Ari Joki

TELIT EZ-10 -GPRS-MODEEMI DATAN EDELLEENLÄ-HETTÄJÄNÄ CINET-SENSORIVERKOSTA

Opinnäytetyö
KESKI-POHJANMAAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma
Toukokuu 2010

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ

Yksikkö	**Aika**	**Tekijä/tekijät**
Tekniikan ja liiketalouden yksikkö Kokkola	07.05.2010	Ari Joki
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma		
Työn nimi Telit EZ-10 -GPRS-modeemi datan edelleenlähettäjänä CINET-sensoriverkosta		
Työn ohjaaja Kauko Kolehmainen		**Sivumäärä** 33+1
Työelämäohjaaja Petri Saviranta		

Työ on toteutus datan edelleenlähetyksestä sensoriverkosta langattomasti palvelimelle käyttäen GSM-verkkoa hyödyntävää modeemia. Sensoriverkko on joukko langattomia lähetinyksikköjä, joihin on liitetty sensoreita kuten lämpötila- ja kosteudenmittausantureita.

Sensoriverkon keräämä data siirtyy langattomasti Yliopistokeskus Chydeniuksen kehittämälle tiedonkeruulaitteelle. Tiedonkeruulaitteelta data siirretään Telit EZ-10 -GPRS-modeemille sarjaportin välityksellä. Työ keskittyy Telit EZ-10 -GPRS-modeemiin ja sille kehitettyyn Python Easy Script -ohjelmaan, joka hoitaa datan vastaanoton ja edelleenlähetyksen halutulle palvelimelle.

Asiasanat

GPRS, langattomuus, Python EasyScript

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	**Date** 07.05.2010	**Author** Ari Joki

Degree programme
Business Information Technology

Name of thesis
Telit EZ-10 GPRS-Modem as a Data Forwarder from CINET Sensor Network

Instructor Kauko Kolehmainen	**Pages** 33+1 Appendix

Supervisor
Petri Saviranta

The purpose of the study was to implement data forwarding wirelessly from a sensor network to a server via a modem, which uses GSM-network. A sensor network is a group of wireless measurement units, which contains a sensor, such as a temperature sensor or a moisture sensor, packed with a transmitter.

Data collected by the sensor network is transmitted to a device made by the University Centre Chydenius. The purpose of this device is to gather the incoming data and to supply it for Telit EZ-10 -GPRS-modem via a serial port. The study focuses on the Telit EZ-10 -GPRS-modem and on Python Easy Script programming on it to fulfill the data forwarding to the server.

Key words

GPRS, wireless communication, Python EasyScript

LYHENTEET

AT-komento Komentoprotokolla sarjaportin kautta suoritettaviin komentoihin, joilla joko pyydetään laitteelta jotain tietoa tai suoritetaan jokin operaatio.

GPRS General Packet Radio Service, GSM-verkossa toimiva langaton tiedonsiirtopalvelu.

GSM Global System for Mobile Communications.

M2M Mobile to Machine, viitataan tiedonsiirtoon langattomasti vastaanottavaan järjestelmään.

RS-232 Standardi sarjamuotoinen tiedonsiirtotapa kahden laitteen välillä.

Www-palvelin
Internetiin yhteydessä oleva tietokone, jossa on palvelinohjelmisto.

ASCII American Standard Code for Information Interchange.

SLIP Serial Line Interface Protocol, kapselointimenetelmä IP-paketeille sarjaportin kautta tapahtuvassa tiedonsiirrossa.

1 JOHDANTO

Työn tarkoitus oli kehittää Python-sovellus Telit EZ-10 -GPRS-moduulille. Moduulin tehtävänä on toimia tiedonkerääjänä sensoriverkosta ja datan lähettäjänä langattomasti palvelimelle. Työn tilaajana oli Kokkolan Yliopistokeskus Chydenius. Yliopistokeskuksella oli projektin osa, jossa tarvittiin langaton ratkaisu sensoriverkkoon, joka koostuu muun muassa lämpötilaa ja kosteutta mittaavista langattomasti toimivista sensoreista sekä laitteesta, joka ottaa langattomasti vastaan sensorien lähettämän datan. Tehtäväni oli rakentaa tuon vastaanotettavan datan käsittely ja edelleenlähetys langattomasti palvelimelle, jossa yliopistokeskuksen Java-ohjelma ottaa sen taas vastaan.

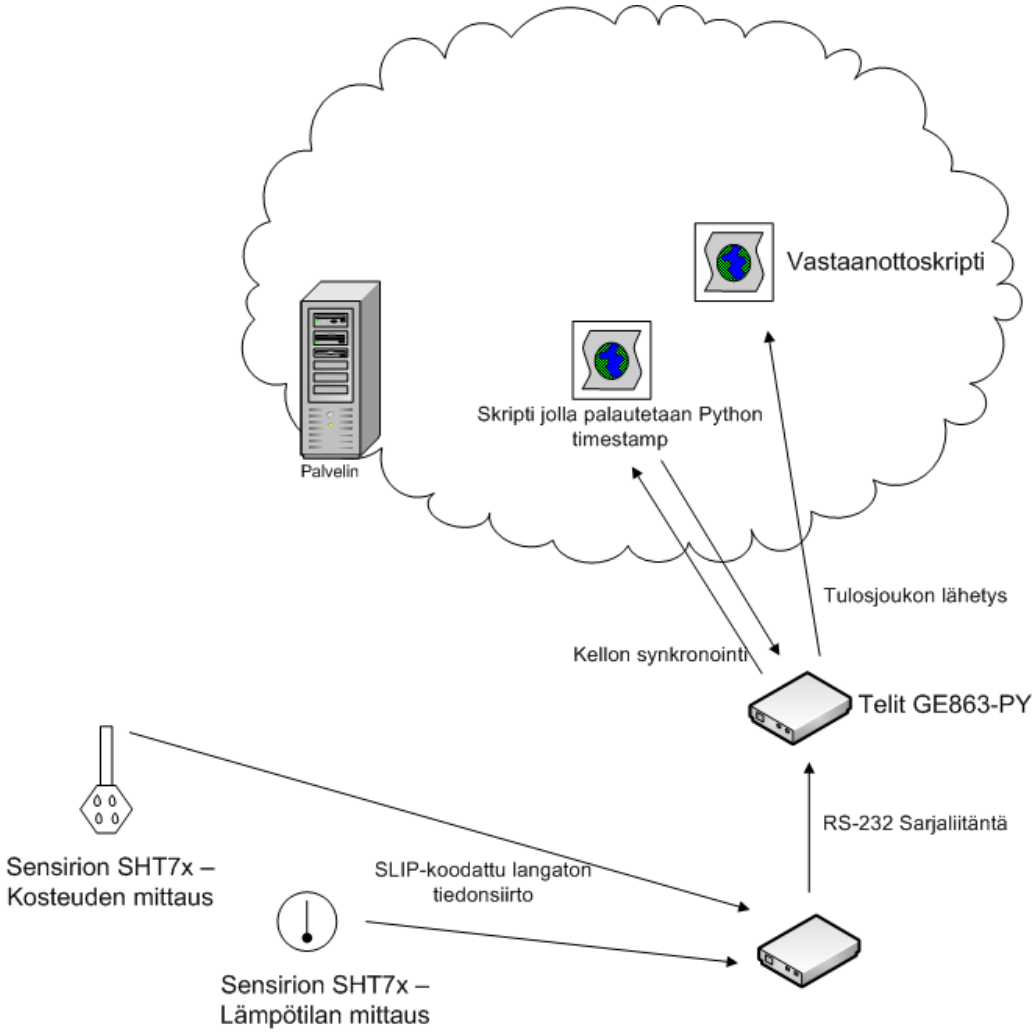
TelitEZ-10 -GPRS-moduuli on langattomaan tiedonsiirtoon ja GPS-paikannukseen tarkoitettu laite, joka perustuu GSM-teknologialle ja toimii käytännössä kuten tavallinen GSM-puhelin GPS- ja GPRS-ominaisuuksilla. Laite on ohjattavissa AT-komennoilla, sekä laitetta varten kehitetyllä Python EasyScript -ohjelmointikielellä. Sen kaikkia ominaisuuksia voidaan ohjata ohjelmallisesti, mikä tekee siitä sopivan vaihtoehdon opinnäytetyöhöni. Tärkein ominaisuus on kuitenkin se, että laite mahdollistaa tietojen vastaanoton sarjaliitännän kautta (RS-232), joten Chydeniuksen tiedonkeruulaite kykenee kommunikoimaan sarjakaapelin kautta moduulin kanssa.

Raportissa työn osa-alueet on jaettu kolmeen pääosioon: laitteisto, kehitystyö ja toteutus. Luvussa laitteisto käsitellään työssä käytettävän laitteiston taustaa ja kunkin laitteen ominaisuuksia. Kehitystyö-osiossa kerrotaan, mitä ohjelmia työn toteutukseen tarvitaan, sekä taustaa käytetystä ohjelmointikielestä ja ohjelmointimenetelmistä. Toteutus-vaihe on jaettu neljään alakategoriaan: määrittely, suunnittelu, ohjelmointi ja testaus. Niissä käsitellään työn varsinaista tekovaihetta, työn teossa ilmenneitä ongelmia sekä ratkaisuja. Lopuksi koostetaan aiemmin käsitellyt asiat tiiviisti yhteenvedossa, esitellään tekijän omia pohdintoja työstä sekä ehdotuksia mahdollisista jatkotutkimuksista.

2 LAITTEISTO

Työssä käytetty laitteisto sisältää tiedonkeruulaitteen, jonka on valmistanut yliopistokeskus Chydenius. Tämän laitteen tehtävänä on kerätä dataa sensoreilta, jotka mittaavat esimerkiksi lämpötilaa tai ilman suhteellista kosteutta. Sensorit on kytketty langattomiin lähettimiin, eli tiedonkulku on langatonta sensoreiden ja tiedonkeruulaitteen välissä. Sensoreiden ja tiedonkeruulaitteen muodostamaa kokonaisuutta kutsutaan sensoriverkon noodiksi. Noodeja voisi olla useampiakin, mutta tässä työssä noodien lukumäärä ei ole oleellinen. Laitteiston oleellisin osa tässä työssä on Telit EZ-10-GPRS-modeemi, joka toimii datan edelleenlähettäjänä. Tässä työssä kyseistä modeemia kutsutaan myös moduuliksi. Moduuli vastaanottaa datan tiedonkeruulaitteelta sarjaportin välityksellä käyttäen RS-232 -standardia ja lähettää sen aikaleimalla varustettuna palvelimelle halutun kokoisina paketteina. Paketilla tässä tapauksessa tarkoitetaan sitä määrää, kuinka monta yhdeltä sensorilta tullutta yksittäistä informaatiopakettia edelleenlähetetään kerralla.

Kuviossa 1 esitetään laitteiston välinen vuorovaikutus. Langattomiin lähettimiin kytketyt anturit lähettävät SLIP-koodattua dataa tiedonkeruulaitteelle, joka vastaanottaa datan ja syöttää sen sarjaporttia pitkin Telitin GPRS-modeemille. GPRS-modeemi synkronoi sisäisen kellonsa käynnistyessään muodostamalla yhteyden palvelimeen ja kutsumalla PHP-skriptiä, joka palauttaa aikaleiman Python-skriptille modeemin sisällä. Kun sisäinen kello on asetettu, on laite valmis vastaanottamaan dataa tiedonkeruulaitteelta. Dataa vastaanotetaan niin kauan kunnes asetuksissa asetettu raja-arvo datapakettien määrästä on saavutettu. Lopuksi datapaketit lähetetään palvelimelle aikaleiman kera, jossa Java-sovellus ottaa datan vastaan.



KUVIO 1. Järjestelmän laitteiston vuorovaikutus

2.1 Laitteiston valinta

Tiedonvälitykseen soveltuvaa laitetta valitessa tuli ottaa huomioon joitakin seikkoja kuten laitteen ohjelmoitavuus, ohjeiden saatavuus sekä tietenkin se, että se tarjoaisi tarvittavat ominaisuudet kyseisen tutkimusongelman ratkaisemiseksi. Laitteen tulee kyetä vastaanottaa dataa sarjaliitännän kautta ja lähettää se langattomasti palvelimelle paketteina, siten että dataa kerätään puskuriin, kunnes se saavuttaa ennalta määritetyn määrän dataa. Jokaisen paketin tulee sisältää aikaleima ennen varsinaista dataa. Laitteeksi valittiin Telitin GPRS-moduuli, joka on ohjelmoitavissa Python-johdannaisella kielellä. Siinä on sarjaliitäntä, joka mahdollistaa datan vastaanoton datankeruulaitteelta. (Telit 2008.)

2.2 Telit -GPRS-modeemi

Telit EZ-10 -GPRS-modeemi toimii 850/900/1800/1900 MHz:n taajuusalueilla GSM-verkossa mahdollistaen langattoman tiedonsiirron palvelimelle mistä tahansa kohteesta, missä kuuluvuusalue on riittävä. Laite toimii 9-24 V:n tasavirralla ja sen mukana tulee muuntaja 230 voltin verkkovirralle sekä auton tupakansytyttimeen liitettävä verkkojohto. Laitteen toimintalämpötilaksi valmistaja lupaa –30 asteesta +75 asteeseen. Tiedonsiirto modeemin ja tietokoneen välillä tapahtuu USB-RS232-johdolla, joka kytketään laitteen sarjaliitäntään ja vastaavasti tietokoneessa normaaliin USB-liitäntään. Laite on ohjelmoitavissa Python EasyScript -ohjelmointikielellä. Kyseessä on Python-kielen karsittu versio, joka sisältää Telitin moduuleille tarkoitetut lisäkirjastot, joilla ohjataan laitteen toimintaa. Pythonin lisäksi moduulia voidaan ohjata AT-komennoilla joko laitteen ollessa kytkettynä tietokoneeseen tai suoraan ohjelmakoodista. (Telit 2008.)

Kuviossa 2 esitetään metallikuoriin pakattu EZ-10 -GPRS-modeemi, jossa ovat kiinni GSM- ja GPS-antennin johdot. Telitin modeemit voivat olla myös muovikuorilla riippuen tuotemallin ja käyttökohteen mukaan, esimerkiksi kuorma-autoon asennettaessa laitteen tulee olla muovikuorinen.

KUVIO 2. Telit EZ-10 GPRS-modeemi

Kuviossa 3 näkyvät Telit EZ-10 -GPRS-modeemin liitännät ja paikka SIM-kortille. Virtajohto kytketään vasemmalla olevaan liitäntään ja liitäntä on muotoiltu niin, että virhekytkentä ei ole mahdollista. Oikealla ylhäällä on liitännät GPS- ja GSM-antenneille. SIM-kortti työnnetään sisään, ja se pysyy sisällä lukitusmekanismin avulla. Apuna on hyvä käyttää jotain pientä esinettä, esimerkiksi kynää tai viivoitinta. Kortin pois ottaminen tapahtuu samalla tavalla, työntämällä korttia uudestaan sisäänpäin, jolloin mekanismi vapauttaa kortin hieman ulospäin, jolloin se on vedettävissä ulos. Laitteessa käytettävästä SIM-kortista tulee poistaa PIN-koodi ennen käyttöä. Se tapahtuu esimerkiksi sijoittamalla kortti matkapuhelimeen, jonka asetuksissa kortin PIN-koodikyselyn voi ottaa pois.

KUVIO 3. EZ-10 -GPRS-modeemin liitännät

Kuviossa 4. GPRS-modeemin toinen pääty, jossa sijaitsee laitteen merkkivalot, RS 232-sarjaliitäntä sekä AUX-liitäntä. Vihreä PWR-led ilmaisee laitteen käynnissäolon, ja punainen STATUS-led kertoo laitteen tilan tietyllä taajuudella vilkkuen. STATUS-ledin avulla erotetaan esimerkiksi, onko laite AT-komentotilassa, jolloin voidaan esimerkiksi lisätä tai poistaa tiedostoja, vai skriptin ajotilassa, jolloin laite alkaa suorittaa skriptiä, joka sille on asetettu ajoon AT-tilassa. Moduulin saa AT-komentotilaan siten, että virtajohto otetaan irti ja sarjakaapeli kytketään laitteeseen ja annetaan olla kytkettynä, kun virtakaapeli kytketään uudestaan. Vastaavasti ajotilaan asettaminen tapahtuu uudelleenkäynnistyksellä ilman sarjakaapelia. (Telit 2007, 65–66.)

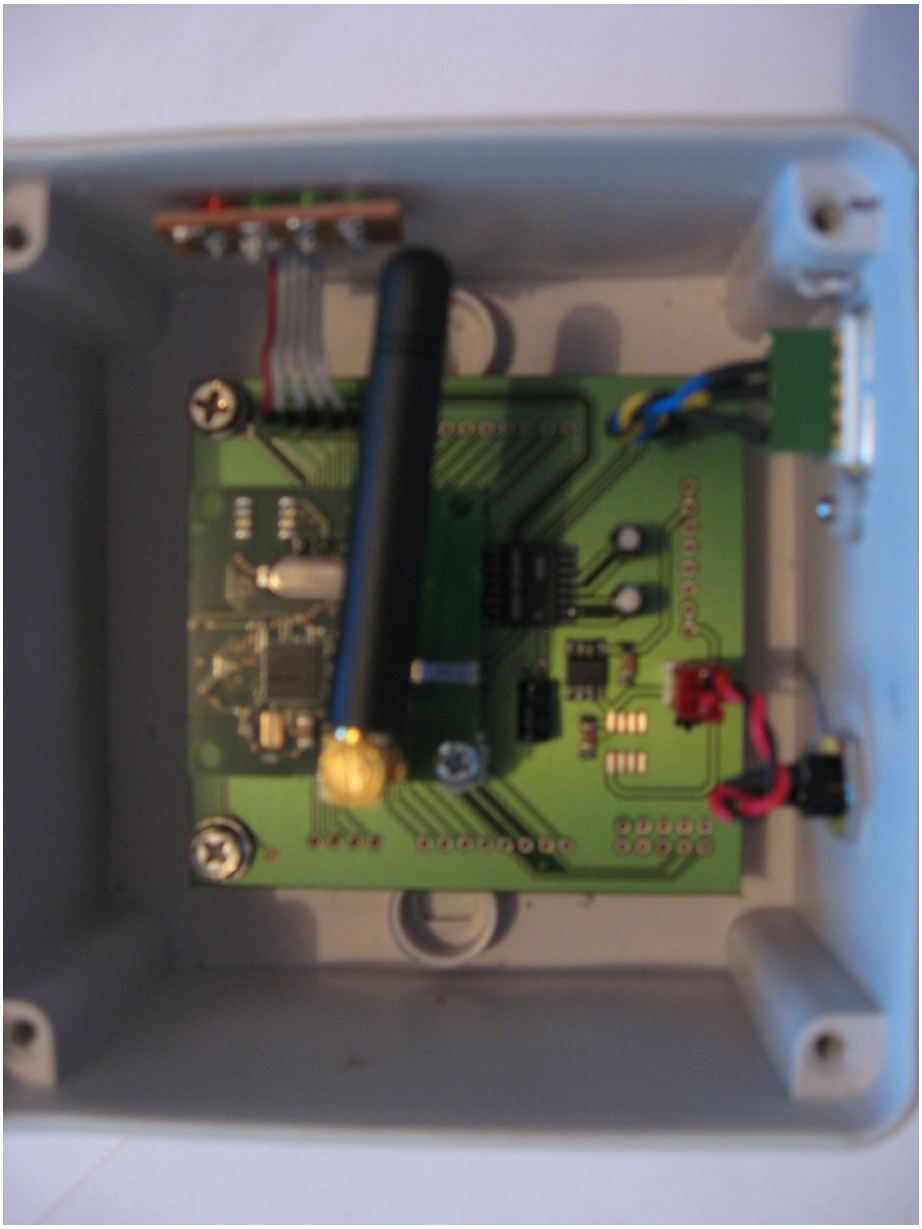
KUVIO 4. Telit EZ-10 -GPRS-modeemin liitännät

GPRS-modeemi liitetään PC-tietokoneeseen RS232-USB-kaapelilla, joka toimitetaan modeemin mukana. Kaapelin mukana toimitetaan myös ajuri, jonka kaapeli tarvitsee toimiakseen. Se asennetaan kuten mikä tahansa muu laitteistoajuri Windows-ympäristössä. Asennuksen jälkeen tietokone käynnistetään uudelleen, ja modeemi on tämän jälkeen käytettävissä AT-komentotilassa PC-tietokoneen välityksellä.

2.3 Chydeniuksen tiedonkeruulaite

Chydeniuksen tiedonkeruulaite on laite, joka kerää dataa langattomilta antureilta ja syöttää datan sarjaportin välityksellä Telitin GPRS-modeemin sarjaporttiin, jossa se otetaan vastaan muistiin puskuriin lähetystä varten. Data on SLIP-koodattua, joka on tosin epäoleellista työni kannalta, koska GPRS-moduulin tehtävä on lähettää data sellaisenaan aikaleiman kera.

Kuviossa 5 tiedonkeruulaite koteloituna IP-65-standardin mukaisesti. Liitännät sijaitsevat kuvan oikeassa reunassa: ylhäällä sarjaliitäntä ja alempana liitäntä virtajohdolle. Vasemmassa yläkulmassa sijaitsevat ledit, joista yksi on punainen ja ilmaisee virran olevan päällä, muut ovat vihreitä ja ilmaisevat laitteen eri toimintoja.

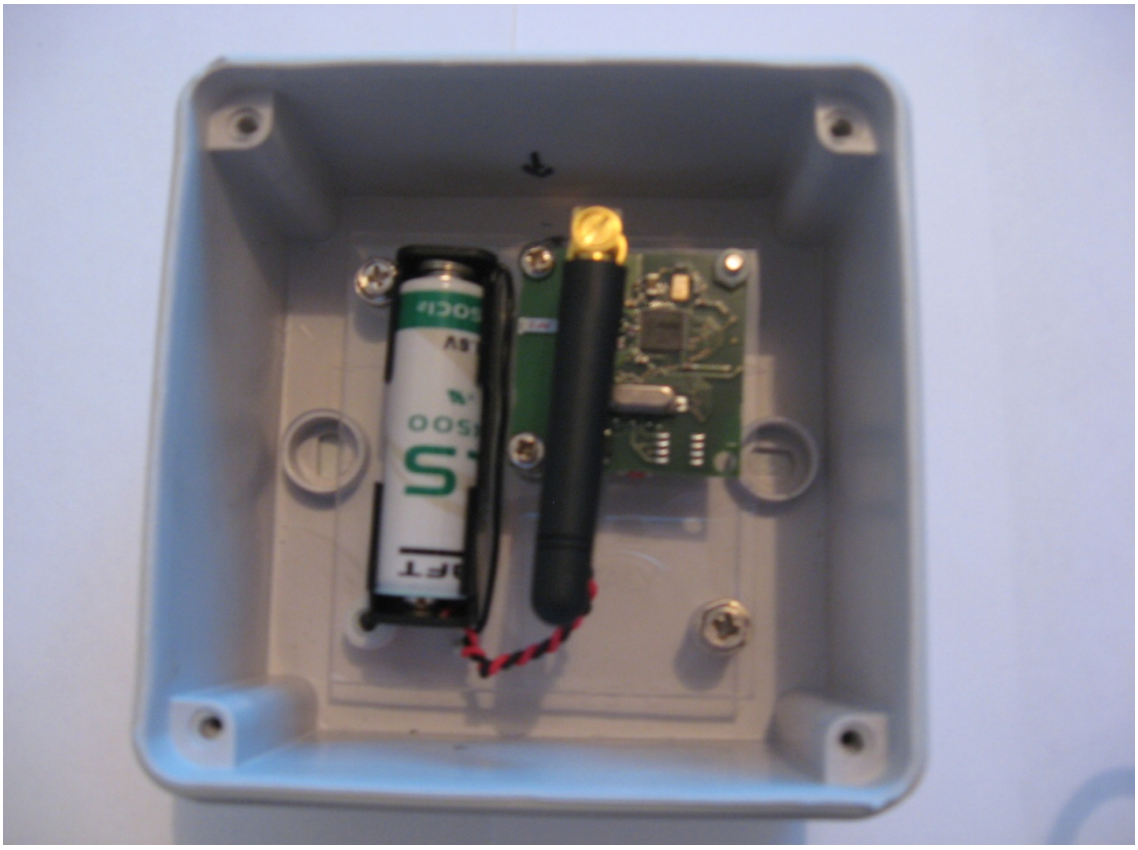


KUVIO 5. Tiedonkeruulaite

2.4 Anturit

Lämpötilan ja kosteuden mittaamiseen käytetään Sensirionin antureita (SHT7x), jotka on koteloitu IP-67 -standardin mukaiseen koteloon langattoman lähettimen ja pienen akun kanssa. Langattoman lähettimen kantama oli joitakin metrejä. Yliopistokeskuksen mukaan lähetystehoa voidaan kuitenkin kasvattaa, jolloin akun kesto tietenkin pienenee. Akun kesto on kuitenkin huomattavan pitkä, jopa vuosia asetuksen mukaan.

Kuviossa 6 on IP-67 -standardin mukaiseen koteloon pakattu Sensirionin lämpötilan mittausanturi, langaton lähetin sekä tarvittavan käyttöjännitteen antava akku. Laite lähettää dataa ennalta säädettyjen arvojen perusteella tietyllä lähetysteholla ja tietyin väliajoin ja tämä määrittää akun keston, joka on joka tapauksessa hyvin pitkä.



KUVIO 6. Lämpötilan mittausyksikkö

2.5 Www-palvelin

Määrittelyssä sovittiin, että Yliopistokeskus Chydenius hoitaa datan vastaanottopuolen omalla palvelimellaan, siten että palvelimella oleva Java-ohjelma kuuntelee tiettyä porttia, jonne data lähetetään Telit-moduulilta. Datan vastaanotosta kirjoitettiin lokia tekstitiedostoon. Tiedosto sisälsi puretun datan, esimerkiksi lämpötilan, sekä palvelimen aikaleiman jokaista vastaanotettua pakettia kohden.

Ohjelman kehitysvaiheessa kuitenkin käytin erillistä palvelinta, johon oli asennettu Apache-palvelinohjelmisto PHP5:n tuella. Tätä palvelinta käytettiin muun muassa vastaanottamaan dataa sellaisenaan tekstitiedostoon PHP-vastaanottoskriptin kautta sekä lähettämään palvelimen aikaleima Telit-moduulille, jossa aikaleiman mukaan asetettiin moduulin sisäinen kello oikeaan aikaan.

3 KEHITYSYMPÄRISTÖ

3.1 Kehitystyökalut

Kehitysympäristönä käytettiin PC-tietokonetta, jossa oli Windows-käyttöjärjestelmä. Ohjelmina kehitystyön apuna käytettiin Script Downloader- ja HyperTerminal-ohjelmia, joista Script Downloader on Telitin valmistama ohjelma tiedostojen siirtoon tietokoneelta EZ-10-moduuliin. Hyper Terminal on Windows XP:n mukana tuleva oletusohjelma tiedonsiirtoon, ja sitä käytettiin AT-komentojen suorittamiseen laitteeseen. Jotta tiedonsiirto tietokoneen ja moduulin välillä toimi, oli asennettava ajurit USB-kaapelille, joka yhdistettiin laitteen sarjaliitäntään sekä tietokoneen USB-liitäntään, jolloin laitteelle voitiin lähettää käskyjä ottamalla yhteys COM-porttiin edellä mainituilla ohjelmilla.

Ohjelmointikielenä käytettiin Pythonia. Sillä ohjataan laitteen kaikkia toimintoja, ja sen mahdollistavat laitteen sisäiseen muistiin asennetut laajennusosat Pythonille. Näiden kokonaisuutta Telit kutsuu nimellä Python EasyScript. Itse ohjelmien kirjoittamiseen käytettiin erinäisiä ohjelmointiin tarkoitettuja tekstieditoreita ja testivaiheessa binääridatan lukemiseen käytettiin siihen tarkoitettua tekstieditoria. Kumpaankin tarkoitukseen on valtavasti eri ohjelmia tarjolla.

3.2 Python-ohjelmointikieli

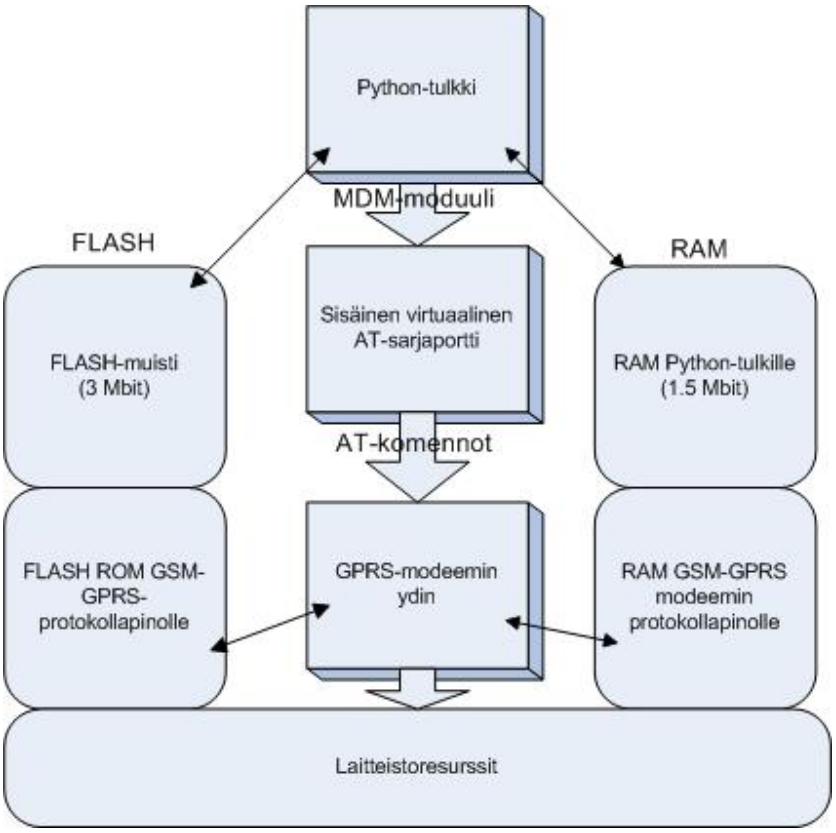
Python on ilmainen avoimen lähdekoodin ohjelmointikieli, joka toimii usealla eri laitealustalla ja käyttöjärjestelmällä. Samaa Pythonin lähdekoodia voidaan periaatteessa ajaa esimerkiksi Windows-, Macintosh- tai Unix-käyttöjärjestelmässä. Python on laajennettava ohjelmointikieli, joka tarkoittaa sitä, että kuka tahansa voi ohjelmoida kieleen laajennuksia, joita voidaan uudelleenkäyttää eri ohjelmistoissa. Pythonin muistinkäyttö on tehokasta ja vähäistä, mikä tekee siitä hyvin soveltuvan pieniin laitteisiin. Python eroaa monesta muusta ohjelmointikielestä siinä, että se on tulkattava ohjelmointikieli, kun vastaavasti esimerkiksi C-kieli on käännettävä ohjelmointikieli. Tämän vuoksi Pythonin koodia ei tarvitse kääntää ensin binäärikoodiksi, josta luotaisiin käynnistettävä tiedosto, vaan Python-tulkki suorittaa lähdekoodin suoraan. (Teller 2007, 1, 5.)

Telitin moduuleissa Pythonia on karsittu siten, että se tukee vain seuraavia Pythonissa yleensä mukana olevia moduuleita, niin kutsuttuja ydinmoduuleita*: marshal, imp, __main__, __builtin__, sys, md5*. Muut moduulit eivät ole tuettuja. Tukea tietotyypeille on karsittu *complex-, float-* ja *docstring*-tietotyyppien osalta. Rajoituksia on muitakin muun muassa loogisten operaattoreiden osalta, joskin ne ovat melko hyvin tuettuja. (Telit 2007, 79.)

3.3 Python EasyScript

EasyScript Python-laajennus mahdollistaa Telitin GPRS-moduulien ohjaamisen sisäisesti Python-ohjelmakoodista. EasyScript-ajattelun tarkoituksena on hoitaa ohjelmallisesti moduulin sisällä samat asiat, joita on aiemmin tehty mikrokontrollerin avulla. Kun aiemmin erillinen mikrokontrolleri on ohjannut IO-pinnejä ja moduulia AT-komentojen avulla, on nykyperiaate EasyScriptin kanssa sellainen, että moduulin sisällä ohjelmakoodi on yhteydessä MDM-laajennuskirjaston kautta virtuaaliseen AT-sarjaporttiin, jonka välityksellä AT-komennot suoritetaan GPRS-modeemissa. (Telit 2007, 8.)

Kuviossa 7 esitetään Telit-moduulin toimintaperiaate AT-komentojen kannalta ja muistiresurssien jakautuminen. Python-tulkille, joka tulkkaa ajettavan skriptin, on varattu 1.5 megabittiä RAM-muistia ja skriptejä varten FLASH-muistia on 3 megabittiä. Muistien määrät kuulostavat pieniltä, mutta ovat hyvin riittäviä sovelluksiin joihin laite on tarkoitettu. Python-tulkki kutsuu virtuaalista AT-sarjaporttia kun skriptissä käytetään MDM-luokasta löytyvää send()-funktiota, ja AT-sarjaportti jakaa käskyn GPRS-modeemin ytimelle, jolloin send()-funktion kautta antama AT-komento suoritetaan. (Telit 2007, 9.)



KUVIO 7. Telit-moduulin toimintaperiaate (mukaillen Telit 2007, 9.)

3.4 Telitin Python-kirjastot

Telit EasyScript tarjoaa Python-ohjelmointikieleen laajennuksia GPRS-moduulin toimintojen ohjelmointiin. Esittelen muutamia luokkia ja niiden funktioita, joita käytän työssäni.

MDM-luokka toimii rajapintana Pythonin ja moduulin AT-komentojäsentäjän välissä. Jos Python-skriptiltä halutaan suorittaa AT-komentoja, käytetään MDM-luokkaa. Luokkaa käytetään sekä komennon lähettämiseen että paluuviestin lukemiseen skriptissä. Luokka pitää sisällään myös funktioita, joilla tarkistetaan AT-komentojäsentäjän asetuksia. (Telit 2007, 20–24.)

Kuviossa 8 tuodaan luokka MDM käyttöön komennolla *import MDM*. Rivillä 2 lähetetään niin sanottu tyhjä AT-komento komentojäsentäjälle (*MDM.send('AT', 0)*) ja rivillä 3 lähetetään heksadesimaaliarvo *0x0d*, joka on desimaaleissa 13, joka vastaavasti ASCII-merkistössä vastaa Enter-näppäintä. Rivillä neljä komentojäsentäjän lähettämä paluuviesti sijoitetaan muuttujaan. *MDM.receive()*-funktiokutsussa oleva numeroarvo 10 tarkoittaa yhtä sekuntia, jonka verran skripti odottaa vastausta AT-komentojäsentäjältä.

```
1 import MDM
2 MDM.send('AT', 0)
3 MDM.sendbyte(0x0d, 0)
4 paluuviesti = MDM.receive(10)
```

KUVIO 8. MDM-luokka

SER-luokka on tehty rajapinnaksi Pythonin ja moduulin sarjaportin välille. Kun skriptissä halutaan kommunikoida sarjaportin kanssa, käytetään SER-luokan *read()*- ja *send()*-funktioita. Luokka sisältää myös erinäisiä asetusten lukufunktioita jotka palauttavat arvonaan tosi tai epätosi. (Telit 2007, 30–34.)

Kuviossa 9 tuodaan rivillä 1 luokka SER käyttöön komennolla *import SER*. Rivillä 2 asetetaan sarjaportin nopeudeksi 112 800 bittiä sekunnissa. Rivi 3 lähettää sarjaporttiin merkkijonon ”hei maailma”, ja rivi 4 lähettää Enter-näppäimen painallusta vastaavan ASCII-koodin. Paluuviesti sarjaportilta luetaan funktiolla *receive()*, jossa timeout-arvoksi on asetettu puoli sekuntia.

```
1 import SER
2 SER.set_speed('112800')
3 SER.send('hei maailma')
4 SER.sendbyte(0x0d)
5 paluuviesti = SER.receive(5)
```

KUVIO 9. SER-luokka

GPIO-luokka toimii Pythonin ja moduulin sisäisen Input/Output-käsittelijän välillä. Sen avulla voidaan esimerkiksi asettaa jonkin fyysisen pinnin arvo päälle tai pois päältä tai vastaavasti lukea pinnin tila. GPIO-luokka sisältää myös mm. funktion, jolla nähdään, onko jokin pin lähettämässä vai vastaanottamassa dataa. Luokalla ohjataan myös moduulin led-valoja, tarkkaillaan moduulin akun tilaa sekä tarkkaillaan I/O-pinnien jännitteitä. (Telit 2007, 38–42.)

Kuviossa 10. tuodaan rivillä 1 luokka GPIO käyttöön komennolla *import GPIO*. Funktio *getCBC()* kysyy akun varauksen tilaa ja akun latauksen tilaa ja sijoittaa sen muuttujaan paluuarvo. Paluuarvon arvo on tämän jälkeen muodossa latauksenTila, akunVaraus, jossa latauksenTila on arvo 0 (laturia ei kytketty), 1 (laturi kytketty ja lataa akkua) tai 2 (laturi kytketty ja akun varaus suoritettu) ja akunVaraus on akun varaus millivoltteina.

```
1 import GPIO
2 paluuarvo = GPIO.getCBC()
```

KUVIO 10. GPIO-luokka

MOD-luokka on kokoelma sekalaisia funktioita sisältäen esimerkiksi funktioita ajastukseen, automaattiseen moduulin uudelleenkäynnistykseen ja virransäästöön. (Telit 2007, 43–45.)

Kuviossa 11 tuodaan *import*-komennolla käyttöön luokka MOD, jonka jälkeen kutsutaan funktiota *watchdogEnable()*, joka asettaa moduulille automaattisen uudelleenkäynnistyksen tunnin (3600 sekuntia) välein.

```
1 import MOD
2 MOD.watchdogEnable(3600)
```

KUVIO 11. MOD-luokka

3.5 AT-komennot

AT-komennot olivat yksi keskeinen osa työtä, sillä niiden avulla siirretään tiedostoja laitteen ja PC-tietokoneen välillä sekä niillä myös ohjataan laitetta suoraan Python-skriptistä. Käytetyimmät komennot liittyivät juuri tiedostonhallintaan laitteessa.

Komennolla *AT#LSCRIPT?* luetaan moduulin tiedostolistaus. Komennon suorittaminen palauttaa onnistuneen suorituksen jälkeen vastauksen OK, sekä tiedostolistauksen. (Telit 2008, 415–416.)

Tiedoston siirtäminen moduuliin toimii komennolla *AT#WSCRIPT=”tiedosto.py”,210*. Komento suoritetaan Hyper Terminal -ohjelmalla. Pilkun jälkeinen numeroarvo tarkoittaa tiedoston kokoa (merkkien määrää tiedostossa, ei fyysistä kokoa tiedostojärjestelmässä). Kun komento on suoritettu, Hyper Terminal tulostaa ruudulle merkit ”>>>”, sitten valitaan valikosta ”Send Text file” ja valitaan tietokoneen tiedostohallinnasta tiedosto.py. Huomattavaa tiedostoa nimetessä on, että tiedoston nimen maksimipituus on 16 merkkiä, sekä isot ja pienet huomioidaan eri merkkeinä. (Telit 2008, 409-411.)

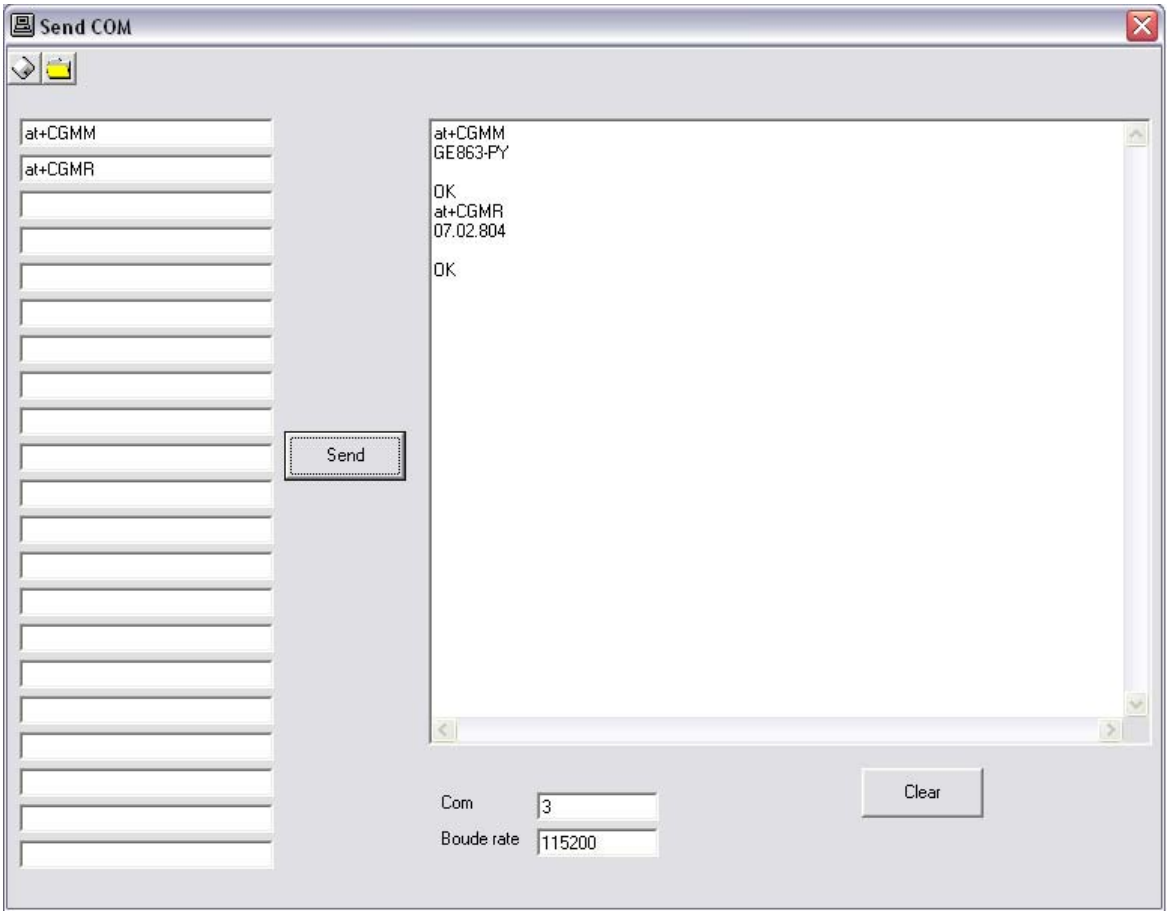
Tiedoston lukeminen onnistuu komennolla AT*#RSCRIPT=”tiedosto.py”*. Onnistunut suoritus palauttaa viestin OK sekä tiedoston sisällön. Jos tiedoston luku epäonnistuu, on tiedosto joko suojattu lukemiselta tai tiedoston siirrossa on tapahtunut virhe. Tällöin on syytä tarkistaa, onko *AT#WSCRIPT*-komentoa käytettäessä annettu oikea arvo tiedoston koolle. (Telit 2008, 414–415.)

Moduulissa ajettavaa Python-skriptiä voidaan vaihtaa komennolla *AT#ESCRIPT=”tiedosto.py”*. Onnistunut suoritus ei palauta mitään, vaan tieto onnistumisesta saadaan komennolla *AT#ESCRIPT?*, joka palauttaa aktiiviseksi asetetun skriptin nimen. (Telit 2008, 411–412.)

3.6 Ensimmäinen ohjelma

Koska ohjelmointikieli oli entuudestaan tuntematon minulle, päätin aloittaa aivan perusasioista, ja ohjelmointiperinteitä kunnioittaen ensimmäinen testiohjelma oli oleva ”Hei maailma” -tyyppinen tulostus Pythonia käyttäen. Aivan ensimmäiseksi kuitenkin oli testattava yhteys tietokoneen ja EZ-10 -moduulin välillä.

Kuviossa 12 näkyy vasemmalla syötetyt AT-komennot ja oikealla laitteen palauttamat vastineet komentoihin. Ohjelmalla voidaan suorittaa useampi komento kerrallaan ja suoritusjärjestys on ylhäältä alaspäin. Ennen komentojen lähettämistä on valittava oikea COM-portti (*Com*) sekä yhteysnopeus (*Boude rate*). Tietokoneessani porttinumero oli 3, ja Telitin manuaalin mukaisesti yhteysnopeus laitteen ja tietokoneen välillä on 115 200 bittiä sekunnissa. Painamalla Send-nappia ohjelma lähettää komennot moduulille ja tulostaa tekstialueelle itse komennot sekä saamansa vastaukset. Kuviossa on kysytty laitteelta mallinumeroa ja ohjelmistoversion numeroa komennoilla CGMM ja CGMR. (Telit 2008, 238–239.)



KUVIO 12. Send COM-ohjelmaan syötetyt AT-komennot

Kun yhteys on todettu toimivaksi tietokoneen ja laitteen välillä, voidaan aloittaa varsinainen ohjelmointi. Työkaluna toimii normaali ohjelmointikäyttöön tarkoitettu tekstieditori, jossa on syntaksivärjäys.

Kuviossa 13 on Python-skripti, joka tulostaa sarjaporttiin ensin tekstin ”*Hello world*” ja sen jälkeen 5 sekunnin välein ”*Hello from loop*”. Kun laite on kytketty kaapelilla tietokoneeseen, tulostuvat tekstit siis näytöllä. Riveillä 5 ja 6 tuodaan käyttöön MOD- ja SER-luokat, jotka ovat valmiiksi sisäänrakennettuna laitteeseen. Rivillä 10 asetetaan sarjaportin nopeus (baud rate), kuten manuaaleissa on käsketty eli, 115 200 bittiä sekunnissa. Tämän jälkeen tapahtuu ensimmäinen tulostus rivillä 13, *SER.send()* on funktio, joka lähettää string-tyyppistä tietoa sarjaporttiin. Rivillä 16 alkaa loputon silmukka, jossa tulostetaan sarjaporttiin teksti ”*Hello from loop*” viiden sekunnin välein. Ajastus toteutetaan *MOD.sleep()*-funktiolla, johon aika asetetaan sekunnin kymmenysosina.

```
1  #################
2  # Hello World!
3  # hello.py
4  #################
5  import MOD
6  import SER
7
8  def main():
9      # initialize
10     SER.set_speed('115200')
11
12     # hello!
13     SER.send('Hello World!\r\n')
14
15     # main loop
16     while 1:
17         SER.send('Hello from loop!\r\n')
18         MOD.sleep(50)
```

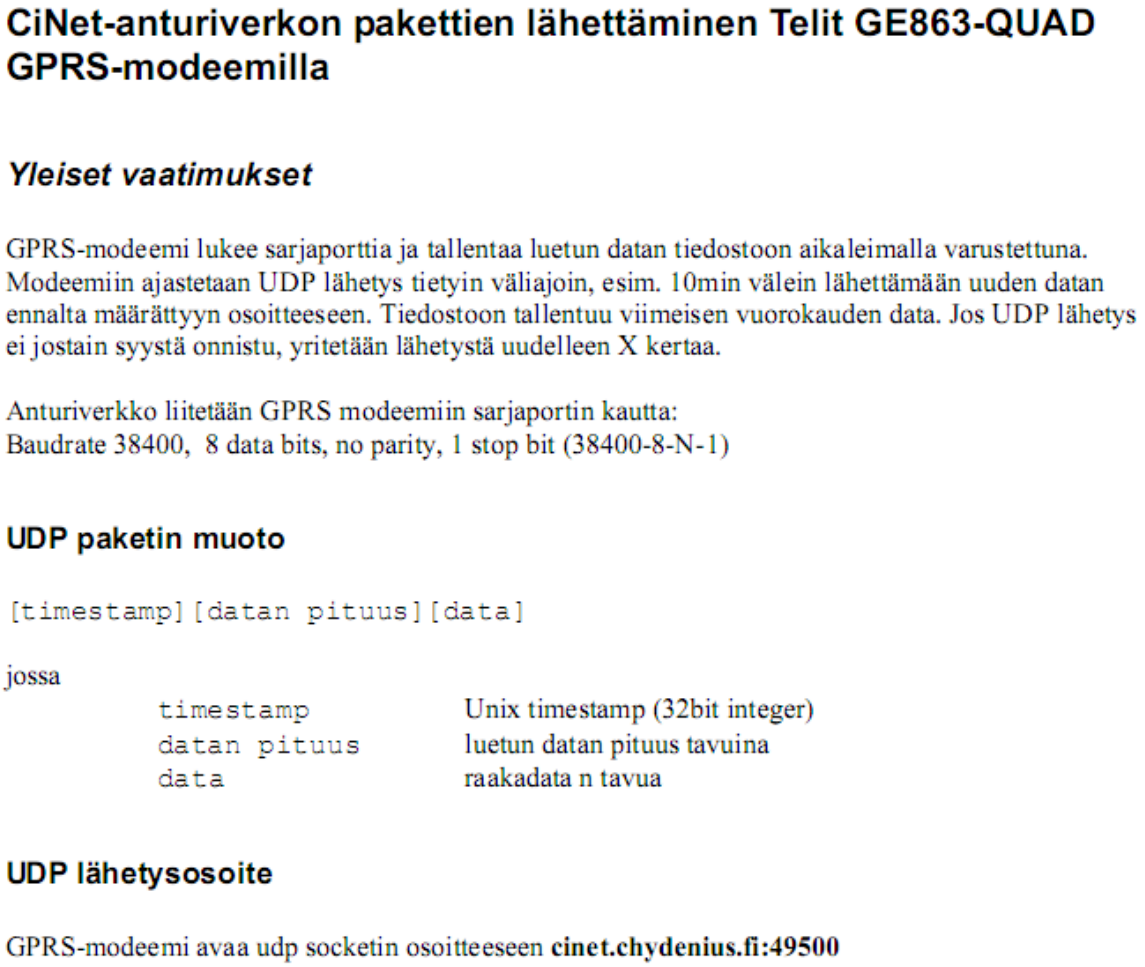
KUVIO 13. Python EasyScript Hello World

4 TOTEUTUS

4.1 Määrittely

Työn määrittely tuli Yliopistokeskus Chydeniukselta ja sisälsi kaksi dokumenttia: varsinaisen sanallisen määrittelyn sekä pdf-liitteen, jossa oli esitetty SLIP-koodaus datapakettien lähetyksessä.

Kuviossa 14 esitetään saatu vaatimusmäärittely. Otsikko kertoo Telitin GE863-QUAD modeemista, mutta työssä käytetty Telit EZ-10 on vastaava laite samoilla tarvittavilla ominaisuuksilla. Dokumentissa on kerrottu oleellisimmat asiat ja muu mahdollinen lisäinfo on saatu sähköpostikeskusteluissa työelämäohjaajan ja Chydeniuksen kontaktihenkilön välillä.



CiNet-anturiverkon pakettien lähettäminen Telit GE863-QUAD GPRS-modeemilla

Yleiset vaatimukset

GPRS-modeemi lukee sarjaporttia ja tallentaa luetun datan tiedostoon aikaleimalla varustettuna. Modeemiin ajastetaan UDP lähetys tietyin väliajoin, esim. 10min välein lähettämään uuden datan ennalta määrättyyn osoitteeseen. Tiedostoon tallentuu viimeisen vuorokauden data. Jos UDP lähetys ei jostain syystä onnistu, yritetään lähetystä uudelleen X kertaa.

Anturiverkko liitetään GPRS modeemiin sarjaportin kautta:
Baudrate 38400, 8 data bits, no parity, 1 stop bit (38400-8-N-1)

UDP paketin muoto

```
[timestamp][datan pituus][data]
```

jossa

`timestamp`	Unix timestamp (32bit integer)
`datan pituus`	luetun datan pituus tavuina
`data`	raakadata n tavua

UDP lähetysosoite

GPRS-modeemi avaa udp socketin osoitteeseen **cinet.chydenius.fi:49500**

KUVIO 14. Vaatimusmäärittely

4.2 Suunnittelu

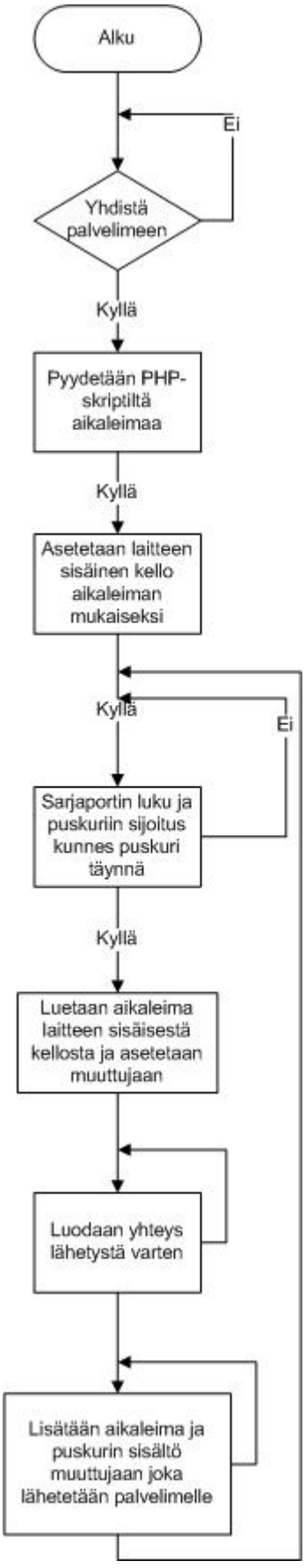
Ohjelmoinnissa kyseiselle laitteelle on tiettyjä asioita, jotka on otettava huomioon jo suunnitteluvaiheessa. Näitä ovat muun muassa laitteen tiedostojärjestelmä, joka sallii tiedostojen kirjoittamisen ja luvun vain yhdellä tasolla, eli ei ole mahdollista luoda alikansioita. Toinen rajoitus on se, että vain yhtä skriptiä voidaan ajaa kerralla. Muita rajoittavia tekijöitä ovat mm. laitteen rajoitettu tallennuskapasiteetti ja laitteen muistin kesto, kun kirjoitus ja luku-operaatioita suoritetaan jatkuvasti. Suoritettavaa skriptiä ajetaan alhaisella prioriteetillä sen vuoksi, että laitteen suorituskyky riittäisi jatkuvasti GPRS-yhteyden ylläpitämiseen mahdollistaen luotettavan tiedonsiirron.

Suunnittelussa jouduttiin aluksi etenemään pieni pala kerrallaan ottamalla selvää, mikä on mahdollista ja miten mikäkin toiminto rakennetaan, ja rakentamalla testiohjelmia haluttujen toimintojen aikaansaamiseksi. Kokonaisuuden hahmottamista helpotti aikajakson suunnitteleminen ohjelman toiminnan kuvaamiseksi ajanjaksoittain. Aikajaksoa päivitettiin aina jonkin osakokonaisuuden tutkimisen jälkeen.

Suunnittelua vaikeutti elävä vaatimusmäärittely; joitakin asioita muutettiin kesken suunnittelun ja toteutuksen. Suunnittelua ja osakokonaisuuksien toteutusta iteroitiin siten, että tarpeen mukaan palattiin suunnittelemaan osakokonaisuuden toiminta uudelleen.

Osakokonaisuuksien suunnittelu toimi siten, että rajattiin jokin yksittäinen tai pieni joukko yksittäisiä toimintoja vaatimusmäärittelystä tai vaatimusmäärittelyn ohella saaduista ohjeistuksesta ja tutkittiin, miten kyseinen toiminto voidaan toteuttaa vaatimusmäärittelyn mukaisella laitteistolla.

Kuviossa 15 esitetään ohjelman toiminta ajanjaksoittain. Aivan ensimmäiseksi ohjelma lukee asetustiedostoista tarvittavat tiedot yhteyden muodostamiseksi kohdepalvelimelle. Kun yhteys on saatu, lähetetään *GET-parametrina* pyyntö PHP-skriptille, joka palauttaa palvelimen kellonajan aikaleimana valmiina muotoiltuna moduulille (*vv/kk/pp,hh:mm:ss*), joka asetetaan laitteen sisäisen kellon ajaksi. Kellon asetuksen jälkeen pääohjelma siirtyy vaiheeseen, jossa luetaan sarjaporttia niin kauan, kunnes moduuli on vastaanottanut dataa ennaltamääritetyn arvon verran. Aina kun sarjaporttiin on saapunut dataa, se siirretään niin kutsuttuun puskuriin, jolla tarkoitetaan normaalia array[]-tyyppistä muuttujaa jonka pituudeksi on siis ennalta määritetty jokin arvo, esimerkiksi 20 ja tällöin silmukassa luetaan dataa 20 kertaa. Kun puskuri on täynnä, luodaan string-tyyppinen muuttuja jonka alkuun lisätään laitteen sisäisen kellon aikaleima ja puskurin sisältö sen perään. Ohjelma lukee jälleen asetustiedostot yhteydenmuodostusta varten. Tämä on tarpeen siksi, että kohdepalvelin voi olla eri kuin sisäisen kellon synkronoinnissa. Kun palvelimelta saapuu vastaus, että data on onnistuneesti vastaanotettu, palaa ohjelma kohtaan, jossa sarjaporttia taas luetaan, kunnes puskuri on täynnä dataa seuraavaa lähetystä varten.



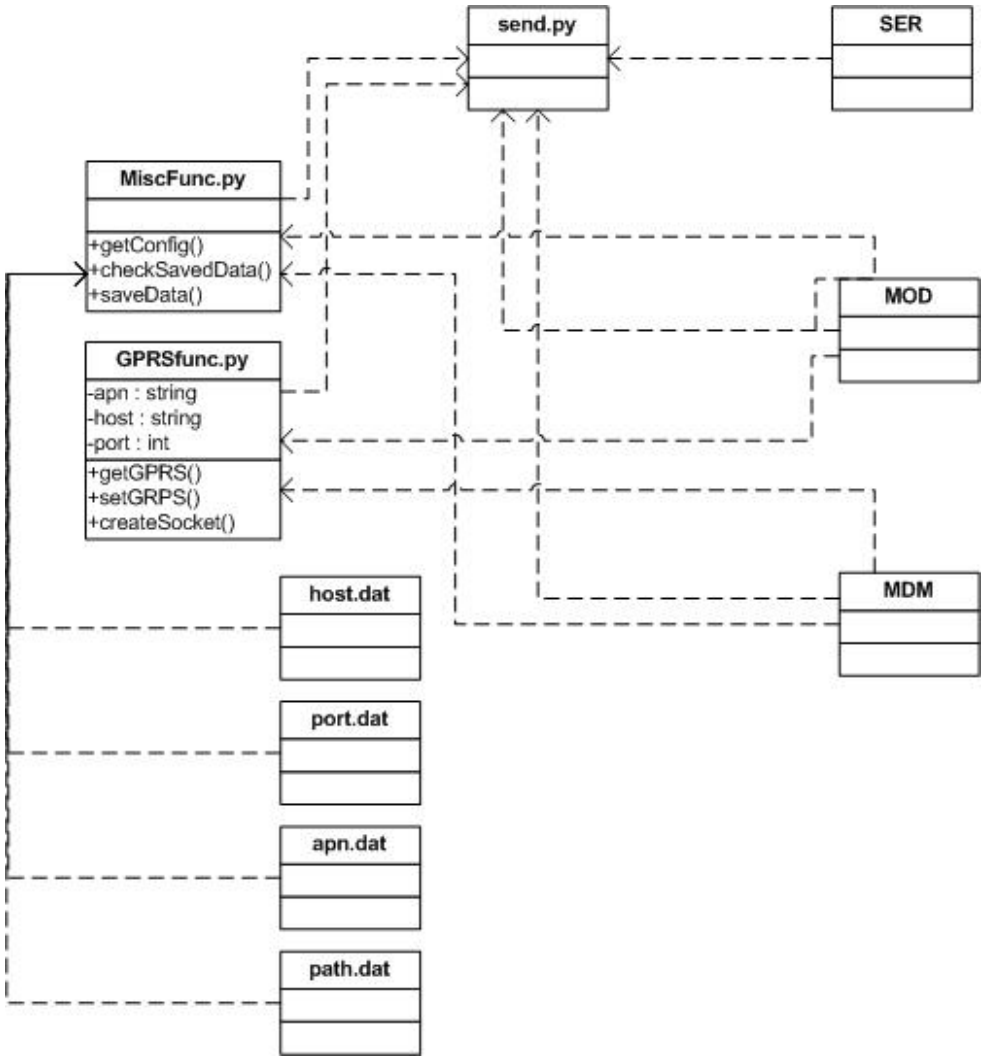
KUVIO 15. Ohjelman toiminta ajanjaksoittain

4.3 Ohjelmointi

Ohjelmointi jaettiin pieniin palasiin ja usein ennen jonkin toiminnallisuuden implementointia sitä testattiin erillisessä ohjelmassa. Esimerkkinä on kellon synkronointi: ensin loin tiedoston *kello.py*, jonka asetin tulostamaan laitteen sisäisen kellon aikaa sekunnin väliajoin. Olin asettanut sisäisen kellon arvon senhetkisen kellonajan mukaiseksi aluksi AT-komennolla. Laite oli käynnistettävä uudelleen, jotta se siirtyisi AT-tilasta skriptin suoritukseen. Tällöin kävi ilmi skriptin suorituksessa, ettei laite kykene muistamaan sisäisen kellonsa aikaa uudelleenkäynnistyksen jälkeen, vaan se olisi suoritettava synkronointi palvelimelta. Rakensin tämän toiminnallisuuden samaan kello.py-skriptiin, ennen kuin siirsin sen vasta valmiina lopulliseen paikkaansa pääohjelmassa *send.py*.

Suunnittelun perusteella minulle oli hyvin selvää, miten jaottelisin ohjelman toimintaa eri luokkiin ja tiedostoihin. Asetukset tehtiin helposti vaihdettavaksi siten, että ne sijoitetaan erillisiin tekstitiedostoihin, ja niiden tiedostopäätteiksi valitsin .dat. Luokat MiscFunc.py ja *GPRSfunc.py* muotoutuivat ohjelmoinnin edetessä, mutta olin jo aluksi päättänyt, että ne jaotellaan erillisiin tiedostoihin toiminnallisuuden mukaan, eli *GPRSfunc.py* sisältääkin kaikki yhteyden muodostukseen liittyvät funktiot.

Kuviossa 16 esitetään tiedostojen välinen yhteys toisiinsa. *Send.py* on pääohjelma ja se käyttää kaikkia muita tiedostoja suoraan tai epäsuorasti. Asetustiedostoja host.dat, port.dat, *apn.dat* ja *path.dat* käytetään *MiscFunc.py* -tiedostossa, jota käytetään esimerkiksi yhteysasetusten asettamiseen pääohjelmasta. *GPRSfunc.py* on yhteyksien luomiseen tarkoitettu luokka, jota kutsutaan pääohjelmasta. Tiedostot *GPRSfunc.py* ja *MiscFunc.py* käyttävät EasyScript -laajennuksia MDM ja MOD. Pääohjelma käyttää lisäksi laajennusta SER, jolla hoidetaan sarjaporttiin tulostus.



KUVIO 16. Tiedostojen välinen yhteys

Pääohjelman voisi jakaa kolmeen osaan: alkuosassa ladataan asetukset asetustiedostoista, toisessa osiossa muodostetaan yhteys palvelimelle, josta haetaan palvelimen kellonaika. Kellonaika asetetaan GPRS-modeemin sisäisen kellon ajaksi. Loppuosassa dataa otetaan vastaan ja lähetetään palvelimelle paketteina aikaleiman kera.

Kuviossa 17 on pääohjelman alku, jossa riveillä 1–5 tuodaan käyttöön luokat *miscFunc* ja *GPRSfunc*, sekä Telitin laajennusmoduulit *MDM*, *MOD* ja *SER*. Rivit 21–30 sisältävät muuttujien esittelyn ja lyhyen kuvauksen. Muuttuja *data* on varattu sisääntulevan datan sijoittamiselle. Muuttuja *apn* tarkoittaa yhteystyyppiä, *host* kertoo palvelimen osoitteen ja *port* palvelimen portin, johon yhteys muodostetaan. *Gprs*-muuttuja on tarkistemuuttuja johon sijoitetaan tieto GPRS-yhteyden tilasta. *Ip*-muuttujaan sijoitetaan moduulin oma IP-

osoite. Muuttuja *buffer* on array-tyyppinen muuttuja, johon sijoitetaan vastaanotettu data muuttujasta *data*. Sille asetetaan maksimiarvo muuttujalla *BUFFER_MAX_SIZE*, joka siis määrittää, montako datapakettia palvelimelle edelleenlähetetään kerralla. Rivillä 36 ladataan asetukset muuttujille *apn*, *host*, *port* ja *path*. Asetukset luetaan tiedostoista *apn.dat*, *host.dat*, *port.dat* ja *path.dat*, jotka on sijoitettu laitteen pysyväismuistiin.

```
1  from miscFunc import *
2  from GPRSfunc import *
3  import MDM
4  import MOD
5  import SER
6  def send():
7
8      ##############################################################################
9      #     Ari Joki                                                               #
10     #                                                                            #
11     #     Program to gather and resend data with GE863-PY/Telit EZ-10            #
12     #     Listens serialport and gathers all incoming data and resends it over   #
13     #     to server using GPRS (TCP)                                             #
14     #     set HOST and PORT in host.dat / port.dat                               #
15     ##############################################################################
16
17     ########################
18     # CONFIG               #
19     ########################
20
21     data = ''
22     apn = ''          # 'internet'
23     host = ''         # remote host where the data is sent to, ip address or domain name
24     port = ''         # remote port on the remote server
25     path = ''         # relational path to the receiving script on the remote server
26     gprs = 0          # GPRS status [0 = no connection, 1 = connected, -1 = error]
27     ip = ''           # if the connection is ok, this variable will receive the ip address of the module
28     buffer = []       # array to store received gps locations
29     BUFFER_MAX_SIZE = 10  # maximum number of data to store before sending them
30     time = ''                 # timestamp from server
31
32     #asetetaan nopeus 38400 8Databittia, ei pariteettia, 1 stop-bit
33     SER.set_speed('38400', '8N1')
34
35     # get settings from files: apn.dat, host.dat, port.dat, path.dat
36     apn, host, port, path = getConfig() # in miscFunc
37
```

KUVIO 17. Pääohjelman lähdekoodin alkuosa

Kuviossa 18 muodostetaan yhteys palvelimeen kellonajan hakemiseksi. Ohjelman osa on sijoitettu while-silmukkaan ja yhteyden muodostusta yritetään niin kauan, kunnes vastaus palvelimelta saadaan oikein. Kun GPRS-yhteys on onnistuneesti luotu, rivillä 59 yhdistetään palvelimelle käyttäen funktiota *createSocket()* ja sen jälkeen rivillä 69 kutsutaan pal-

velimella olevaa PHP-tiedostoa lähettämällä sille *GET*-pyyntö, joka on luotu rivillä 67. Parametrina on ”*time*” ja sen arvona 1. PHP-skriptin palauttama vastaus sijoitetaan muuttujaan *time* rivillä 73 ja modeemin sisäinen kello asetetaan riveillä 74 ja 75 lähettämällä modeemille AT-komento *AT+CCLK([time-muuttujan arvo])*.

```
40  # ====================================
41  while(time == ''):
42      # ====================
43      #  Check GPRS
44      # ====================
45      if(gprs != 1):
46          # no gprs connection, set gprs connection
47          SER.send('CONNECTING TO GPRS...')
48          gprs, ip = setGPRS(apn) # GPRSfunc.py
49          if(gprs == 1):
50              SER.send(' CONNECTED [IP: '+ip+'\r\n')
51          if(gprs == -1):
52              SER.send(' ERROR\r\n')
53          if(gprs == 0):
54              SER.send(' NO CONNECTION\r\n')
55
56      if(gprs == 1):
57          SER.send('SENDING DATA OVER GPRS...')
58
59          sock = createSocket(port, host) # create socket | miscFunc.py
60
61          if(sock == 1):
62              SER.send(' Socket OK\r\n')
63
64              SER.send("["+senddata+"]")
65
66              # commit http request
67              str_send = 'GET ' + path + '1' + ' HTTP/1.1\r\nHost: ' + host + '\r\nConnection: close\r\n\r\n'
68
69              MDM.send(str_send, 5)
70
71              res = MDM.receive(10)
72              SER.send("\r\nRESPONSE RECEIVED=["+res+"]\r\n")
73              time = res
74              MDM.send('AT+CCLK='+time+',0')
75              MDM.sendbyte(0x0d, 0)
76              # no carrier ends the connection unless an error occurs
77              SER.send("WAITING RESPONSE...")
78              while(res.find('NO CARRIER') == -1):
79                  r = MDM.receive(2)
80                  SER.send(".")
81                  res = res + r
82
83              SER.send("LOOP FINISHED\r\n")
84              #sendSuccessful = 1
85              SER.send("RESPONSE=[ " + res + "]\r\n")
86          else:
87              SER.send(" No socket!!!\r\n")
```

KUVIO 18. Pääohjelman lähdekoodin osa

Kuviossa 19 esitetään *time.php* -tiedoston sisältö. PHP-skriptin tarkoituksena on palauttaa palvelimen kellon mukainen aikaleima, joka formatoidaan Telit-moduulin tarvitsemaan muotoon. Lisätoimintona skripti tallentaa myös aikaleiman palvelimella sijaitsevaan

count.dat-tiedostoon. Skripti aktivoidaan parametrilla *time*, jonka arvon tulee olla 1, jotta skripti palauttaa aikaleiman. Aikaleima muodostetaan rivillä 15 ja lopuksi rivillä 27 se tulostetaan muotoiltuna takaisin skriptiä kutsuneelle EZ-10 -GPRS-moduulille.

```
1  <?PHP
2  /*
3          Palauttaa responsena timestampin muodossa:
4          "yy/MM/dd,hh:mm:ss"
5  */
6
7  $tiedosto = 'count.dat';
8  if(isset($_GET['time'])) {
9  $kutsu = $_GET['time'];
10 }else { exit('0'); }
11
12
13 if($kutsu == 1)
14 {
15 $str_aika = date('Y/m/d,H:i:s');
16
17 $fp = fopen($tiedosto, "a");
18 fwrite ($fp, $str_aika);
19 fclose ($fp);
20
21
22 }else { exit('0'); }
23
24
25 // poistetaan kaksi ensimmäistä merkkiä
26 $aika = substr($str_aika, 2,17);
27 echo $aika;
28
29 ?>
```

KUVIO 19. Kellonajan palauttava PHP-tiedosto

Kuviossa 20 aloitetaan ohjelman pääsilmukka. Rivillä 98 pääsilmukan sisällä aloitetaan sisäinen while-tyypin silmukka, jota suoritetaan niin kauan, kunnes dataa on vastaanotettu sarjaportista. Rivillä 101 käytetään SER-laajennosluokan *receive()*-funktiota, jolla luetaan sarjaporttia 2 sekuntia kerrallaan. Kun data on vastaanotettu, se lisätään *buffer*-muuttujaan Pythonin *append()*-funktiolla. Rivillä 112 tarkistetaan, onko *buffer*-muuttuja saavuttanut maksimiarvonsa ja jos on, siirrytään yhteyden luontiin ja datan muotoilemiseen, jos ei, toistetaan sarjaportin luku ja *buffer*-muuttujaan datan lisäys.

```
88  # Main loop
89  #============
90  while 1:
91
92      res = ''
93      #=========================
94      # Request data from serial
95      #=========================
96      SER.send('\r\nWaiting for more data...')
97
98      while(data == ''):
99
100         #luetaan sarjaporttia 2s
101         data = SER.receive(20)
102
103     SER.send(' OK \r\n[data='+data+']\r\n')
104
105     buffer.append(data)
106     data = ''
107
108     #=============================================================
109     # Check buffer capacity. If buffer is full, send data
110     # over GPRS. Otherwise, request more data
111     #=============================================================
112     if(len(buffer) != BUFFER_MAX_SIZE):
113         SER.send('BUFFER NOT FULL, REQUEST MORE DATA...')
114         MOD.sleep(10)
115     else:
116         SER.send('BUFFER FULL, PREPARE TO SEND DATA...')
117         #=======================
118         #  Check GPRS
119         #=======================
120         if(gprs != 1):
121             #no gprs connection, set gprs connection
122             SER.send('CONNECTING TO GPRS...')
123             gprs, ip = setGPRS(apn) # GPRSfunc.py
124             if(gprs == 1):
125                 SER.send(' CONNECTED [IP: '+ip+'\r\n')
126             if(gprs == -1):
127                 SER.send(' ERROR\r\n')
128             if(gprs == 0):
129                 SER.send(' NO CONNECTION\r\n')
```

KUVIO 20. Pääohjelman lähdekoodin osa

Kuviossa 21 muodostetaan yhteys palvelimeen ja lisätään lähetettävään dataan aikaleima. Yhteyden luonti tapahtuu rivillä 139 käyttämällä *createSocket()*-funktiota, jonka parametreina ovat kohdepalvelimen portti ja IP-osoite. Yhteyden muodostuksen onnistuttua *buffer*-muuttujan sisältö käydään läpi riveillä 149–156 ja sisältö lisätään muuttujaan *senddata*. Riveillä 148, 151 ja 157 kommentoituna olevat koodit on tarkoitettu testausvaiheeseen,

jolloin data kirjoitettiin laitteen muistissa sijaitsevaan tekstitiedostoon. Viimeisillä riveillä olevat kommentoidut koodit taas oli tarkoitettu HTML-enkoodaukseen, joissa esimerkiksi välilyönti muutetaan muotoon ”*%20*”.

```
131
132                 #=============================================
133                 #  Send buffered DATA over GPRS
134                 #=============================================
135                 sendSuccessful = 0
136                 if(gprs == 1):
137                       SER.send('SENDING DATA OVER GPRS...')
138
139                       sock = createSocket(port, host) # create socket | miscFunc.py
140
141                       if(sock == 1):
142                             SER.send(' Socket OK\r\n')
143
144                             # concatenate data stored in buffer
145                             senddata = ""
146                             bufsize = len(buffer)
147                             counter = 0
148                             #f = open('data.txt', 'ab')
149                             for item in buffer:
150                                   counter = counter + 1
151                                   #f.write(item)
152
153                                   if counter != bufsize:
154                                         senddata = senddata + item
155                                   else:
156                                         senddata = senddata + item
157                             #f.close()
158                             # encode special characters
159                             # senddata = senddata.replace(';','%3B')
160                             #senddata = senddata.replace(' ','%20')
161                             #senddata = '*R' + senddata + '%3E'
162
```

KUVIO 21.Pääohjelman lähdekoodin osa

Kuviossa 22 toteutetaan datan lähetys kohdepalvelimelle. Rivin 164 sarjaporttiin tulostus on testausta varten, ja siinä tulostetaan muodostettu datapaketti kokonaisuudessaan. Rivillä 170 lähetetään data palvelimelle. Kuviossa on hieman eri versio koodista kuin lopullinen Yliopistokeskukselle päätynyt versio: *str_send*-muuttujan pitäisi sisältää muuttuja

send_data eikä rivillä 167 esiintyvä muodostettu parametri, joka on tarkoitettu GET-parametrin lähetykseen pyydettäessä aikaleimaa PHP-skriptiltä.

```
164                     SER.send("["+senddata+"]")
165
166                     # commit http request
167                     str_send = 'GET ' + path + time + ' HTTP/1.1\r\nHost: ' + host + '\r\nConnection: close\r\n\r\n'
168                     #str_send = senddata
169
170                     MDM.send(str_send, 5)
171
172                     res = MDM.receive(10)
173                     SER.send("\r\nRESPONSE RECEIVED=["+res+"]\r\n")
174
175                     # no carrier ends the connection unless an error occurs
176                     SER.send("WAITING RESPONSE...")
177                     while(res.find('NO CARRIER') == -1):
178                         r = MDM.receive(2)
179                         SER.send(".")
180                         res = res + r
181
182
183                     SER.send("LOOP FINISHED\r\n")
184                     sendSuccessful = 1
185                     SER.send("RESPONSE=[ " + res + "]\r\n")
186                 else:
187                     SER.send(" No socket!!!\r\n")
188
189             if(sendSuccessful == 1):
190                 SER.send("SEND SUCCESSFUL, BUFFER CLEARED\r\n")
191             else:
192                 #huom ei tiedostotallennusta talla hetkella
193                 SER.send("SEND FAILED, BUFFER WRITTEN IN FILE\r\n")
194             buffer = []   # clear buffer
195             senddata = ''
196
```

KUVIO 22. Pääohjelman lähdekoodin loppuosa

4.4 Testaus

Testaus ja ongelmien korjaus osoittautuivat työssä melkoiseksi haasteeksi; ongelmia tuottivat usein ihan pienetkin asiat, kuten aiheutuneen ohjelmointivirheen etsiminen. Ongelmat johtuivat lähinnä siitä, että mitään virheilmoitusta ei generoitunut, mutta ohjelma ei toiminut. Tämä yhdistettynä siihen, että laitteen käynnistymisen jälkeen tarvittava aika skriptin suorittamiseen saattoi olla useita minuutteja, teki työstä hidasta ja kärsivällisyyttä vaativaa. Joissakin ongelmatilanteissa virheilmoituksia sai tallennetuksi GPRS-moduulin sisäiseen muistiin tekstitiedostoon käyttämällä poikkeustenkäsittelyä.

Kuviossa 23 on *appl.py*-skripti, jonka tarkoituksena on suorittaa virheviestin lokitus tekstitiedostoon *loadError.txt*. Pääohjelma ajetaan *try*-osiossa, ja jos pääohjelma *send.py* tekee poikkeuksen, poikkeuksen syy kirjataan tekstitiedostoon. Rivillä 9 avataan tekstitiedosto käsittelyä varten, ja parametreina funktiolle *open()* menee tekstitiedoston nimi sekä tieto siitä, että tiedosto avataan kirjoitusta varten binäärimuodossa (*'wb'*). Rivillä 10 tulostetaan tiedostoon teksti *"ERROR LOADING send()*" sekä itse virheilmoitus. Lopuksi tiedosto suljetaan rivillä 11 käyttämällä funktiota *close()*.

```
1  from send import *
2
3  # try to run the main appliction
4  try:
5      send()
6
7  except:
8      # if an error occurs, write the error message to a file
9      error_log = open('loadError.txt', 'wb')
10     error_log.write('ERROR LOADING send(): + sys.exc_info()[0] + \r\n\r\n')
11     error_log.close()
```

KUVIO 23. Virheen kirjoittaminen lokiin

Alkuvaiheessa kun onnistuin lähettämään ensimmäiset datapaketit testipalvelimelle, oli tarkistettava, oliko data oikeaa muotoa. Tämä tarkistettiin yksinkertaisesti lataamalla PHP-skriptin tallentama tekstitiedosto, johon data oli lähetetty, ja sitten avaamalla se binäärimuodossa sopivalla tekstieditorilla. SLIP-koodatuista paketeista selvisi siten esimerkiksi sensorin numero ja varsinainen informaatio, esimerkiksi lämpötila.

Myöhemmin ohjelman ollessa kutakuinkin valmis testaus tapahtui suoraan lopullisen kohdepalvelimen kanssa. Testausta suoritettiin muun muassa siten, että laitteisto jätettiin viikonlopun yli lähettämään dataa kohdepalvelimen porttiin, jossa Java-ohjelma otti sen vastaan ja lokitti tiedot saapuneista paketeista tekstitiedostoon. Tällä tavoin saatiin selville laitteiston luotettavuus, sillä jokainen SLIP-koodattu datapaketti sisältää paketin järjestysnumeron. Voitiin siis todeta, montako yksittäistä sensorilta tullutta datapakettia jäi saapumatta.

5 YHTEENVETO

Opinnäytetyön tarkoituksena oli toteuttaa kerätyn datan edelleenlähetys langattomasti tiedonkeruulaitteelta palvelimelle. Alussa sain työstä vaatimusmäärittelyn, jota kuitenkin työn edetessä muuteltiin ja tarkennettiin sanallisesti työn edistyessä. Työn tekovaiheessa työn vastaanottaja oli aktiivisesti käytettävissä lisätietoja tarvittaessa, mutta varsinaista dokumentaatiota en yliopistokeskuksen laitteistoista saanut enkä ole perillä laitteiston spekseistä kovin tarkasti. Tästä ei kuitenkaan aiheutunut mitään ongelmaa, sillä työn pääosassa on selkeästi Telitin GPRS-modeemi.

Työn haastavuus ei ollut niinkään tehdyn ohjelman laajuudessa, vaan siinä, että itse laitealusta ja ohjelmointikieli, jolla toteutus tehtiin, olivat minulle täysin uusia. Työssä tulikin itseopiskeltua aika paljon Python-ohjelmointieltä sekä kokeilun ja virheen kautta opittua miten käytetään Telitin GPRS-modeemeja.

Lähes kaikki informaatio etenkin Telitin tuotteista löytyi vain Internetistä, mutta tämä oli tiedossa etukäteen, koska laitteet ovat kovin yleisiä. Se ei kuitenkaan tuottanut ongelmia, ehkä jopa päinvastoin, sillä iso määrä tietoa löytyi keskitetysti saman Internet-sivun alta ja dokumentaatio oli erittäin kattava. Käytännön esimerkkejä olisin ehkä kaivannut lisää Python EasyScript -ohjelmoinnista, mutta muutoin Telitin omilla ohjeilla selviää varsin pitkälle.

Työn toteutus onnistui hyvin jakamalla kokonaisuus pieniin ongelmiin. Esimerkiksi GPRS-modeemin kellon synkronointiin palvelimen kellonajan kanssa tein erillisen ohjelman, jolla kokeilin, miten se toteutetaan. Harjoittelin siis Python Easy Scriptiä tekemällä yksinkertaisia pieniä työstä erillisiä ohjelmia, joilla kuitenkin oli jokin merkitys työn kannalta, kuten edellä mainittu kellon synkronointi. Työ onnistui mielestäni määrittelyjen mukaisesti; se tekee juuri sen, mitä sen oli tarkoitettukin tekevän. Jatkokehittelyn kannalta olennaista olisi tietää, mitä työn tilaaja haluaa, joten minulla ei varsinaisia lisäominaisuusehdotuksia ole, vaan keskittyisin enemmän nykyisen toiminnallisuuden virheensietokykyyn. Ohjelma on varsin yksinkertainen rakenteeltaan, ja toimintavarmuus on hyvä jo sellaisenaan, mutta erilaiset ongelmatilanteista palautumiset olisivat mielestäni tärkeä ottaa huomioon, jos ohjelmaa kehitetään edelleen.

Laitteisto mahdollistaa monenlaisten ominaisuuksien lisäämisen, esimerkiksi etäohjauksen SMS-viestillä, jolla voitaisiin vaikkapa konfiguroida moduulin asetuksia siten, että moduuli lähettää dataa asetusviestissä asetettuun IP-osoitteeseen. Erilaisia ongelmatilanteita miettiessä tulevat mieleen sähkökatkokset, jolloin tekemäni ohjelma tietenkin lopettaisi toimintansa, koska virransyöttö loppuisi. Moduulista on kuitenkin saatavilla akullinen versio, jolloin voitaisiin kehittää ohjelmaan lisäominaisuus, joka tunnistaa, kun ollaan vain akkuvirran varassa. Tällöin ohjelma lähettäisi palvelimelle viestin sähkönsyötön häiriöstä ja oman sarjanumeronsa, jonka perusteella laitteet identifioidaan. Rajana alustan ohjelmakehitykselle on lähestulkoon ainoastaan mielikuvitus. Rajoittavia tekijöitä on toki myös, esimerkiksi riippuvuus GSM-verkosta, riippuvuus näkyvyydestä satelliittien ja GPS-antennin välillä (jos käytetään paikannusta), rajallinen laitteiston teho, sähkön tarve (akullisen versionkin käyttöaika on rajallinen) ja kevennetty ohjelmointikieli Python Easy Script.

LÄHTEET

Telit. 2007. Telit Easy Script Python. WWW-dokumentti.
Saatavissa: http://www.telit.com/module/infopool/download.php?id=617. Luettu 12.10.2009.

Telit. 2008. Telit AT Reference Guide. WWW-dokumentti.
Saatavissa: http://www.telit.com/module/infopool/download.php?id=542. Luettu 12.10.2009.

Telit. 2009. Telit GE863 Family Hardware User Guide. WWW-dokumentti.
Saatavissa: http://www.telit.com/module/infopool/download.php?id=545. Luettu 12.10.2009.

Teller, Matt. 2007. Python Power! : The Comprehensive Guide

Sensirion. SHT1x / SHT7x Humidity & Temperature Sensor. WWW-dokumentti. Saatavissa: http://www.sensirion.com/images/getFile?id=25. Luettu 5.11. 2009.

```
from miscFunc import *
from GPRSfunc import *
import MDM
import MOD
import SER
def send():

    ##############################################################################################
    #     Ari Joki                                                                               #
    #                                                                                            #
    #     Program to gather and resend data with GE863-PY/Telit EZ-10                            #
    #     Listens serialport and gathers all incoming data and resends it over                   #
    #     to server using GPRS (TCP)                                                             #
    #     set HOST and PORT in host.dat / port.dat                                               #
    ##############################################################################################

    ########################
    # CONFIG               #
    ########################
    data = ''
    apn = ''          # 'internet'
    host = ''         # remote host where the data is sent to, ip address or domain name
    port = ''         # remote port on the remote server
    path = ''         # relational path to the receiving script on the remote server
    gprs = 0          # GPRS status [0 = no connection, 1 = connected, -1 = error]
    ip = ''           # if the connection is ok, this variable will receive the ip address of the module
    buffer = []       # array to store received gps locations
    BUFFER_MAX_SIZE = 10      # maximum number of data to store before sending them
    time = ''         # timestamp from server

    #asetetaan nopeus 38400 8Databittia, ei pariteettia, 1 stop-bit
    SER.set_speed('38400', '8N1')

    # get settings from files: apn.dat, host.dat, port.dat, path.dat
    apn, host, port, path = getConfig() # in miscFunc
```

```
# ================================
# ===========
# Main loop
# ===========
while 1:

        res = ''
        # ====================
        # Request data from serial
        # ====================
        SER.send('\r\nWaiting for more data...')

        while(data == ''):

                #luetaan sarjaporttia 2s
                data = SER.receive(20)

        SER.send(' OK \r\n[data='+data+']\r\n')

        buffer.append(data)
        data = ''

        # ================================================
        # Check buffer capacity. If buffer is full, send data
        # over GPRS. Otherwise, request more data
        # ================================================
        if(len(buffer) != BUFFER_MAX_SIZE):
                SER.send('BUFFER NOT FULL, REQUEST MORE DATA...')
                MOD.sleep(10)
        else:
                SER.send('BUFFER FULL, PREPARE TO SEND DATA...')
                # ====================
                #  Check GPRS
                # ====================
                if(gprs != 1):
                        # no gprs connection, set gprs connection
                        SER.send('CONNECTING TO GPRS...')
                        gprs, ip = setGPRS(apn) # GPRSfunc.py
                        if(gprs == 1):
                                SER.send(' CONNECTED [IP: '+ip+'\r\n')
                        if(gprs == -1):
                                SER.send(' ERROR\r\n')
                        if(gprs == 0):
                                SER.send(' NO CONNECTION\r\n')
```

```
#  Send buffered DATA over GPRS
# =====================================
sendSuccessful = 0
if(gprs == 1):
	SER.send('SENDING DATA OVER GPRS...')
	sock = createSocket(port, host) # create socket | miscFunc.py
	if(sock == 1):
		SER.send(' Socket OK\r\n')

		# concatenate data stored in buffer
		senddata = ""
		bufsize = len(buffer)
		counter = 0

		for item in buffer:
			counter = counter + 1

			if counter != bufsize:
				senddata = senddata + item
			else:
				senddata = senddata + item

		SER.send("["+senddata+"]")
		# Get time from inner clock
		MDM.send('AT+CCLK?', 0)
		MDM.sendbyte(0x0d, 0)
		time = MDM.receive(10)
		senddata = time + senddata
		# commit http request
		str_send = senddata
		MDM.send(str_send, 5)
		res = MDM.receive(10)
		SER.send("\r\nRESPONSE RECEIVED=["+res+"]\r\n")

		# no carrier ends the connection unless an error occurs
		SER.send("WAITING RESPONSE...")
		while(res.find('NO CARRIER') == -1):
			r = MDM.receive(2)
			SER.send(".")
			res = res + r

		SER.send("LOOP FINISHED\r\n")
		sendSuccessful = 1
		SER.send("RESPONSE=[ " + res + "]\r\n")
	else:
		SER.send(" No socket!!!\r\n")

if(sendSuccessful == 1):
	SER.send("SEND SUCCESSFUL, BUFFER CLEARED\r\n")
else:
	SER.send("SEND FAILED, BUFFER WRITTEN IN FILE\r\n")
buffer = []  # clear buffer
senddata = ''
```