ajettavat ohjelmat tulee käynnistää Linux-käytännön mukaisesti [4, s. 550]. Ohjelmien pitää myös osata vastata hallintasignaaleihin [4, s. 203] ja toimia rajatuilla käyttäjäoikeuksilla.

3.2 Työssä käytetyt teknologiat ja ohjelmistot

3.2.1 Valintaperusteet

Käyttöjärjestelmä määriteltiin työnannossa. Linux-järjestelmä on käytössä työnantajalla jo entuudestaan. Sen etuina ovat ilmaisuus käyttäjälle, nopeus, luotettavuus, ylläpidettävyys ja hyväksi havaitut palvelinominaisuudet.

Modeemien rajapintaa käyttämään valittiin SMS Server Tools 3 -ohjelmisto. Se on laitteisto- ja käyttöjärjestelmäriippumaton kokonaisuus, jolla voidaan käyttää useita GSM-laitteita rinnakkain. Ohjelmisto on suunniteltu palvelinkäyttöön eikä siinä ole erillistä graafista käyttöliittymää lainkaan, mutta se on helposti integroitavissa muihin ohjelmiin. [5.]

Ohjelmointikielen abstraktiotason pitää olla riittävän korkea, jotta kieli on nopeasti omaksuttavissa. Kielestä tulee löytyä selkeä dokumentaatio. Sen täytyy olla yleisesti käytössä Linux-järjestelmissä ja käytettävissä myös tulevaisuudessa. Valitulla kielellä pitää myös päästä käsiksi käyttöjärjestelmän matalan tason ominaisuuksiin.

Python-ohjelmointikieli täyttää kaikki kielelle asetetut vaatimukset. Lisäksi se on tulkkaava kieli, joten Pythonilla kirjoitettua ohjelmakoodia ei tarvitse kääntää. Siksi Python-sovellukset ovat myös helposti siirrettävissä koneelta toiselle. [6, s. 3–6.]

3.2.2 Linux sovellusympäristönä

Linuxissa ohjelma voidaan suorittaa vain prosessina. Prosessissa määritellään ohjelman oikeudet tiedostoihin ja muistialueisiin, joita se voi käyttää. Käyttöjärjestelmä antaa jokaiselle käynnistetylle prosessille eri tunnusluvun (process ID, PID). Tunnusluku on voimassa kunnes prosessi lopetetaan. [4, s. 105–107.]

Prosessi on itsenäinen kokonaisuus, jossa ajettava ohjelma suoritetaan alusta loppuun. Mikäli halutaan suorittaa tehtäviä rinnakkain, voidaan luoda uusia prosesseja. Ohjelmat voivat luoda uusia prosesseja kopioimalla itsensä (fork). Ohjelman luomia prosesseja kutsutaan lapsiprosesseiksi, ja ne toimivat kuten alkuperäinen prosessi. [4, s. 125.]

Vaihtoehtoinen tapa rinnakkaissuoritukseen on säikeiden käyttäminen. Ohjelma voi luoda uusia prosesseja, säikeitä, joissa ohjelman resurssit jaetaan. Säikeitä käyttämällä prosessien hallinta helpottuu, mutta resurssien käyttö on koordinoitava huolellisesti. [7.]

Prosessien hallintaan Linuxissa käytetään signaaleja. Prosessin kannalta signaali on tiedote ulkoisesta tapahtumasta, johon sen toivotaan vastaavan. Signaalit on jaettu useaan tyyppiin, joille ohjelmassa voidaan määritellä haluttu vaste. Signaalit kohdistetaan prosesseille käyttämällä niiden tunnuslukua. [4, s. 203.]

Ohjelmaa voidaan ajaa joko käyttäjän ohjaamana tai käyttäjästä riippumatta taustalla. Ohjelmat käynnistetään käyttäjän omaan istuntoon. Käyttäjän istunnon poistuessa kaikki siellä ajettavat ohjelmat lopetetaan. Jos ohjelmaa halutaan ajaa taustalla istunnosta riippumatta, se täytyy muuttaa palveluksi (Daemon). [4, s. 550.]

3.2.3 SMS Server Tools

SMS Sever Tools, eli lyhyesti SMS-tools, on tekstiviestien välittämiseen tarkoitettu avoimen lähdekoodin ohjelmisto. Siinä on tiedostopohjainen rajapinta tekstiviestiliikenteen hallintaan, ja se tukee useamman GSM-laitteen yhtäaikaista käyttöä. [5.]

Stefan Frings julkaisi ensimmäisen version ohjelmistosta vuonna 2000. Kuuden vuoden kehityksen jälkeen ohjelma saavutti toisen version ja samalla kehitystyö siirtyi Keijo Kasville. SMS-tools on nyt kolmannessa versiossa ja sen kehitys jatkuu edelleen aktiivisesti. [8.]

Ohjelmisto on kirjoitettu puhtaasti C-kielellä, ja se toimii Unix-pohjaisilla käyttöjärjestelmillä, kuten Linuxilla. Windows-käyttöjärjestelmällä sitä voidaan ajaa CygWin-rajapinnan avulla. SMS-tools tukee melkein mitä tahansa GSM-laitetta, kunhan se noudattaa GSM 07.05- ja GSM 03.38-standardia. Laitteita voidaan käyttää joko sarjaportti-, infrapuna- tai USB-liitännän kautta (Universal Serial Bus). [5.]

3.2.4 Python

Python on Guido van Rossumin luoma alustariippumaton oliopohjainen ohjelmointikieli, jota voidaan myös käyttää proseduraaliseen ohjelmointiin. Python-kieli on yksinkertainen, syntaksiltaan yhtenäinen ja laajennettavissa. Kieli on myös helposti sidottavissa muihin ohjelmointikieliin, joten sitä voidaan käyttää osana monimutkaisissa ohjelmistokokonaisuuksissa. [9, s. 3–5.]

Eräs kielen keskeisistä ominaisuuksista on muuttujien dynaaminen tyypitys, jonka takia muuttujien tyyppiä ei tarvitse esitellä lainkaan. Esimerkiksi muuttuja, jolle annetaan arvoksi kokonaisluku, tulkitaan kokonaislukumuuttujaksi. Muuttujan tyyppi ei ole myöskään sidottu ensimmäiseen määritykseen, vaan se voi vaihtua ohjelman suorituksen aikana. [10.]

Koska Python on korkean abstraktiotason ohjelmointikieli, ohjelmoijan ei tarvitse puuttua laitetason asioihin. Esimerkiksi muistinhallintaan Python käyttää omaa automaattista roskienkeruujärjestelmää, jonka myötä ohjelman ei tarvitse varata eikä tyhjentää muistialueita. Python-ohjelmaa ei myöskään tarvitse kääntää. [11.]

Kieli kehitettiin 1990-luvun alussa, ja vuonna 2008 julkaistiin sen kolmas versio, jossa keskityttiin siivoamaan päällekkäisyyksiä rajapinnoista sekä yksinkertaistamaan merkkijonojen ja merkistöjen käyttöä. Valitettavasti muutoksista johtuen kielen kolmas versio ei ole täysin yhteensopiva vanhempien Python-ohjelmien kanssa. [11.]

3.2.5 Eclipse

Eclipse on avoimen lähdekoodin graafinen kehitysympäristö. Se luotiin aluksi Java-ohjelmointia varten, mutta laajennettiin pian tukemaan myös muita ohjelmointikieliä. Eclipse on ilmainen ja alustariippumaton ohjelmointityökalu, joka on ominaisuuksiltaan verrattavissa kaupallisiin sovelluskehittimiin. [12.]

Tässä työssä ohjelmointiin käytettiin Eclipseä pydev-laajennuksella, joka mahdollistaa Python-ohjelmien kirjoittamisen kehitysympäristöllä.

3.3 Sijaishäly-järjestelmän rajapinta

Sijaishäly-järjestelmän rajapinta toimii kuvan 2 mukaisilla HTTP-protokollaa käyttävillä komennoilla.

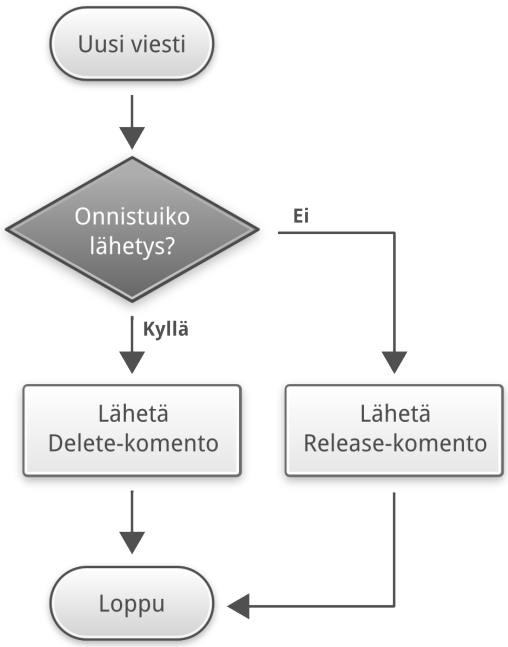
Kuva 2. Sijaishäly-järjestelmässä käytettävän HTTP-komennon rakenne.

Talukossa 1 on lueteltu järjestelmän käyttämät komennot sekä niihin odotettavat vastaukset. Esimerkiksi Get-komento antaa uuden viestin vastauksessa tai, jos uusia viestejä ei ole, tyhjän vastauksen.

Taulukko 1. Sijaishäly-järjestelmän rajapinnassa käytetyt komennot.

Komento	**Data**	**Vastaus**
Get	-	PkMSG**n**<ID>**n**<GSM numero>**n**<viestin sisältö>
Delete	<ID>	Ok / -
Release	<ID>	Ok / -
Reply	<GSM numero>**t**<viestin sisältö>	Email Ok / -

Sijaishäly-järjestelmässä viestien hallinta on pyritty tekemään yksinkertaiseksi ja toimintavarmaksi. Kuvan 3 vuokaaviossa esitetään lähtevien viestien käsittely välittimen kannalta. Viestien lähetystä seurataan GSM-verkkoon asti ja lopputuloksesta raportoidaan takaisin palvelimelle. Mikäli viestin käsittely kestää liian pitkään, tulkitaan sen lähetys epäonnistuneeksi. Näin järjestelmä ei koskaan hukkaa lähteviä viestejä vahingossa.



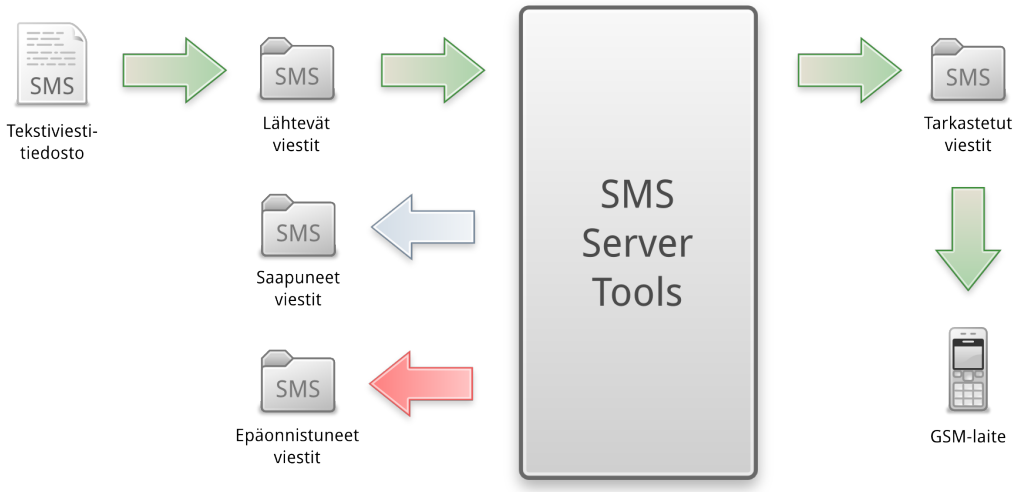
Kuva 3. Kaavio viestin käsittelytapahtumasta välittimellä

Välittimeen saapuneet tekstiviestit lähetään palvelimelle Reply-komentoa käyttäen taulukon 1 mukaisesti. Palvelin kuittaa viestin, mikäli lähetys onnistui.

3.4 Tekstiviestien käsittely SMS Server Tools -ohjelmistolla

Järjestelmän rakenne

SMS-tools käsittelee tekstiviestejä kuvan 4 mukaisilla jonoilla. Lähteville, saapuneille, epäonnistuneille ja tarkistetuille viesteille on omat jononsa. Järjestelmässä jonot ovat hakemistoja ja viestit ovat yksittäisiä tekstitiedostoja, jotka on muotoiltu määritelmän mukaisesti. Lähtevissä viesteissä tiedostonimellä ei ole väliä, kunhan tiedostoja ei ylikirjoiteta ennen käsittelyä. [8.]



Kuva 4. Kaavio SMS Server Tools -ohjelmiston hakemistopohjaisista jonoista

Tekstiviestitiedostojen rakenne

Viestitiedoston rakenne koostuu kahdesta osasta. Otsikkotiedoissa määritellään esimerkiksi, minne viesti lähetetään, millä maakoodilla numero tulkitaan ja mitä merkistöä viestin tekstissä käytetään. Ainut pakollinen määritelmä on tieto numerosta, johon viesti lähetään. Muut ominaisuudet voidaan jättää määrittelemättä, jolloin ne tulkitaan oletusasetuksien mukaisesti. Loppuosa tiedostosta on varattu viestin sisällölle. [13.]

Tekstiviestien lähettäminen

Tekstiviestien lähettäminen SMS-tools ohjelmistolla on suoraviivaista. Tekstiviestitiedon rakennetta noudattava viesti kirjoitetaan lähtevien viestien hakemistoon yksilöllisellä nimellä, ja SMS-tools huolehtii sen lähettämisestä. [8.]

SMS-tools tutkii lähettävien viestien hakemistoa määrävälien ja ottaa sieltä löytyneitä tiedostoja käsiteltäväksi. Tarkistuksen läpäissyt viesti siirretään tarkistettujen viestien hakemistoon. Mikäli tarkistus epäonnistuu, SMS-tools siirtää tiedoston hylättyjen viestien hakemistoon. [8.]

SMS-tools välittää hyväksytyn viestin ensimmäiselle vapaalle modeemille lähetettäväksi. Jos lähetys onnistuu, viesti poistetaan. Jos viestin lähetys epäonnistuu, yritetään lähettää viesti kertaalleen uudestaan. Ellei viestin lähetys onnistu toisellakaan yrityksellä, SMS-tools siirtää sen hylättyjen viestien hakemistoon. [8.]

Tekstiviestien vastaanottaminen

Modeemille saapunut viesti tallennetaan tiedostoksi saapuneiden viestien hakemistoon. Viestitiedoston nimenä käytetään modeemin nimen ja satunnaisen merkkijonon yhdistelmää, jotta vältyttäisiin tiedostojen ylikirjoittumiselta. [8.]

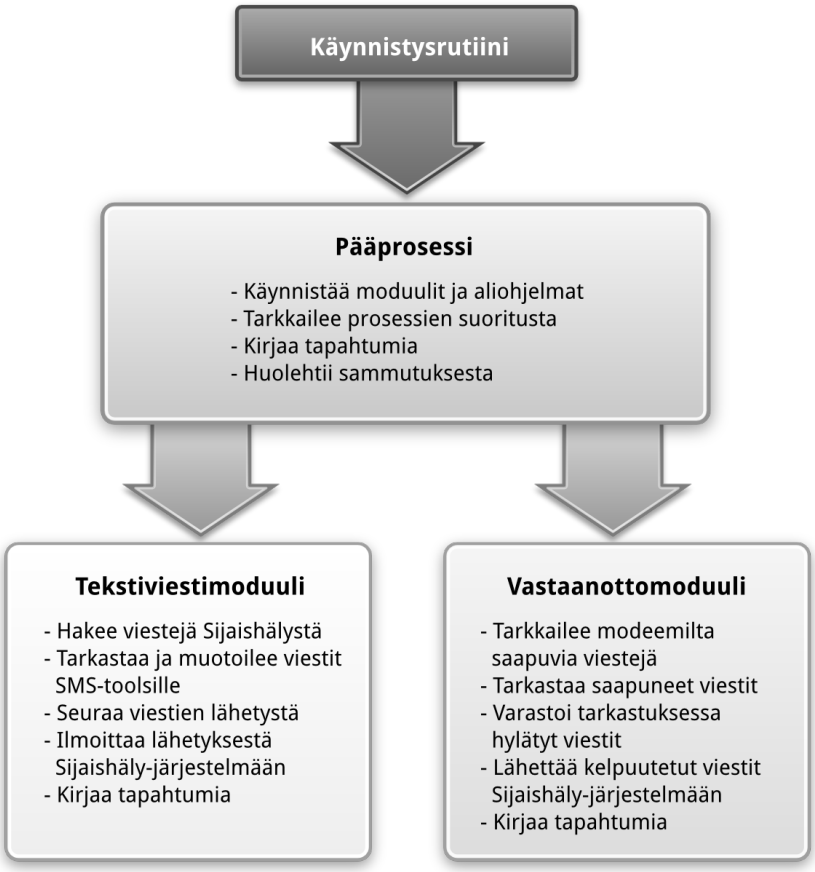
3.5 Python-sovelluksen suunnittelu

Sovelluksen rakenne

Sijaishäly-järjestelmän ja SMS-toolsin välille tarvittiin sovellus, joka välittää viestejä näiden kahden välillä. Python-sovelluksen suunnittelussa lähdettiin siitä, että kumpaankin suuntaan tapahtuva välitys jaetaan omaksi riippumattomaksi prosessikseen.

Kahden erillisen sovelluksen hallinta sellaisenaan on kuitenkin hankalaa. Erillisien sovelluksien sijaan päätettiin tehdä yksi ohjelma, jossa eri tehtävät suoritetaan rinnakkaisissa moduuleissa.

Moduulien käynnistyksestä, valvomisesta ja sammuttamisesta huolehtii pääprosessi. Koska sovellusta on tarkoitus ajaa taustapalveluna, pääprosessin käynnistykseen tarvitaan käynnistysrutiini, joka voi myös vastaanottaa käyttäjän antamat parametrit. Kuvassa 5 esitelty rakenne tekee sovelluksesta selkeämmän ja helposti laajennettavan.



Kuva 5. Python-sovelluksen rakennekaavio

Sovelluksen vikasietoisuus

Koska ohjelmiston on tarkoitus toimia itsenäisesti, se ei saa mennä lukkoon suorituksen aikana. Logiikkavirheet ohjelmakoodissa voivat johtaa suorituksen lukkiutumiseen, joten ohjelmakoodia pitää testata huolella kehityksestä alkaen. Toinen lukkiutumiseen johtava ongelma on odotuslohkoissa, joissa ei ole aikarajaa. Jos jotain suorituksessa menee pieleen eikä odotuslohkosta päästä eteenpäin, ohjelman suoritus voi jäädä jumiin.

Pitää ottaa huomioon ettei, yksi lukkiutuminen saa koko järjestelmää pysähtymään. Siksi perustoimintojen pitää tapahtua toisistaan riippumatta. Jotta esimerkiksi yhden GSM-laitteen hajoaminen ei johtaisi tekstiviestilähetyksien pysähtymiseen, myös rinnakkaisten toimintojen pitää tapahtua toisistaan riippumatta.

Sovelluksessa saatetaan myös tulla tilanteeseen, josta jatkaminen ei yksinkertaisesti enää onnistu. Yksi tilanne on esimerkiksi se, jos SMS-toolsin hakemistoihin kirjoittaminen tai niistä lukeminen ei onnistu. Jos tiedostojen käsittely ei onnistu, prosessit eivät voi kommunikoida keskenään. Jos sisäisessä tallennusmediassa on ongelmia, toimintaa ei ole syytä jatkaa. Sijaishäly-järjestelmän kanssa on oltava kärsivällisempi, sillä internet-yhteydessä saattaa esiintyä tilapäisiä katkoksia.

Kumpaankin suuntaan välitetyn viestin pitää säilyä virhetilanteissa. Sijaishäly-järjestelmästä haettaessa pitää varmistaa, että viesti on GSM-verkossa ennen kuin käsittelykuittaus annetaan. Viesti ei katoa mihinkään, vaikka välitin hukkaisi sen käsittelyssä, koska Sijaishäly-järjestelmä palauttaa viestin kiertoon käsittelyyn varatun aikarajan umpeutuessa. Tällä tavoin varmistetaan, ettei viesti voi kadota lähetyksessä.

Vastaanotettavat viestit saapuvat aluksi GSM-laitteille. SMS-tools hakee saapuneet viestit GSM-laitteilta ja kirjoittaa ne tiedostoiksi saapuvien viestien hakemistoon. Python-sovellus käsittelee saapuneet viestit ja yrittää lähettää ne Sijaishäly-järjestelmään. Vasta kun Sijaishäly-järjestelmästä saadaan myöntävä vastaus, viestin voi poistaa. Näin varmistetaan, ettei viesti voi kadota myöskään vastaanotettaessa.

4 Python-sovelluksen toteutus

4.1 Käynnistysrutiini

Ensimmäisenä sovellukseen toteutettiin käynnistysrutiini ja runko. Ohjelman käynnistyksessä tarkistetaan komentoriviltä annetut parametrit ja toimitaan niiden mukaan. Esimerkiksi jos halutaan saada enemmän tietoa ohjelman toiminnasta, voidaan parametriksi antaa ”--help”. Tällöin ohjelma vain listaa komennot eikä käynnistä varsinaista sovellusta.

Parametrien syöttämisen lisäksi käynnistysrutiini ottaa ohjelman sijainnin talteen ja yrittää etsiä määritystiedoston, jossa kerrotaan, missä asetustiedosto sijaitsee. Jos määritystiedostoa ei löydy, asetustiedostoa etsitään ohjelmahakemistosta oletusnimellä.

4.2 Asetuksien säilyttäminen

Asetustiedostossa käytetään yksinkertaista muotoilua, missä ”=”-merkillä erotetaan toisistaan asetettava parametrin avainsana ja asetuksen arvo. Avainsanan loppuun ja asetuksen arvon alkuun on suositeltavaa laittaa yksi välilyönti luettavuuden parantamiseksi, mutta se ei ole pakollista.

Asetustiedosto on jaettu osioihin, joiden otsikot merkitään hakasulkein. Ensimmäisessä osioissa ei ole otsikkoa, ja se tulkitaan aina pääosioksi. Jaottelun tarkoituksena on selkeyttää parametrien asiayhteyttä, ja se mahdollistaa myös samojen avainsanojen käytön eri osioissa.

Asetukset luetaan tiedostosta rivi kerrallaan. Jos asetustiedoston rivi on tyhjä tai alkaa ”#”-merkillä, sitä ei käsitellä lainkaan. Jokaisen osion käsitellyistä riveistä luodaan lista, joka lisätään hakuteokseen (dictionary [9, s. 44]) osion otsikon alle.

4.3 Pääprosessi

Ohjelman pääprosessin suoritus alkaa siitä, että prosessista tehdään tausta-ajoprosessi. Jotta ohjelmaan päästäisiin helposti käsiksi käynnistyksen jälkeen, sen tunnusluku (PID) kirjataan erilliseen tiedostoon. Pääprosessi on periytetty liitteen 5 daemon-oliosta.

Käynnistyksen yhteydessä pääprosessi alustaa itselleen uudet signaalikäsittelijät yleisimmille Linuxissa käytetyille signaaleille. Signaalikäsittelyn tarkoituksena on varmistaa, että sovelluksen sammutuksen yhteydessä myös aliprosessit saadaan ajettua alas.

Alustuksen jälkeen pääprosessi käy läpi asetukset ja käynnistää alimoduulit käyttäen moduulilataajaa, joka on esitetty liitteessä 6. Ulkoiset ohjelmat käynnistetään moduulien jälkeen asetusmäärityksien mukaisesti. Jokaisen käynnistetyn aliprosessin tunnusluku kirjataan samaan tiedostoon pääprosessin tunnusluvun perään. Samalla sekä alimoduulit että ulkoiset ohjelmat kerätään omiin listoihinsa, joista niitä voidaan tarkkailla ohjelman suorituksen aikana.

Kun aliprosessit on käynnistetty, pääprosessin on valvottava, ettei yhdenkään aliprosessin suoritus päädy virheeseen. Valvominen tapahtuu tarkastelemalla aliprosessien tilaa määräajoin silmukassa. Silmukan suoritusta jatketaan, kunnes pääprosessi saa ulkoisen lopetussignaalin tai havaitsee ongelmia aliprosessien suorituksessa. Mikäli moduulin tai ulkoisen ohjelman suoritus on päättynyt virheeseen, pääprosessi aloittaa koko ohjelmiston sammutuksen.

Ohjelman sammutuksessa pääprosessi lähettää jokaiselle moduulille ja ulkoiselle ohjelmalle lopetussignaalin ja jää odottamaan niiden sammumista. Kun aliprosessit ovat sammuneet, pääprosessi viimeistelee sammutuksen poistamalla tunnusluvut sisältävän tiedoston.

4.4 Moduulit

4.4.1 Tekstiviestimoduuli

Tekstiviestimoduuli alustaa käynnistyksen yhteydessä itselleen signaalikäsittelijät ja yrittää hakea asetuksilleen arvot asetustiedostosta. Jos asetukselle ei löydy asetustiedostosta arvoa tai sen lukeminen epäonnistuu, käytetään oletusarvoa. Asetuksista luetaan esimerkiksi käsittelyyn tarvittavat viestihakemistot.

Käynnistyksen jälkeen moduulin täytyy tarkistaa luku- ja kirjoitusoikeudet viestihakemistoihin. Mikäli oikeudet ovat puutteelliset tai hakemistoja ei löydy, moduuli sammuttaa itsensä virheeseen. Koska käyttöympäristössä saattaa suorituksen aikana tapahtua arvaamattomia muutoksia, täytyy tarkistus tehdä säännöllisesti.

Moduulin pääsilmukan tehtävänä on hakea viestejä Sijaishäly-järjestelmästä ja toimittaa niitä SMS-tools -ohjelmistolle lähetettäväksi. Moduuli valvoo viestin lähetystä ja ilmoittaa viestin lopullisesta kohtalosta takaisin Sijaishäly-järjestelmään liitteessä 1 esitetyn vuokaavion mukaisesti. Pääsilmukkaa suoritetaan, kunnes tekstiviestimoduuli vastaanottaa lopetussignaalin tai sen suorituksessa tapahtuu vakava virhe.

Kun Sijaishäly-järjestelmästä saadaan haettua uusi viesti, sen sisältö tarkastetaan. Jos viesti kelpuutetaan, kirjoitetaan sen sisältö tekstiviestitiedostoksi lähtevien viestien hakemistoon. Tekstiviestitiedostoille generoidaan yksilölliset nimet, jotka rakentuvat ”SJH + aikaleima + kahdeksan satunnaista alfanumeerista merkkiä” -kaavan mukaisesti. Jotta tiedostonimien kanssa ei tulisi odottamattomia ongelmia, käytetään kirjain-merkkeinä vain englanninkielisiä aakkosia.

Tekstiviestitiedoston lähetystä valvotaan tutkimalla lähtevien viestien hakemistoa sekunnin määrävälein. Mikäli viestitiedosto katoaa hakemistosta kahden minuutin aikana, tarkastetaan, löytyykö se muista hakemistoista. Jos viestiä ei löydetä mistään hakemistosta, voidaan olettaa, että SMS-tools lähetti viestin. Lopuksi lähetyksen onnistumisesta ilmoitetaan Sijaishäly-järjestelmälle.

Jos viesti on löydettävissä kahden minuutin päästä mistä tahansa hakemistosta, sen lähetys ei ole onnistunut. Tällöin viestitiedosto poistetaan ja epäonnistuneesta lähetyksestä ilmoitetaan Sijaishäly-järjestelmälle.

4.4.2 Vastaanottomoduuli

Tekstiviestimoduulin tapaan myös vastaanottomoduuli alustaa käynnistyksessä signaalikäsittelijät ja hakee asetukset asetustiedostosta. Mikäli asetuksien hakeminen ei onnistu, moduuli käyttää oletusarvoja.

Käynnistyksen jälkeen tarkistetaan luku- ja kirjoitusoikeudet viestihakemistoihin. Mikäli oikeudet ovat puutteelliset tai hakemistoja ei löydy, moduuli sammuttaa itsensä virheeseen. Tarkistus täytyy suorittaa määräajoin, koska ajoympäristössä saattaa tapahtua odottamattomia muutoksia.

Vastaanottomoduulin pääsilmukan tehtävänä on lähettää SMS-toolsin vastaanottamia tekstiviestejä Sijaishäly-järjestelmään. Moduuli tarkkailee saapuneiden viestien hakemistoa ja lähettää kelpuutetut tekstiviestit Sijaishäly-järjestelmälle. Pääsilmukan suoritusta jatketaan, kunnes vastaanottomoduuli saa lopetussignaalin tai sen suorituksessa tapahtuu vakava virhe. Moduulin toiminta kokonaisuudessaan on kuvattu liitteen 2 vuokaaviossa.

Uudet viestit poimitaan vastaanotettujen viestien hakemistosta satunnaisesti, jotta yksi viallinen tiedosto ei voisi jumittaa koko viestien käsittelyä. Saapuneesta tekstiviestistä yritetään hakea lähettäjän numero sekä viestin sisältö. Jos viestin sisältöä tai lähettäjän numeroa ei kelpuuteta, vastaanottomoduuli siirtää viestitiedosto talteen hylättyjen viestien hakemistoon.

Mikäli vastaanottomoduuli kelpuuttaa saapuneen viestin, se muotoillaan ja lähetetään Sijaishäly-järjestelmälle. Lähetyksen onnistuessa viestitiedosto poistetaan saapuneiden viestien hakemistosta. Jos lähetys epäonnistuu, moduuli yrittää lähettää sen myöhemmin uudestaan.

Koska vastaanottomoduuli joutuu arkistoimaan tiedostoja, se ei saa kuormittaa kiintolevyä turhaan. Tästä syystä uutta tiedostoa kirjoitettaessa moduuli tarkistaa, ettei tiedostojen määrä hakemistossa pääse ylittämään annettua rajaa. Mikäli raja ylittyy, moduuli poistaa vanhimman tiedoston hakemistosta ennen uuden kirjoittamista.

4.4.3 Säikeiden käyttö moduuleissa

Tehtävänannossa edellytettiin, että viestin lähetys- ja vastaanottointensiteettiä voidaan muuttaa kuormituksen mukaan. Alkuperäisessä moduulitoteutuksessa jokainen tapahtuma käsitellään alusta loppuun ennen seuraavan aloittamista. Yksittäistä käsittelytapahtumaa ei voi juurikaan nopeuttaa, mutta useita toisistaan riippumattomia tapahtumia voidaan säikeiden avulla käsitellä rinnakkain. [14.]

Kummassakin moduulissa rakennetta muutettiin siten, että uusia viestejä haetaan useampi kerrallaan käsiteltäväksi. Haetut viestit pistetään käsittelyjonoon, josta ne jaetaan käsittelysäikeille. Moduuli jää odottamaan säikeiden työskentelyä. Kun säikeet saavat työnsä valmiiksi, moduuli odottaa hetken ja palaa pääsilmukan alkuun. Viestejä haetaan enintään yksi jokaiselle moduulin säikeelle. [14.]

Säikeitä käyttävät moduulit toimivat vähäisillä viestimäärillä samaan tapaan kuin ilman säikeitä. Kiireellisinä hetkinä usean viestin rinnakkainen käsittely kuitenkin nopeuttaa moduulien työskentelyä huomattavasti. Koska viestien käsittelytapahtumat eivät ole täysin toisistaan riippumattomia, moduulien toteutuksissa piti varmistaa, etteivät säikeet aiheuta uusia ongelmia.

Tekstiviestimoduulin uusi toteutus toimi liitteen 3 kaavion mukaan. Virhetilanteiden välttämiseksi kirjoitustapahtuma piti kuitenkin varmistaa lukko-objektilla [14], jotta vain yksi säie kerrallaan voi kirjoittaa viestitiedoston.

Vastaanottomoduuli saatiin vastaavasti toimimaan säikeillä liitteen 4 mukaisella toteutuksella. Säikeitä käytettäessä saapuneiden viestien hakemistosta joudutaan poimimaan useampi tiedosto käsiteltäväksi. Yhdellä käsittelykierroksella saa poimia saman tiedoston vain kerran. Jos uusia viestejä on vähemmän kuin käsittelysäikeitä,

otetaan kaikki uudet viestit kerralla käsittelyyn. Muutoksia jouduttiin myös tekemään viallisten viestien varastointiin. Lukko-objektia käyttämällä varmistetaan, että vain yksi säie kerrallaan voi suorittaa luku- ja kirjoitusoperaatioita.

4.5 Tapahtumien kirjaus

Kirjattavan tiedon määrää on pystyttävä hallitsemaan, jotta vältyttäisiin turhalta kiintolevyn täyttämiseltä. Python-kielestä löytyy monipuolinen logging-kirjasto tapahtumien kirjaamiseen. Sitä voidaan käyttää myös säikeissä ongelmitta. Kirjasto tukee esimerkiksi kirjaustiedostojen kokorajoituksia, kiertävää kirjausta ja viestien kirjaustasoja. [15.]

Kiertävässä kirjauksessa kirjaustiedostolle määritellään enimmäiskoko. Kun kirjaustiedosto tulee täyteen, se suljetaan ja otetaan talteen. Suljetuille tiedostoille annetaan juokseva numero. Niiden lukumäärä on rajoitettu. Jos tiedostojen lukumäärä lisäyksen yhteydessä ylittyy, vanhin tiedosto poistetaan. Sovelluksessa pääprosessi ja moduulit kirjaavat tapahtumat omiin tiedostoihinsa, joista pidetään kymmenen kirjaustiedoston historia kiertävällä kirjauksella. [15.]

Tapahtumat jaetaan neljään eri kirjaustasoon. Kehitystason (debug) viestit sisältävät hyvin yksityiskohtaista ja useasti toistuvaa tietoa, joten niiden kirjaamisesta on hyötyä sovellusta kehitettäessä. Informaatiotason (information) viestit sisältävät tietoa sovelluksen toiminnasta normaalitilassa. Kriittiset (critical) ja virhetason (error) viestit sisältävät tietoa suorituksen aikana havaituista ongelmista. Kahden viimeisen tason viestit ovat oleellisia vikatilanteiden selvityksessä. [15.]

Tapahtumien kirjauksen asetuksia voi muuttaa sovelluksen asetustiedostosta, ja ne voidaan halutessa myös tulostaa ruudulle suorituksen aikana.

4.6 Yhteensopivuus Python-versioiden välillä

Ohjelmoinnissa jouduttiin ottamaan huomioon Python-kielen versiot 2.5+ ja 3.x. Vielä toistaiseksi kielen toinen versio on laajalti käytössä, mutta tilanne saattaa muuttua tulevaisuudessa. Vaikka Python-kielen eri versioita voidaan asentaa samaan

järjestelmään rinnakkain, sovelluksen käyttöönotto helpottuu huomattavasti, mikäli voidaan käyttää valmiiksi asennettua versiota. Tästä syystä sovelluksen tulisi toimia kummallakin versiolla. Koska sovelluksessa ei käytetä ulkoisia Python-kirjastoja, yhteensopivuuden takaaminen on helpompaa, muttei täysin ongelmatonta.

Kolmannessa versiossa on suuria muutoksia merkkijonojen ja merkistöjen käsittelyssä. Kolmas versio käsittelee merkkijonot UTF-8-merkistöstandardilla (Unicode Transformation Format), kun vanhemmissa versioissa merkkijonot käyttivät ASCII-merkistöstandardia (American Standard Code for Information Interchange). Toinen suuri muutos on raakadatan käsittelyssä. Aikaisemmat versiot käsittelevät myös dataa merkkijonoina, mutta kolmannessa versiossa data käsitellään tavujonoina (bytes), jotka muodostuvat kokonaisluvuista kirjaimien sijaan. [16.]

HTTP-kirjaston kanssa ilmeni ongelmia, koska Python-versiosta riippuen käsiteltävä data on joko merkki- tai tavujonoina. Kolmatta versiota käyttäessä tavujono täytyy muuntaa merkkijonoksi. Tavujonon muunnoksessa saattaa kadota sisältöä, jos se muunnetaan väärään merkistöön. Voidaan kuitenkin olettaa, että kaikki Sijaishäly-järjestelmästä haettava data käyttää ISO 8859-1 -merkistöä. [16.]

Sama ongelma ilmenee vastaanotettuja viestitiedostoja luettaessa, koska tiedostojen luku palauttaa kolmannella versiolla tavujonon merkkijonon sijaan [16]. Vaikka viestin sisältö voi käyttää montaa eri merkistöä, lopulta se joudutaan kuitenkin kääntämään ISO 8859-1 -merkistölle Sijaishäly-järjestelmän takia.

5 Järjestelmän testaus

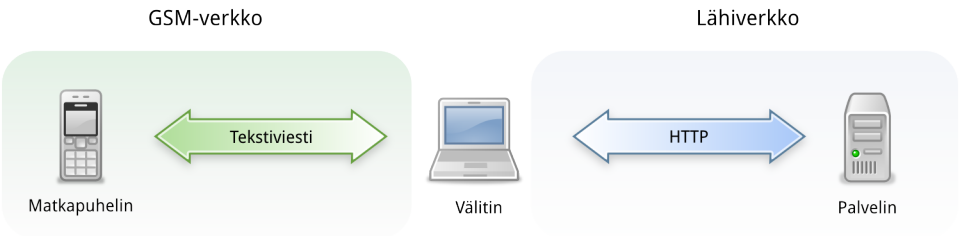
Testauslaitteisto

Hakupaja toimitti Huawei E169 USB -modeemin, jolla voitiin lähteä kokeilemaan GSM-laitteiden toimintaa. Parin viikon kuluttua Hakupaja toimitti Acer Aspire One -minikannettavan ja Bandluxe C100S USB -modeemin, joilla prototyyppi oli tarkoitus saada toimimaan.

Minikannettavassa oli Windows XP -käyttöjärjestelmä asennettuna, mutta USB-massamuistilta saatiin Linux asennettua sen rinnalle ongelmitta. Hakupajan suosituksena on käyttää Ubuntu 9.04 -jakelun palvelinversiota, mutta testauksen helpottamiseksi koneelle asennettiin Mandriva 2010.0 -työpöytäjakelu.

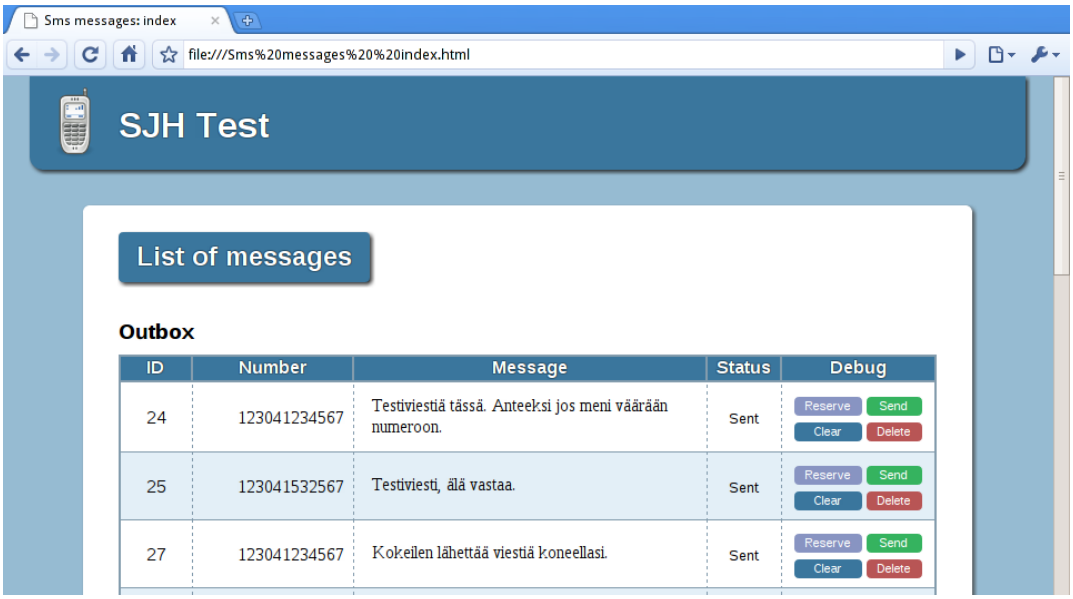
Testausympäristö

Testeissä välittimenä toimi Hakupajalta saatu minikannettava, johon oli yhdistetty kaksi GSM-modeemia. Minikannettavalle asennettiin SMS-tools käyttämään molempia modeemeja rinnakkain ja Python-sovellus välittämään viestejä palvelimelle. Palvelinkoneena toimi samassa lähiverkossa oleva pöytäkone, jolla ajettiin Sijaishäly-järjestelmän rajapinnan toteuttavaa simulaattoria. Testausympäristö on kaaviona kuvassa 6.



Kuva 6. Kaavio työn testaukseen käytetystä testiympäristöstä

Työtä varten jouduttiin ohjelmoimaan erillinen Sijaishäly-simulaattori, koska tuotantojärjestelmää ei vielä tässä vaiheessa saatu käyttöön. Simulaattori tehtiin Ruby on Rails -sovelluksena. Testauksen helpottamiseksi simulaattoriin tehtiin selainpohjainen, kuvassa 7 näkyvä, käyttöliittymä.



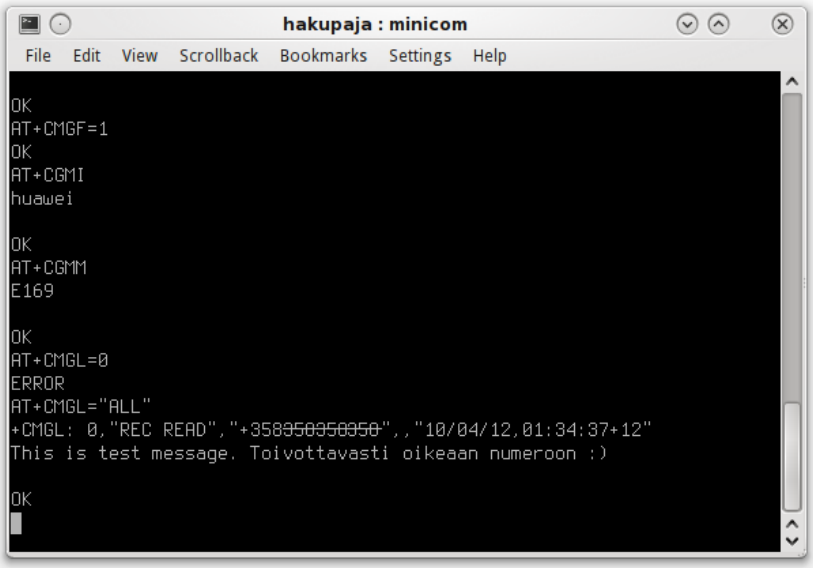
Kuva 7. Välittimen testausta varten ohjelmoidun simulaattorin selainpohjainen käyttöliittymä

GSM-modeemien testaus

Linux-ympäristössä laitteet esitetään tiedostoina. Modeemit näkyvät laitetiedostoina järjestelmän /dev-hakemistossa. Jokainen modeemi luo kytkettäessä oman laitetiedoston, jonka kautta laitetta voidaan käyttää. [4, s. 156.]

Modeemin yhteysnopeus ja tarvittavat alustuskomennot vaihtelevat mallista riippuen. Laitteen toimivuutta voidaan testata sarjaliikenneohjelman avulla. Mikäli modeemi saadaan toimimaan sarjaliikenneohjelman kanssa, sitä voidaan käyttää samoilla asetuksilla myös SMS-toolsissa.

Työnantajalta saadut modeemit testattiin aluksi käyttäen minicom-sarjaliikenne-ohjelmaa. Testeissä yritettiin lähettää ja lukea viestejä manuaalisesti. Kuvassa 8 on esimerkki modeemin testauksesta minicom-ohjelmalla.



Kuva 8. Saapunen viestin lukeminen manuaalisesti minicom-sarjaliikenneohjelmalla

Prototyypin testaus

Kun sovelluksen kehityksessä päästiin tarpeeksi pitkälle, prototyyppiä voitiin alkaa testata kuvan 6 mukaisessa testausympäristössä.

Prototyyppiä pyrittiin testaamaan erilaisilla tekstiviesteillä, joita tositilanteessa saattaa tulla vastaan. Useaan osaan jaettujen tekstiviestien lähetys ja vastaanotto toimi ongelmitta. Multimediaviestit eivät käytännön syistä toimineet oikein, mutta eivät myöskään aiheuttaneet ongelmia. Binääriviestit hylättiin oletuksen mukaisesti.

Testeissä kuitenkin huomattiin, että matkapuhelimet saattavat käyttää oletuksesta poikkeavaa merkistöä. Esimerkiksi yhdessä testiviestissä oli käytössä UCS2-merkistö (Universal Character Set), jonka käsittely aiheutti aluksi ongelmia. Ongelma saatiin kuitenkin korjattua ja kaikki suomen kielessä käytetyt aakkoset sekä puhelimen tarjoamat erikoismerkit todettiin toimiviksi. Ainut poikkeus sääntöön oli €-merkki, jota ei ole Sijaishäly-järjestelmän käyttämässä ISO 8859-1 -merkistössä lainkaan [17].

Prototyyppi selvisi tekstiviestien käsittelystä hyvin, mutta testeissä ilmeni yksi vakava puute. Ohjelmisto osoittautui testeissä liian hitaaksi vaatimuksiin nähden, koska se kykeni käsittelemään vain yhden viestin kerrallaan. Prototyypin pitäisi kyetä hyödyntämään useaa rinnakkaista modeemia, joten viestien käsittelyä muutettiin. Paranneltu säikeitä käyttävä prototyyppi oli testien perusteella riittävän nopea.

Vikasietoisuutta testattaessa törmättiin ongelmaan vanhemman Linux-ytimen version kanssa, jota käyttäessä modeemin irroittaminen kesken kaiken saattoi kaataa koko käyttöjärjestelmän. Tämä ongelma kuitenkin korjaantui käyttämällä uudempaa ydintä.

6 Jatkokehittely

Insinöörityössä kehitetty prototyyppi suorittaa tekstiviestien välityksen, mutta se ei ole vielä valmis tuotantokäyttöön. Testejä tuotantojärjestelmän kanssa ei ole suoritettu, koska viestiyhteys Sijaishäly-järjestelmässä on salattu. Prototyyppiin tulee lisätä viestiyhteydessä käytetty salaus. Kun salaus on saatu toteutettua, pitää koko järjestelmä testata uudestaan.

Koska välittimen pitää pystyä toimimaan itsenäisesti, tulee sen myös osata raportoida vakavista virhetilanteista automaattisesti. Prototyyppivaiheessa ei vielä tarkkaan tiedetä, minkälaisiin virhetilanteisiin välitin saattaa tuotantojärjestelmän kanssa joutua. Työnantaja ei myöskään ole määritellyt, minkälaisista virheistä tulee raportoida, joten ominaisuutta ei vielä toteutettu prototyyppiin.

Tarvittaessa välitin voisi myös lähettää säännöllisiä raportteja tilastaan ylläpitäjälle. Tämän tyyppisestä ominaisuudesta ei sovittu prototyypin yhteydessä, mutta se voidaan toteuttaa jatkokehittelyn yhteydessä.

Koska välittimeen ei normaalisti pääse käsiksi ulkoapäin lainkaan, ohjelmistoon pitäisi kehittää jokin tapa, jolla ylläpitäjä voisi muodostaa etäyhteyden haluttuun välittimeen.

7 Yhteenveto

Työssä suunniteltiin ohjelmisto tekstiviestien välityslaitteeseen. Välitinlaitteena käytettiin minikannettavaa, johon asennettiin Linux-käyttöjärjestelmä ja liitettiin GSM-modeemit. Käyttöjärjestelmään valittiin sopivat ohjelmistot ja kokonaisuudesta puuttuvat komponentit ohjelmoitiin itse. Testausta varten koottiin testiympäristö, johon ohjelmoitiin palvelinsimulaattori. Välittimen toimintaa testattiin testausympäristössä. Testeissä tutkittiin, miten tekstiviestien välitys kumpaankin suuntaan onnistuu.

Testeissä havaittiin, että modeemit ja niiden ohjaus toimi halutulla tavalla. SMS Server Tools 3 oli onnistunut valinta modeemien käyttörajapinnaksi, koska se selvisi kaikista tekstiviestitesteistä eikä sen käyttöönotto aiheuttanut ongelmia.

Ohjelmointikieleksi valittu Python osoittautui hyväksi valinnaksi. Sen omaksuminen oli helppoa, ja kielelle ohjelmointi oli nopeaa. Python ei asettanut ylimääräisiä teknisiä rajoituksia sovellukselle, vaan se voitiin toteuttaa alkuperäisen suunnitelman mukaisesti. Havaittiin myös, ettei sovelluksen siirrettävyydessä esiintynyt ongelmia, kun sitä testattiin muissa ympäristöissä.

Linuxin asennus minikannettavalle oli helppoa. Laite saatiin käyttökuntoon lyhyessä ajassa. Välitinohjelmistojen asennus minikannettavalle oli suoraviivaista ja Python-sovellus toimi sillä ongelmitta. Testeissä nähtiin, että suorituskyvyltään minikannettava on enemmän kuin tarpeeksi tehokas välittimeksi.

Työn tarkoituksena oli tutkia, voidaanko välitinlaitteena käyttää halpaa minikannettavaa. Asetetut toimintavaatimukset saavutettiin. Välitinlaitte on halpa, eikä siinä käytetystä käytetystä ohjelmistosta tarvitse maksaa lisenssimaksuja, joten laitteiston lisäksi ylimääräisiä kustannuksia ei kertynyt.

Insinöörityössä tehty prototyyppi osoitti, että työnantajan vaatimuksilla ja annetulla laitteistolla voidaan tehdä toimiva välitin.

Lähteet

1 Hakupaja. 2007. (WWW-dokumentti.) Hakupaja Oy. <https://www.hakupaja.fi/index.html>. Luettu 28.10.2009

2 Sijaishäly. 2007. (WWW-dokumentti.) Hakupaja Oy. <http://www.hakupaja.fi/fi/sijaishaly.html>. Luettu 28.10.2009.

3 Turunen, Vesa. 7.9.2006. Sijainen viestin päässä. (WWW-dokumentti.) <http://hakupaja.fi/fi/uutiset_files/Tehy_artikkeli.html>. Luettu 28.10.2009.

4 Johnson, Michael K., Troan, Erik W.. Linux Application Development, 2nd Edition. Upper Saddle River, USA: Pearson Education, 2005.

5 Kasvi, Keijo. 2009. SMS Server Tools 3. (WWW-dokumentti.) <http://smstools3.kekekasvi.com/>. Luettu 10.11.2009.

6 Kasurinen, Jussi Pekka. Python 3 ohjelmointi, 1. painos. Jyväskylä: WSOY, 2009.

7 Native POSIX Thread Library. 2010. (WWW-dokumentti.) Wikipedia. <http://en.wikipedia.org/wiki/Native_POSIX_Thread_Library>. Luettu 9.4.2010.

8 Kasvi, Keijo. 2009. Slideshow, watch this first. (WWW-dokumentti.) <http://smstools3.kekekasvi.com/index.php?p=s1>. Luettu 10.11.2009.

9 Martelli, Alex. Python in a Nutshell, 2nd Edition. Sebastopol, USA: O'Reilly Media, 2006.

10 Why is Python a dynamic language and also a strongly typed language. 15.11.2008.(WWW-dokumentti.) The Python Wiki. <http://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language>. Luettu 11.3.2010.

11 Python. 2010. (WWW-dokumentti.) Wikipedia. <http://fi.wikipedia.org/wiki/Python>. Luettu 11.3.2010

12 Eclipse (software). 2010. (WWW-dokumentti.) Wikipedia. <http://en.wikipedia.org/wiki/Eclipse_%28software%29>. Luettu 11.3.2010.

13 Kasvi, Keijo. 2009. SMS file format. (WWW-dokumentti.) <http://smstools3.kekekasvi.com/index.php?p=fileformat>. Luettu 10.11.2009.

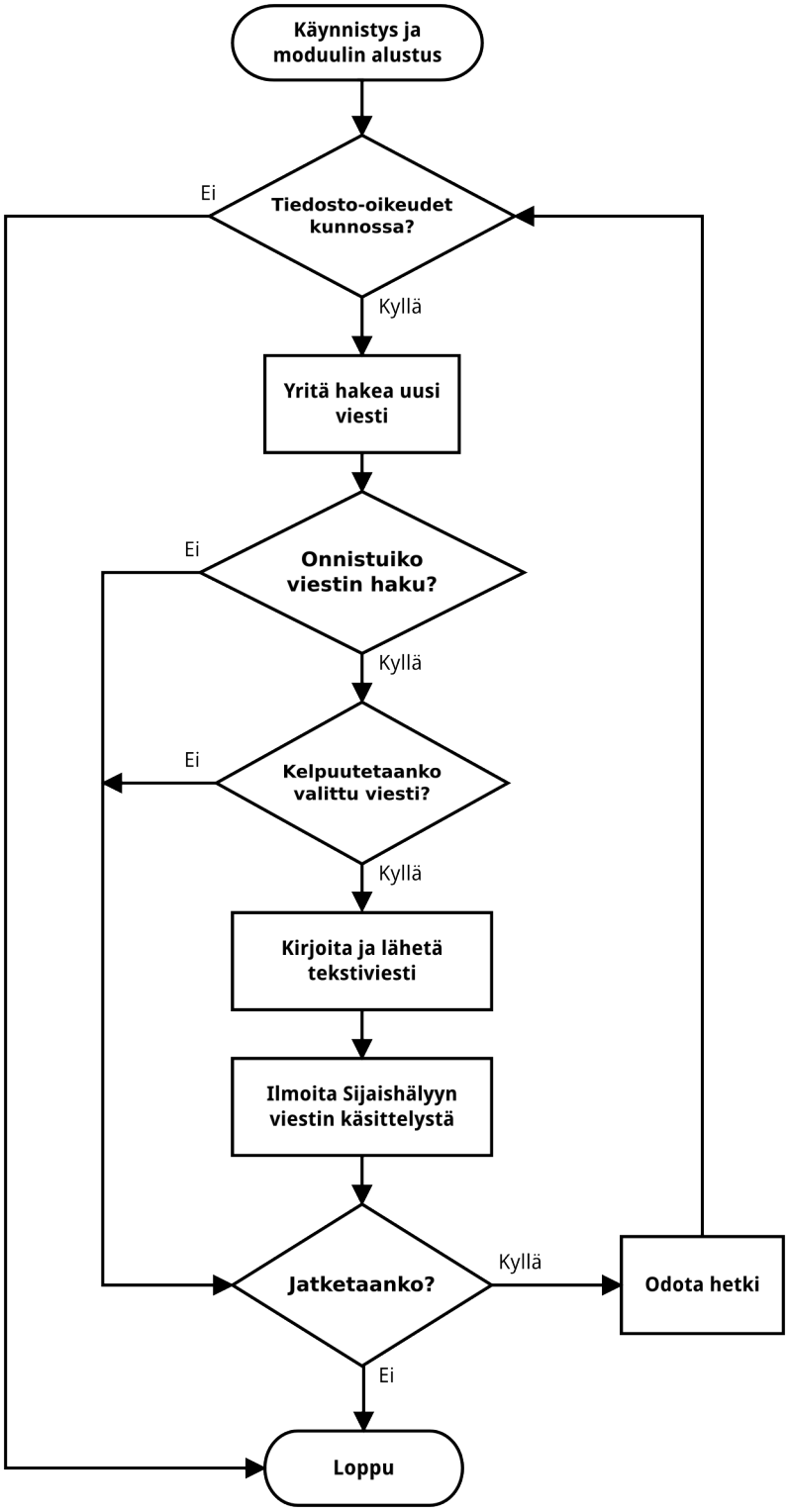
14 Higher-level threading interface. 2010. (WWW-dokumentti.) Python v2.6.5 documentation. <http://docs.python.org/library/thread.html>. Luettu 2.2.2010.

15 Logging facility for Python. 2010. (WWW-dokumentti.) Python v2.6.5 documentation. <http://docs.python.org/library/logging.html>. Luettu 19.1.2010.

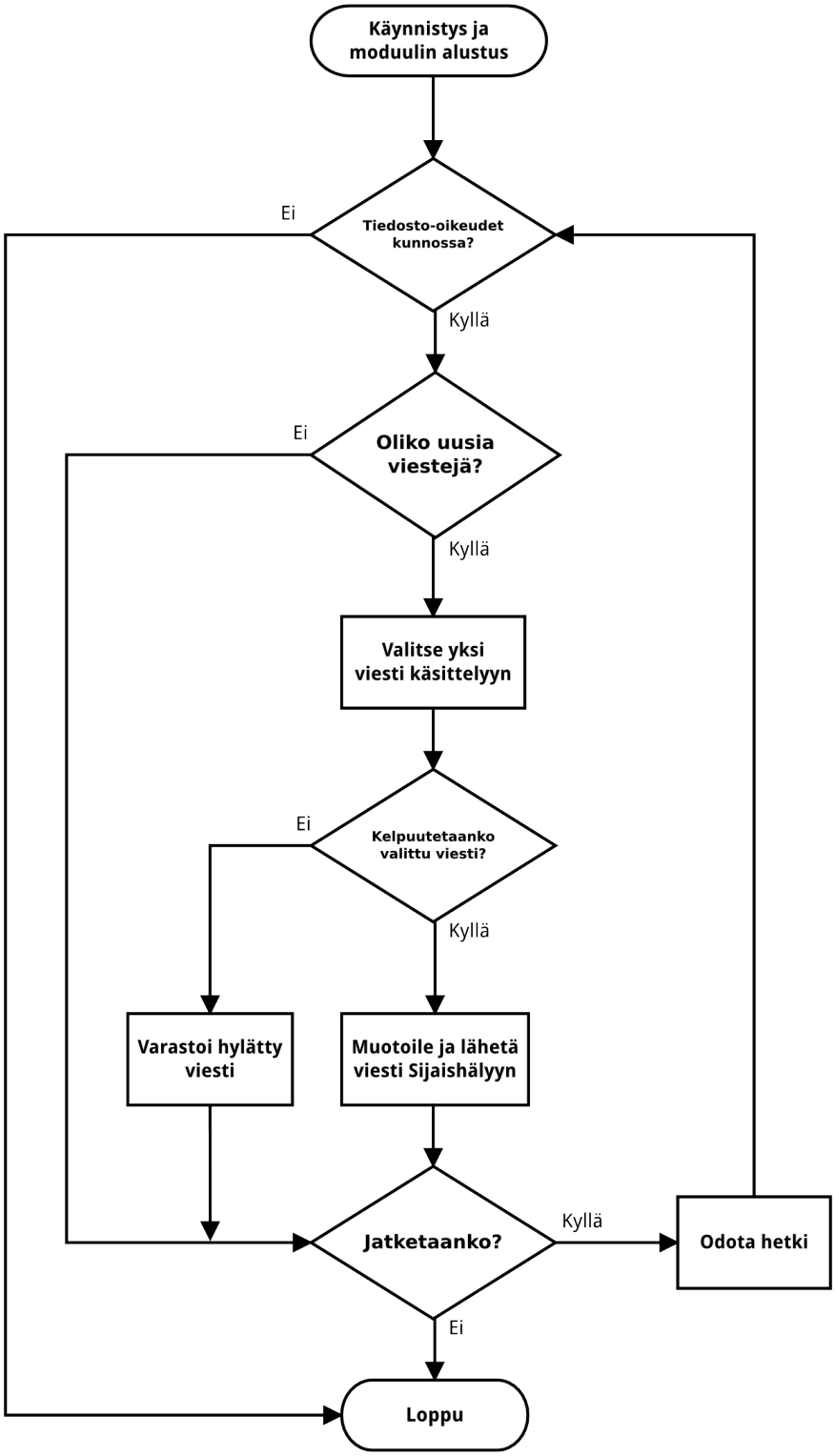
16 Pilgrim, Mark. Dive Into Python 3. 2010. (WWW-dokumentti.) <http://diveintopython3.org/>. Luettu 20.3.2010.

17 ISO/IEC 8859-1. 2010. (WWW-dokumentti.) <http://en.wikipedia.org/wiki/ISO/IEC_8859-1>. Luettu 11.3.2010.

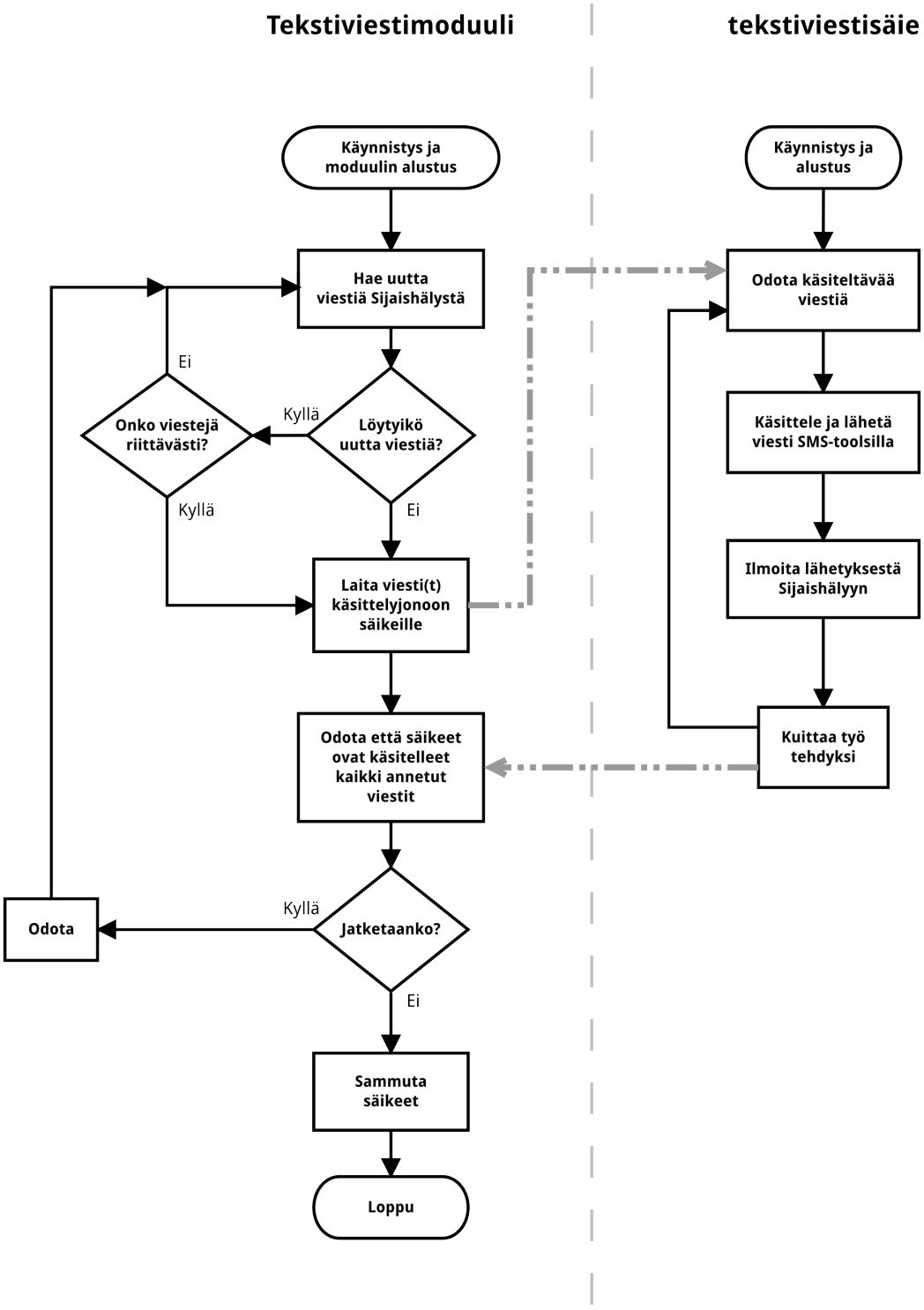
Liite 1: Tekstiviestimoduulin vuokaavio



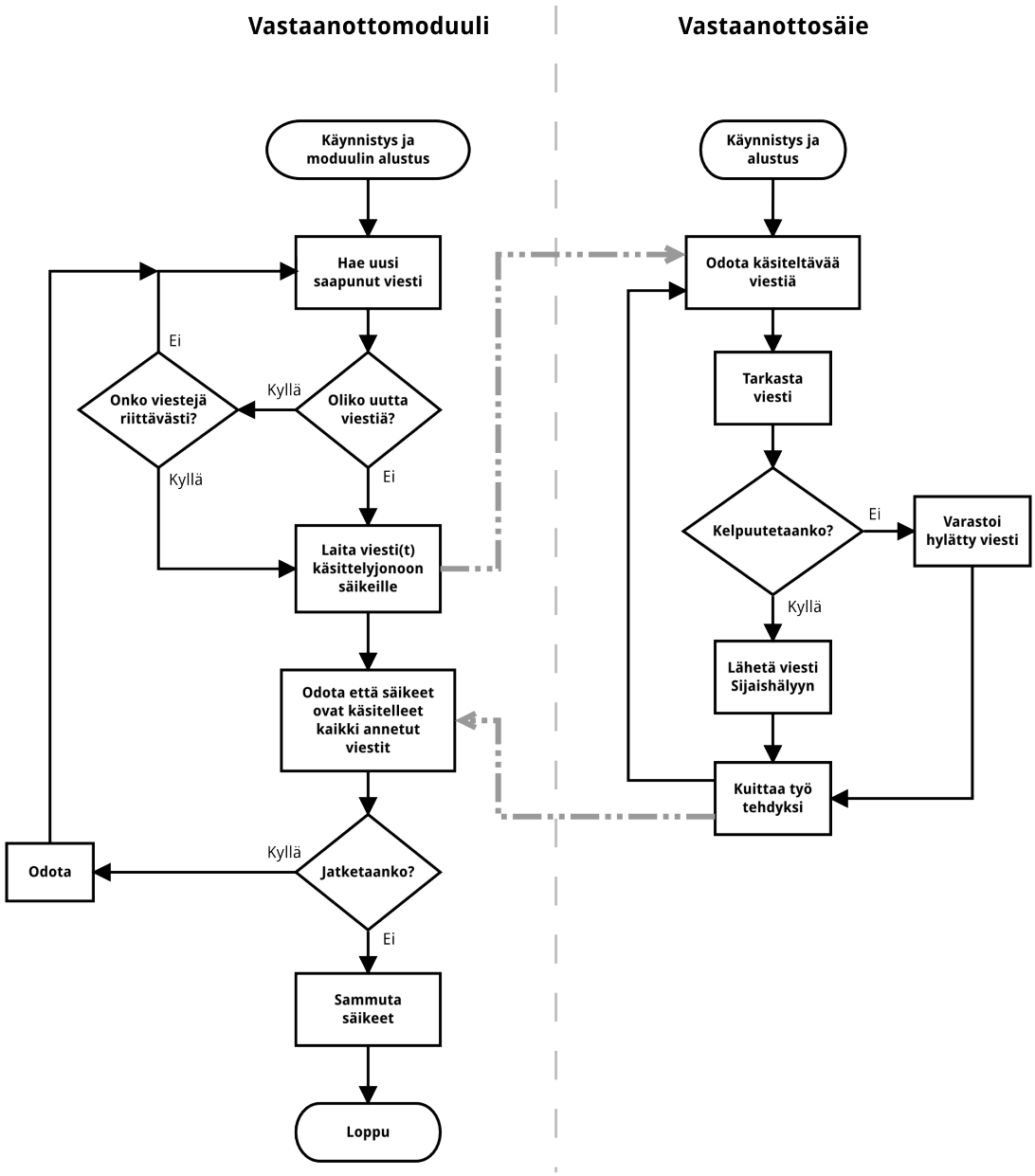
Liite 2: Vastaanottomoduulin vuokaavio



Liite 3: Säikeistetyn tekstiviestimoduulin vuokaavio



Liite 4: Säikeistetyn vastaanottomoduulin vuokaavio



```
"""
Created on Nov 19, 2009
@author: Vesa Ruohonen

@summary: Generic Linux daemon class from an example presented at:
http://www.jejik.com/articles/2007/02/a_simple_unix_linux_daemon_in_python/
"""

import sys, os, time, signal

class Daemon(object):
    """
    A generic daemon class.
    Usage: subclass the daemon class and override the run() method.
    """

    def __init__(self, pidfile, redir = True):
        self.pidfile = pidfile
        self.redir = redir

    def daemonize(self):
        """
        Deamonize class. UNIX double fork mechanism.
        """

        try:
            pid = os.fork()
            if pid > 0:
                # exit first parent
                sys.exit(0)
        except OSError as err:
            sys.stderr.write('fork #1 failed: {0}\n'.format(err))
            sys.exit(1)

        # decouple from parent environment
        os.chdir('/')
        os.setsid()
        os.umask(0)

        # do second fork
        try:
            pid = os.fork()
            if pid > 0:

                # exit from second parent
                sys.exit(0)
        except OSError as err:
            sys.stderr.write('fork #2 failed: {0}\n'.format(err))
            sys.exit(1)

        # redirect standard file descriptors
        sys.stdout.flush()
        sys.stderr.flush()
        si = open(os.devnull, 'r')
        so = open(os.devnull, 'a+')
        se = open(os.devnull, 'a+')

        if self.redir:
            os.dup2(si.fileno(), sys.stdin.fileno())
            os.dup2(so.fileno(), sys.stdout.fileno())
            os.dup2(se.fileno(), sys.stderr.fileno())

        # write pidfile
        pid = str(os.getpid())
        with open(self.pidfile,'w+') as f:
            f.write(pid + '\n')

    def delpid(self):
        os.remove(self.pidfile)

    def start(self):
        """
        Start the daemon.
        """
```

```
        # Check for a pidfile to see if the daemon already runs
        try:
            with open(self.pidfile,'r') as pf:
                pid = int(pf.read().split('\n')[0].strip())

        except IOError:
            pid = None

        if pid:
            message = "pidfile {0} already exist. " + \
                    "Daemon already running?\n"
            sys.stderr.write(message.format(self.pidfile))
            sys.exit(1)

        # Start the daemon
        self.daemonize()
        self.run()

    def stop(self):
        """
        Stop the daemon.
        """

        # Get the pid from the pidfile
        try:
            with open(self.pidfile,'r') as pf:
                pid = int(pf.read().strip())
        except IOError:
            pid = None

        if not pid:
            message = "pidfile {0} does not exist. " + \
                    "Daemon not running?\n"
            sys.stderr.write(message.format(self.pidfile))
            return # not an error in a restart

        # Try killing the daemon process
        try:
            while 1:
                os.kill(pid, signal.SIGTERM)
                time.sleep(0.1)
        except OSError as err:
            e = str(err.args)
            if e.find("No such process") > 0:
                if os.path.exists(self.pidfile):
                    os.remove(self.pidfile)
            else:
                print (str(err.args))
                sys.exit(1)

    def restart(self):
        """
        Restart the daemon.
        """
        self.stop()
        self.start()

    def run(self):
        """
        You should override this method when you subclass Daemon.

        It will be called after the process has been daemonized by
        start() or restart().
        """
```

Liite 6: Moduulilataajan lähdekoodi


```
#!/usr/bin/python
"""
Created on Nov 25, 2009

@author: Vesa Ruohonen

@summary: Python module loader
"""

import sys, time

if __name__ == '__main__':

    # Remove self from commandline arguments
    sys.argv[0:1] = []
    module_class = None
    module_name = ''

    # Use the first argument as module name to load
    if len(sys.argv) > 0:
        try:
            # Try to import module file and module class name from it
            module_init = __import__(str(sys.argv[0]), globals(), locals(), [], -1)
            module_class = module_init.__getattribute__(module_init.getmodulename())
            module_name = sys.argv[0]
            sys.argv[0:1] = []

        except (ImportError, AttributeError, SyntaxError) as err:
            sys.exit('Error loading module: {0}\n'.format(err))

    else:
        sys.exit('Error: No module specified')

    if module_class != None:
        try:
            # Instance module class as an object
            module = module_class()

            # Pass startup arguments to module object
            if len(sys.argv) > 0:
                module.startup_path = sys.argv[0]

            # Start the module
            module.start()

        except AttributeError as err:
            sys.exit('Error: {0}\n'.format(err))
    else:
        sys.exit('Error: No module loaded')

    sys.exit(0)
```