LAUREA AMMATTIKORKEAKOULU Uuden edellä

Liikennemäärien visualisointi Javalla DSiP-lokitiedostoista

Alaverronen, Sami

2012 Leppävaara

Laurea-ammattikorkeakoulu
Laurea Leppävaara

Liikennemäärien visualisointi Javalla DSiP-lokitiedostoista

Alaverronen, Sami
Tietojenkäsittelyn koulutusohjelma
Liikennemäärien visualisointi Javalla DSiP-lokitiedostoista
Kesäkuu, 2012

Laurea-ammattikorkeakoulu **Tiivistelmä**
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma

Alaverronen, Sami

Liikennemäärien visualisointi Javalla DSiP-lokitiedostoista

Vuosi 2012 Sivumäärä 33

DSiP, Distriputed Systems intercommunication Protocol on Ajeco Oy:n luoma ohjelmisto, jonka tarkoituksena on taata luotettava tiedonsiirto ja turvallisuus verkossa. Tämä saavutetaan reitittämällä tietoliikennettä monikanavaisesti, usean eri siirtotien avulla. Ohjelmisto tuottaa lokitiedostoja, joiden avulla voidaan seurata DSiP-verkon eri solmukohtien käyttäytymistä. Näiden lokitiedostojen sisältö ei kuitenkaan ole helposti luettavissa silmämääräisesti, joten toinen tapa niiden tulkitsemiseksi tuli kehittää.

Opinnäytetyöni tavoitteena on luoda ohjelma, joka lukee DSiP-verkon luomia lokitiedostoja ja muuttaa niissä olevat liikennemäärät helposti ymmärrettävään visuaaliseen muotoon. Lisäksi halutaan ohjelman tukevan useita eri kieliä ja kielten lisääminen onnistuisi ilman koodinmuutoksia itse ohjelmaan. Tavoitteena oli myöskin luoda ohjelma, joka toimii monella eri käyttöjärjestelmäalustalla.

Ohjelma kirjoitettiin Java-ohjelmointikielellä Netbeans-ohjelmointiympäristössä. Ohjelman avulla käyttäjä pystyy valitsemaan haluamansa lokitiedoston, määrittelemään aloitusajan sekä halutun aikavälin. Näiden arvojen avulla ohjelma pystyy lukemaan ja piirtämään pylväsgrafiikkaa lokitiedoston sisällön mukaisesti. Pylväsgrafiikan lisäksi ohjelma kirjoittaa lokitiedoston tekstin erilliselle välilehdelle. Sekä pylväsgrafiikka että kirjoitettu teksti on värikoodattu kolmella värillä, punainen, keltainen ja vihreä, ja värien raja-arvoja pystyy muuttamaan konfiguraatiovalikosta. Myös ohjelman kielen vaihto on yksinkertaista ja mahdollista ilman koodin muutosta ohjelmassa.

Ohjelma on saavuttanut sille asetetut toiminnalliset tavoitteet ja sen lähdekoodi on luovutettu yritysasiakkaalle. On kuitenkin epäselvää tuleeko yritys ottamaan sitä käyttöön vai muokataanko siitä vielä sopivampi versio.

Asiasanat: DSiP, Monikanavareititys, Java, visualisoiminen

Laurea University of Applied Science **Abstract**
Laurea Leppävaara
Bachelor's Degree Programme in Business Information Technology

Alaverronen, Sami

Visualization of data traffic amounts from DSiP log files with Java

Year 2012 Pages 33

DSiP, Distributed Systems intercommunication Protocol, is a system created by Ajeco Oy that aims to guarantee a reliable data transfer and security on the network. This is achieved by routing communications via multiple channels, using many different communication mediums. The system produces log files that can be used to follow the behavior of nodes on the DSiP network. The data in these log files are not very easily readable, so another way had to be created to interpret the data.

The purpose of the thesis is to create a program that reads the log files created by the DSiP system and modifies the data traffic amounts in those log files to more easily understandable visual form. Also the program is required to support multiple languages and adding a new language does not require any code changes to the program itself. The program should also be executable on multiple platforms.

The program is written in the Java programming language in the Netbeans integrated development environment. With the help of this program, the user can choose the required log file, and define the start time as well as the timeframe. With these values, the program can read from the log file the values and draw them as a column graph. The program will also write the data to a separate tab. Both the column graph and the written text are color coded with three different colors, red, yellow and green. The border values of these colors can also be changed in the configuration menu. Changing the language is simple and does not require any code changes to the program.

The program has achieved all the functional requirements set for it and its source code has been handed over to the customer. It is still unclear if the company will start using the product, or if they will refine it even further.

Key words: DSiP, Multichannel-routing, Java, visualization

Sisällys

1 Johdanto

Tämän opinnäytetyön aiheena on monikanavajärjestelmän lokitiedostojen liikennemäärien visualisointi. Tällä hetkellä järjestelmän tuottamat lokitiedostot ovat puhtaassa tekstitiedostomuodossa ja vielä sellaisessa muodossa, josta niitä on erittäin vaivalloista ja vaikeaa lähteä tarvittaessa tutkimaan. Nämä lokit muodostavat ongelman, kun tarvitaan nopeaa ja koottua yleiskatsausta järjestelmän toiminnasta. Myös tarkempaa tietoa on vaikea lähteä katsomaan lokista sen sekavan muodon vuoksi. Tähän pyritään kehittämään ratkaisu tämän opinnäytetyön kautta.

Tämän opinnäytetyön tuotos keskittyy visualisoimaan järjestelmän tuottamia lokeja ja muuttamaan ne sellaiseen muotoon, että mahdolliset poikkeamat ovat helppo nähdä ja tunnistaa. Lisäksi opinnäytetyö tulee käsittelemään lokien muuttamista yksinkertaiseen raporttimuotoon siten, että lokien sisältö on selvemmin esitetty kuin tällä hetkellä. Tämä opinnäytetyö liittyy lähemmin Laurea-ammattikorkeakoulussa vuonna 2010 aloitettuun Mobi-hankkeeseen. Hankkeen tarkoituksena on muuttaa tapaa, miten viranomaisten ajoneuvoissa yhdistetään laitteiden sähkönkulutusta ja kommunikaatiovalmiuksia. Mobi-hanketta käsitellään myöhemmin tässä opinnäytetyössä.

2 Työn lähtökohdat

Opinnäytetyöt voidaan yleisesti jaotella ammattikorkeakouluissa tutkimuksellisiin ja toiminnallisiin opinnäytetöihin. Yleisesti tutkimuksellisia opinnäytetyöitä painoitetaan eri ohjeistuksissa ja oppaissa toiminnallisia enemmän, mutta yritysmaailmasta on tullut selvä toive saada työelämää konkreettisesti kehittäviä hankkeita (Vilkka & Airaksinen 2009, 5).

Tämä opinnäytetyö on tehty toiminnallisena opinnäytetyönä. Vilkan ja Airaksisen mukaan toiminnallisessa opinnäytetyössä ei tarvitse käyttää mitään tiettyä tutkimusmenetelmää. Kuitenkin he mainitsevat samalla, että tutkimusmenetelminä voidaan käyttää joko määrällistä tai laadullista tutkimusmenetelmää (Vilkka & Airaksinen 2009, 56-57).

Määrällistä tutkimusmenetelmää käytetään silloin, kun opinnäytetyöhön tarvitaan mitattavaa, tilastollista tietoa, joka esitetään numeraalisesti. Ennen määrällisen tutkimusmenetelmän käyttöä, työn tekijän pitää miettiä ja tutkia, onko selvitystä tarvitseva kohde mitattavissa. Yleinen keräystapa tähän tutkimusmenetelmään on lomake. Tämä lomake lähetetään kohderyhmälle ja saaduista tiedoista muodostetaan numeraalista tietoa (Vilkka & Airaksinen 2009, 58).

Laadullisessa tutkimusmenetelmässä tavoitellaan ilmiön kokonaisvaltaista ymmärtämistä. Lähtökohtana voi toimiva työn tekijän halu saada kirjoittamatonta faktatietoa tai tavoite toteuttaa kohderyhmä näkemys ideasta. Laadullisessa tutkimusmenetelmässä aineistoa kerätään joko yksilö- tai ryhmähaastatteluna. Näissä voidaan käyttää joko lomake- tai teemahaastattelua. (Vilkka & Airaksinen 2009, 63.)

Tässä työssä käytetään laadullista tutkimusmenetelmää. Aineistona toimii Mikko Bertlingin suorittama, Ajeco Oy:n toimitusjohtajan, John Holmströmin haastattelu, joka on tässä työssä liitteenä 1 (Liite 1).

2.1 Keskeiset käsitteet

Tässä luvussa esitetyillä käsitteillä luodaan pohjaa opinnäytetyön teoreettiselle viitekehykselle. Tavoitteena on avata käsitteet lukijalle, jolla ei mahdollisesti ole aikaisempaa tietämystä tietotekniikasta.

2.1.1 Java

Java on Sun Microsystemsin vuonna 1995 julkaisema ohjelmointikieli ja -alusta (Oracle). Se syntyi osana tutkimusprojektia, jonka tarkoitus oli luoda kehittynyt ohjelma, joka toimisi useassa eri tietoverkkolaitteessa ja upotetuissa järjestelmissä. Tämän ohjelman syntymiseen tarvittiin toiminta-alusta, joka olisi kevyt, luotettava, siirrettävä, jaettava sekä reaaliaikainen. (Sun Microsystems 1997a.)

Java-ohjelmointikieli on suunniteltu alusta alkaen olemaan olio-pohjainen kieli, tarkoittaen että ohjelma rakennetaan käyttämällä erillaisia objekteja, olioita (Sun Microsystems 1997b). Objektit ovat ohjelman ohjelmointimalleja, joilla on tila ja tietty käyttäytyminen (Sun Microsystems 1997c). Javalla kirjoittamiseen ei tarvitse paljon koulutusta, sillä kieli on hyvin yksinkertainen. Jo kokeneille C++-kielen taitajille Java tuntuu tutulta näiden kahden kielen läheisen suhteen takia. Javan kehityksessä ottettiin oppia useista eri kielistä, kuten C++:sta. Kuitenkin Javan tärkein vahvuus on sen siirrettävyys useiden eri alustojen ja käyttöjärjestelmien välillä. Tämä onnistuu siten, että Java Compiler-kääntäjä kääntää kielen tavukoodiksi, joka on alustariippumaton formaatti.Tämän tavukoodin pystyy suorittamaan halutulla alustalla, kunhan siinä on Java-virtuaalikone(JVM). JVM suorittaa koodin ja muuntaa sen binäärikoodiksi, jotta tietokone ymmärtää sitä. (Sun Microsystems 1997b.)

2.1.2 Mobi-hanke

Tekniikan kehittyessä ajoneuvoihin tulee uusia laitteita, jotka tukevat viranomaisten työtä. Tämä kehitys on nähty esim. tietokoneiden ja satelliittipaikannuksen tulon myötä. Mitä enemmän laitteita viranomaisten ajoneuvoihin laitetaan, sitä hankalammaksi tulee näiden laitteiden virransaannin ja yhteensopivuuden sekä matkustajien turvallisuuden varmistaminen ajoneuvossa. MOBI-hankkeen tarkoituksena on kehittää standardi hälytysajoneuvojen ICT-laitteille ja näin ollen varmistaa niiden toimivuus ja matkustajien turvallisuus. Projekti alkoi vuoden 2010 syksyllä, ja se kestää kolme vuotta. Laurea-ammattikorkeakoulun lisäksi hankkeessa on mukana myös useita muita toimijoita ja viranomaisia sekä teknologian ja innovaatioiden kehittämiskeskus TEKES. (Hult & Rajamäki 2011.)

MOBI (Mobile Object Bus Interaction)-hankkeeseen kuuluvat laitteistot ovat kaikki viranomaisajoneuvossa olevat ICT-laitteet kaapeloinnista ja tietokoneista tulostimiin ja ääni- ja datakommunikaatiolaitteisiin. Hanke on jaettu kahteen teolliseen projektiin ja yhteen tutkimusprojektiin. Teollisista projekteista vastaavat teollisen puolen yhteistyökumppanit ja tutkimusprojektissa ovat mukana muut tutkimuslaitokset ja yrityskumppanit Laurean johdolla. Projektin tavoitteena on varustaa demoajoneuvo toimivalla ICT-laitteistolla, joka perustuu tässä projektissa toteutettuun tutkimukseen ja kehitykseen. Lisäksi on tarkoitus luoda kaupallistumissuunnitelma valmiille tuotteelle. Suunnitelmana on tarjota tuotetta ainakin Euroopan markkinoille. (Hult & Rajamäki 2011).

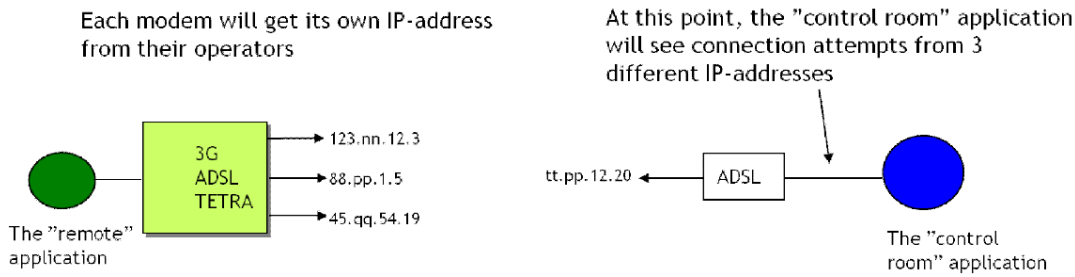
2.1.3 Lokitiedosto

Loki on tiedosto, johon järjestelmä tai palvelin tallentaa olennaisia tietoja sen toiminnasta. Sen tarkoituksena on toimia vianetsintävälineenä, jos järjestelmässä tulee jonkinlainen ongelma. Lokitiedostot ovat erinomaisia tähän varsinkin monimutkaisissa järjestelmissä, joissa mahdollisia ongelmia voi olla useita ja ongelman selvittämisen pitää olla nopeaa. Lokitiedosto tallentaa tiedon ilman ihmisen vuorovaikutusta mutta sen tallentamat tiedot eivät aina ole helposti luettavissa. Tämän takia lokitiedostojen lukemiseen on erillinen ohjelma, joka pystyy tehokkaasti lukemaan ja kokoamaan lokitiedoston tiedot sekä näyttämään ne käyttäjälle selvässä muodossa. (Brick Marketing 2011.)

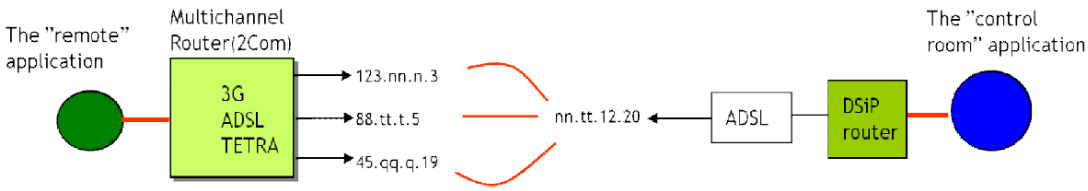
Lokitiedostot ovat yleensä käyttäjältä piilossa ja niitä eikä niitä huomaa ennenkuin ne kasvavat liian isoiksi, sisältäen vain turhaa tietoa.Myös ongelmatilanteissa lokitiedostot on helppo huomata ja juuri tällöin ne pääsevät arvoonsa. Esimerkiksi haittaohjelmien pääsy järjestelmään jättää lokiin jäljen. Lokeja tutkimalla voi myöhemmin selvittää, miten haittaohjelma pääsi järjestelmään alun perin ja miten mahdollinen tietoturva-aukko tukitaan. (Grimes 2009.)

2.1.4 DSiP-ohjelmisto

Distributed Systems intercommunication Protocol (DSIP) on ohjelmisto, jonka tarkoituksena on taata luotettava tiedonsiirto ja turvallisuus verkossa. DsiP:n tärkein ominaisuus on sen tarjoama mahdollisuus monikanavaiseen tietoliikenteen reitittämiseen. Monikanavareititys tarkoittaa usean eri tietoliikenneyhteyden kautta siirtyvää liikennettä. IP-protokolla ei itsessään tue monikanavareititystä mutta useat eri valmistajat ovat luoneet ohjelmistoja, joilla voidaan taata kriittisten tietoliikenneyhteyksien toiminta. Monikanavareititys auttaa parantamaan luotettavuutta varsinkin hälytysajoneuvojen sisäisessä tietoliikenteen siirtämisessä. Ajeco OY:n kehittämällä ohjelmistolla pystytään reitittämään erilaisia tietoliikenneprotokollia, IP- ja ei-IP-pohjaisia. Tämä onnistuu siten, että kun paketti tulee DSiP-verkkoon, se kapseloidaan DSiP-paketin sisään, jotta se voidaan reitittää DSiP-verkon sisällä. DSiP-verkossa olevat laitteet pystyvät saumattomasti kommunikoimaan toistensa kanssa välittämättä minkälainen tekninen ratkaisu on toisessa laitteessa. Kuvio 1 esittää, miten reitytys toimii perinteisessä monikanavareititys-järjestelmässä. Kuvio 2 esittää, miten Ajecon kehittämä DSiP-järjestelmä hoitaa monikanavareitityksen. (Holmström, Knuutila & Rajamäki 2011).



Kuvio 1: Perinteisen monikanavareititys-järjestelmän toimintaperiaate



Kuvio 2: DSiP-Järjestelmän monikanavareititys

Verrattuna perinteisiin TCP/IP ratkaisuihin, DSiP-ohjelmisto tuo kolme konkreettista parannusta,varsinkin liittyen Mobi-projektin vaatimuksiin. Ensimmäinen näistä on turvallisuus. DSiP-verkon liikenteen reitityksessä käytetään SSL(Secure Soccet Layer)-yhteyttä. Lisäturvaa saa ohjaamalla tiedonsiirron VPN-tunnelin kautta. Lisäksi DSIP-protokollaa käyttäen voidaan jakaa yksi VPN-tunneli usean eri fyysisen median kanssa ilman mitään erillisiä

uudelleenavauksia tai siitä johtuvaa viivettä. Näin ollen VPN linkki pysyy rikkoutumattomana ja siirtyy vaihtoehtoiseen reittiin, jos alkuperäinen reitti katkeaa. (Holmström, Hämäläinen, Lehtonen, Nordman & Ramstedt 2003; Holmström ym. 2011).

Toisena parannuksena on palvelun laadun parannus. DSiP-järjestelmä pystyy heti reitittämään liikenteen uudestaan, jos yhteys katkeaa. Lisäksi DSIP-reitittimet päivittävät jatkuvasti listaa verkossaan olevista solmuista. DSIP-paketissa on myös keep-alive ja syke mekanismit turvaamaan, että paketti ei siirron aikana katoa, kuten voi tapahtua TCP/IP verkossa. (Holmström ym. 2003).

Kolmas konkreettinen parannus liittyy datan laatuun verkossa sekä solmujen lähettämään datan määrään. DSiP vähentää turhan tiedon jakamista, varsinkin multi-cast liikennettä. Solmujen liikenteen määrää voidaan kontrolloida, varsinkin sitä, minkälaista liikennettä tietyn yhteyden yli siirtyy. Esimerkiksi laajakaistayhteyden, 3G:n tai vaikka WiMAX:in yli pystytään siirtämään videokuvaa kun taas alhaisen kaistanleveyden omaavien yhteyksien ja varsinkin niiden,joiden datasiirtokustannukset ovat korkeita,kuten satelliitin, yli tulisi siirtää vain välttämätöntä, erittäin pientä dataa. (Holmström ym. 2003).

2.1.5 UML

UML, Unified Modelling Language, on standardisoitu mallinnuskieli, joka koostuu diagrammeista. Nämä diagrammit on kehitetty auttamaan järjestelmien ja ohjelmistojen kehittäjiä määrittämään, visualisoimaan, rakentamaan ja dokumentoimaan kehitteillä olevan järjestelmän tai ohjelman. Lisäksi se auttaa käyttäjiä ohjelmiston tai järjestelmän arkkitehtuurisessa suunnittelussa sekä simulaatiossa ja testauksessa. UML kehitettiin alun perin suosimaan kommunikaatiota ja tuotteliaisuutta olio-pohjaisten järjestelmien kehittäjien parissa. Kuitenkin siitä on kasvanut vahva yleinen mallinnuskieli järjestelmien ja ohjelmistojen parissa. (Chonoles & Schart 2003, 9-10)

Mallinnuskieli on hyvin käytännöllinen, sillä sen avulla pystytään tehokkaaseen kommunikaatioon järjestelmää tai ohjelmaa kehittävien jäsenien kesken. UML-kaaviot ovat helposti ymmärrettäviä; niiden avulla kehittäjät saavat järjestelmän toiminnasta kokonaiskuvan. Lisäksi kehittäjä voi pyytää muilta alalla toimivilta ihmisiltä mielipiteitä ja palautetta järjestelmästä vain näyttämällä UML-mallinnuskielellä tehtyjä kaavioita. Mielipiteiden ja palautteiden pohjalta kehittäjä voi kehittää ohjelmistoaan paremmaksi ja sulavammaksi. (Chonoles & Schart 2003, 10)

UML-kaavio luo abstraktin mallin järjestelmästä. Se on kehittäjän päätettävissä, miten tarkasti hän haluaa järjestelmän kuvata. UML ei koostu pelkästään yhdestä kaaviosta vaan se

rakentuu monista, eri tarkoituksiin suunnitelluista kaavioista. Näitä kaaviota on virallisesti 13 erilaista ja ne voidaan jakaa karkeasti kolmeen ryhmään: rakennekaavioihin, käyttäytymiskaavioihin sekä vuorovaikutuskaavioihin. Rakennekaavioihin kuuluvat esimerkiksi luokkakaavio sekä komponenttikaavio, käyttäytymiskaavioihin kuuluvat käyttötapauskaavio ja toiminnallisuuskaavio, ja vuorovaikutuskaavioon kuuluvat kommunikaatiokaavio sekä sekvenssikaavio. (Chonoles & Schart 2003, 10-12)

UML ei ole prosessi eikä metodi eikä sen käyttö vaadi minkäänlaisen yhtenäisten prosessin, unified process, käyttämistä. Sitä pystytään käyttämään erillisenä sillä on mahdollista että mikään dokumentoitu tai yleinen prosessi ei sovi kehittäjälle. (Chonoles & Schart 2003, 17)

2.1.6 Käyttöjärjestelmä

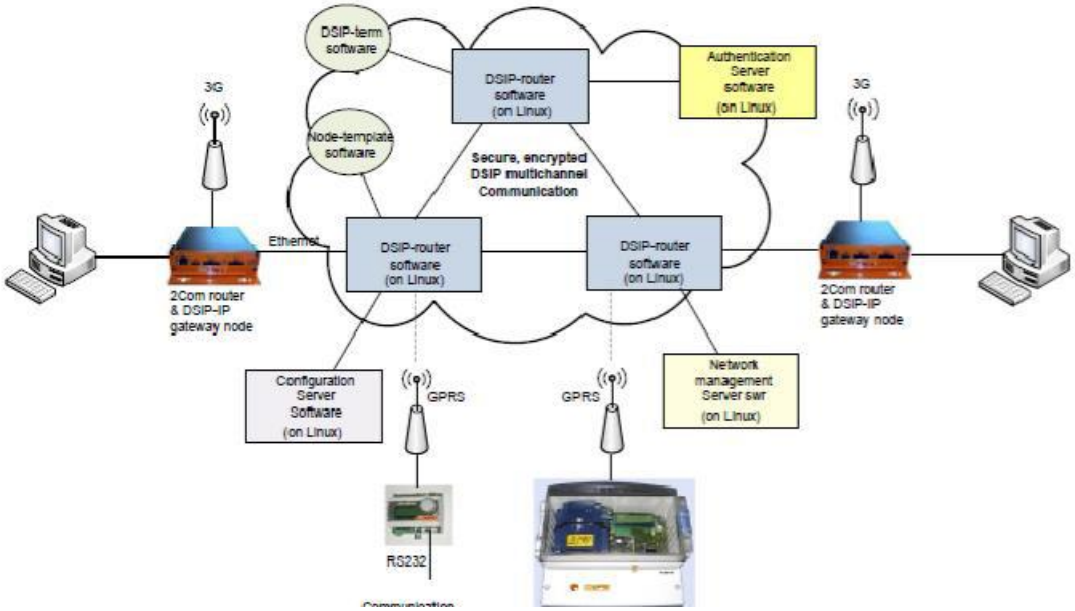
Käyttöjärjestelmä on tietokoneessa toimiva ohjelma jonka tarkoitus on suorittaa muita ohjelmia. Se on koko koneen tärkein ohjelma, hallinnoiden sekä ohjelmiston että laitteiston resursseja. Käyttöjärjestelmä kontrolloi kaikkea tietokoneessa tapahtuvaa aina muistin jaosta kuvan ohjaamisesta näytölle sekä kovalevyillä olevien tiedostojen hallintaan. (Madison 2011.)

Käyttöjärjestelmiä on useita, vaihdellen aina käyttötarkoitusten perusteella. Kuluttajalle tutuimmat käyttöjärjestelmät ovat Linux, Applen Mac OS X ja Microsoft Windowsin tuoteperhe, sisältäen vanhentuneet Windows XP:n ja Vista:n sekä uusimman Windows 7:n. Näistä käyttöjärjestelmistä suljettuja, tarkoittaen että käyttöjärjestelmän koodi ei ole muokattavissa, ovat Applen Mac OS X ja Microsoft Windowsin tuoteperhe. Avoimen lähdekoodin Linux on kaikkien vapaasti muokattavissa ja siitä on tullutkin useita eri versioita, joista suurin osa on ilmaisia.

2.2 Projektin suhde aikaisempiin hankkeisiin

Opinnäytetyö liittyy konkreettisesti Laureassa aikaisemmin tehtyyn DSiP-demoympäristö hankkeeseen. Kyseisessä hankkeessa opiskelijaryhmä rakensi toimivan DSiP-ympäristön. Ympäristössä käytetään kolmea konetta, jotka toimivat linux-käyttöjärjestelmillä. Ympäristöön kuuluvat seuraavat ohjelmistot ja laitteet: jokaisessa koneessa on oma reititin ja näistä kolmesta koneesta yhdessä on järjestelmän autentikaatiopalvelin, toisessa konfiguraatiopalvelin ja kolmannessa verkonhallintapalvelin. Lisäksi ympäristöön kuuluu muutama Ajecon toimittama, DSIP-järjestelmän avulla kommunikoiva laite. Tämä demoympäristö tuottaa lokitiedostoja, joita tässä opinnäytetyössä pyritään visualisoimaan.

Demoympäristö kuvataan kuviossa 3.



Kuvio 3: DsiP-Demojärjetestelmä Laurea-ammattikorkeakoulun tiloissa

Tämä opinnäytetyö sivuaa hieman Mikko Bertlingin aikaisempaa opinnäytetyötä. Molemmissa töissä pohjimmaisena aiheena on DSIP-järjestelmän lokitiedostojen visualisointi, erona kuitenkin se, minkälaisesta lokitiedostosta näitä tietoja visualisoidaan. Bertlingin aiheena oli yhteysmäärien visualisionti ja tässä työssä aiheena on liikennemäärien visualisoiniin.

2.3 Opinnäytetyön ominaisuudet

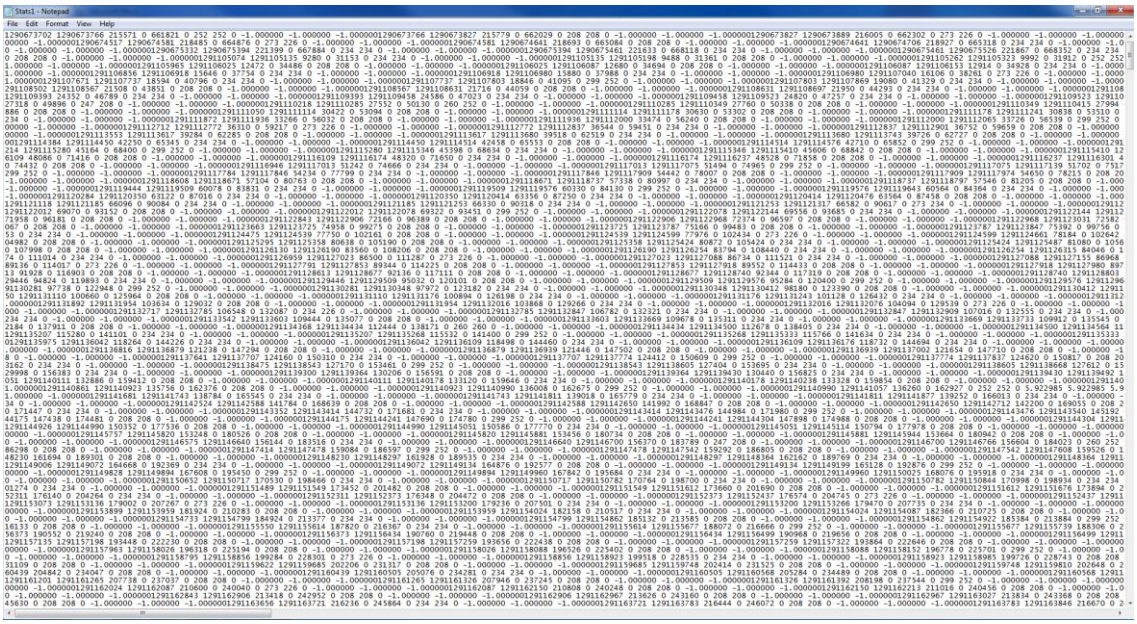
Opinnäytetyö on yksilötyö eli tekijä vastaa kaikesta, mitä tämän työn kanssa tapahtuu. Tämän opinnäytetyön asiakkaana toimii DSIP-järjestelmän luonut Ajeco Oy.

Ainoa sidosryhmä tälle opinnäytetyölle on pääkäyttäjä, joka vastaa DSIP-järjestelmän ylläpidosta. Näitä henkilöitä tulee olemaan useita, sillä toisistaan erillisillä DSIP-verkkoilla on jokaisella on omat pääkäyttäjänsä. Vaikka näitä pääkäyttäjiä tulee olemaan fyysisesti useita, tullaan heitä käsittelemään vain yksikkönä, sillä heidän jokaisen tärkeät, ohjelmaan liittyvät koulutukset, tulevat olemaan samanlaisia. Nämä Pääkäyttäjät tulevat olemaan DSIP-järjestelmän ammattilaisia, jotka osaavat käsitellä järjestelmää.

2.4 Toiminnallinen viitekehys

Tämän opinnäytetyön tavoitteena on luoda ohjelma, joka visualisoi edellä mainitusta DSIP-järjestelmästä muodostuvia lokitiedostoja. Näitä lokitiedostoja muodostuu yhteysmääristä, liikennemääristä sekä latenssimääristä. Liikennemäärät kuvaavat liikennettä reitittimien ja

solmujen eli nodejen välillä. Järjestelmä huolehtii automaattisesti lokien luomisesta, ja jo valmiit lokit pysyvät siellä muokkaamattomina kunnes käyttäjä poistaa ne. Ohjelma tulee lukemaan näistä järjestelmän luomista tekstitiedostoista tekstiä ja muokkaamalla sen raporttimuotoon, josta se halutessa visualisoidaan. Kuvio 4 näyttää missä muodossa lokitiedoston tiedot ovat ennen visualisointia.



Kuvio 4: Lokitiedostojen sisältämän tieto

Itse opinnäytetyö kuuluu osana Laurea-ammattikorkeakoulussa syksyllä 2010 alkaneeseen Mobi-hankkeeseen, jonka tarkoitus on luoda yhtenäinen ICT-infrastruktuuri kaikille hälytysajoneuvoille. Tässä projektissa käytetään eräänä osana Ajecon luomaa DSIP-järjestelmää mahdollistamaan saumaton tietoliikenneyhteys kaikissa tilanteissa ympäri Suomea.

2.5 Omat oppimistavoitteeni

Omat oppimistavoitteeni on tutustua kolmeen aiheeseen, joita tässä opinnäytetyössä käsitellän: Java, UML ja sovelluskehitys. Java tulee näistä kolmesta tärkeimpänä. Java-kielen osaaminen on hyvä taito työelämässä, sillä se on laaja-alainen ja monissa eri alustoissa toimiva ohjelmointikieli. Sitä pystytään käyttäämään sekä normaaleissa työasemissa, palvelimissa että mobiililaitteissa. Java taipuu moniin eri käyttötarkoituksiin ja taitavalla kehittäjällä on varmasti kysyntää työmarkkinoilla.

Toisena tulee mallinnuskieli UML. Suurissa projekteissa, joissa on monta liikkuvaa osaa, tarvitaan yhteistä mallinnuskieltä, jonka avulla projekti toimii saumattomasti. UML:n osaaminen on hyvä tapa erottautua työmarkkinoilla. Viimeisenä listassa on sovelluskehitys.

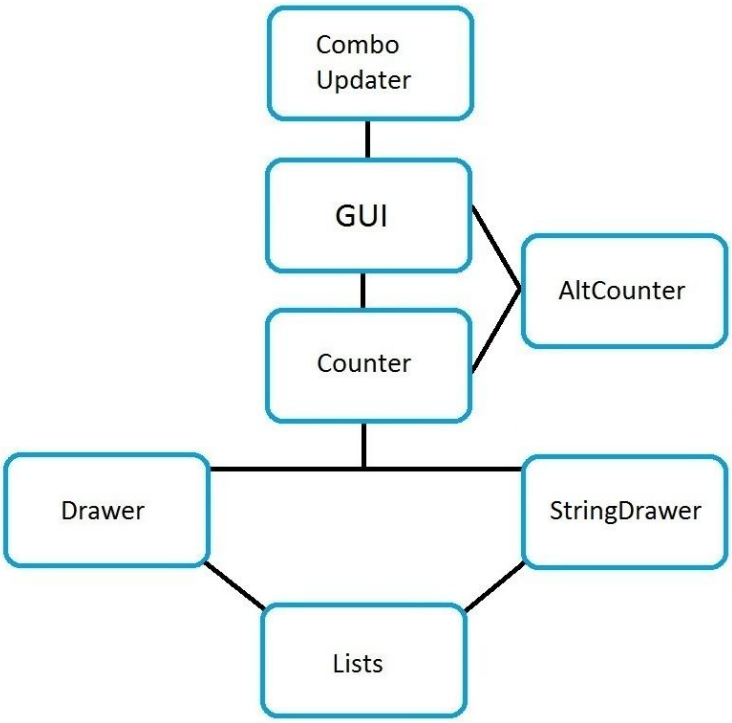
Koko sovelluskehityksen ja sen ympärillä olevien prosessien ymmärtäminen ja hallitseminen on tärkeää, jos haluaa toimia sovelluskehitysprojekteissa esimiehenä. Peruskoodaajalle on tärkeää tietää, missä vaiheessa mennään ja mitä sovelluskehityksen tekniikkaa kannattaa käyttää.

3 Toteutus

Toiminnallisen opinnäytetyön tärkein osa-alue on itse tuotos. Tämän opinnäytetyön toiminnallinen viitekehys määrittää tuotoksen tärkeimmän ominaisuuden. Tuotoksen pitää pystyä lukemaan DSiP-järjestelmän luomia lokitiedostoja ja piirtämään niistä liikennemäärät visuaaliseen muotoon. Tässä luvussa käsitellään tuotoksen eli ohjelman eri osat. Ohjelma on kirjoitettu käyttämällä Netbeans-ohjelmointiympäristöä.

3.1 Ohjelman luokat

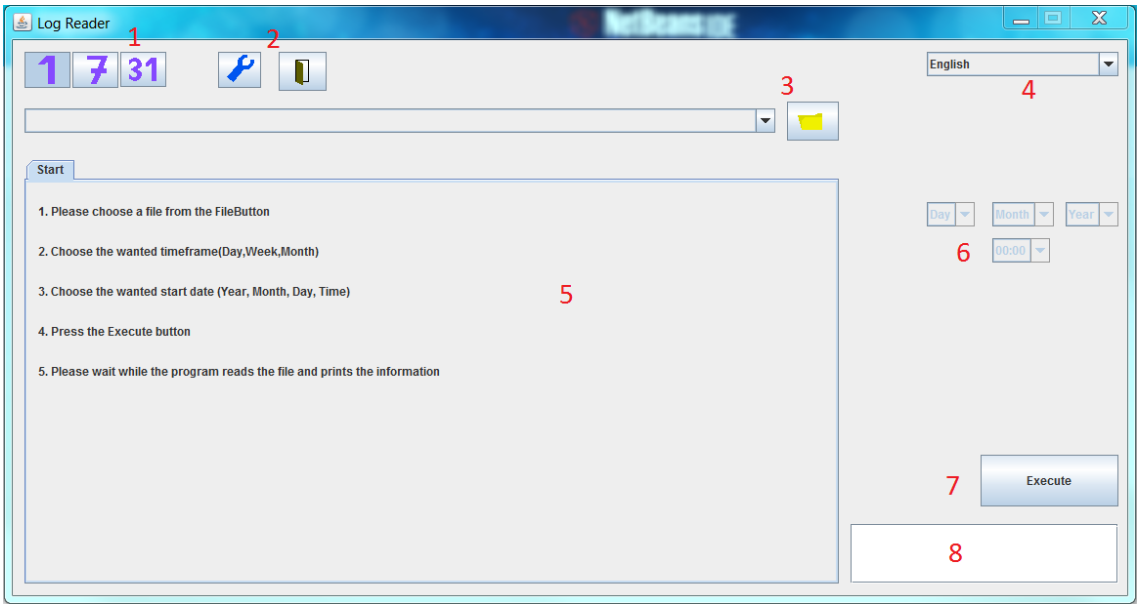
Ohjelmassa on yhteensä 23 eri luokkaa. Näistä 10 luokkaa voidaan sisällyttää kahteen isonpaan luokkakokonaisuuteen, Draweriin ja StringDraweriin. Molempaan kokonaisuuteen kuuluu viisi luokkaa. Kuviossa 5 havainnollistetaan ohjelman kannalta tärkeimmät luokat. Näitä tärkeitä luokkia on yhteensä 7, sisältäen nämä kaksi aikaisemmin mainittua luokkakokonaisuutta.



Kuvio 5: Havainnekuva ohjelman tärkeimmistä luokista

3.1.1 GUI

Ohjelmassa on graaffinen käyttöliittymä, Graphical User Interface (GUI), ja se on rakennettu omaksi luokakseen.Tämän käyttöliittymän avulla käyttäjä pystyy määrittämään mitä ohjelma tekee. GUI-luokka pitää sisällään kaikki käyttöliittymän toiminnallisuudet. Se on rakennettu Netbeansin mukana tulleella graaffisella käyttöliittymä-rakentajalla. Kuitenkin sitä on vahvasti muokattu, jotta se käsittää kaikki ohjelmaan liittyvät ja halutut toiminnallisuudet. Kuvio 6 on suora otos ohjelman käyttöjärjestelmän etusivulta. Tässä kuvassa näytetään punaisilla numeroilla käyttöliittymän tärkeimmät toimintakohdat.



Kuvio 6: Ohjelman aloitusikkuna

Käyttöliittymässä on kahdeksan tärkeätä kohtaa, ne on merkitty kuviossa 6 punaisilla numeroilla. Ne ovat seuraavat:

1. Aikamäärityspainikkeet. Näillä kolmella painikkella kontrolloidaan halutun aikavälin pituutta. Vaihtoehtoina ovat päivä, viikko ja kuukausi.

2. Konfiguraatio- ja lopetuspainikkeet. Konfiguraatiopainike vie käyttäjän konfiguraatioikkunaan, jossa käyttäjä pystyy määrittelemään raja-arvoja eri väreille ja pylväiden sekä Drawer näkymässä olevan vasemman puolen leveyttä. Lopetuspainikkeella käyttäjä voi sulkea ohjelman.

3. Tiedostojen alasvetovalikko ja tiedostonvalintapainike. Tiedostojen alasvetovalikolla voidaan valita nopeasti ohjelman käynnissäolon aikana jo luettuja tiedostoja.

Tiedostonvalintapainikkella käyttäjä pystyy valita, minkä tiedoston hän haluaa ohjelman lukevan.

4. Kielivalikko. Tätä alasvetovalikkoa käyttämällä pystytään valita ohjelmassa käytetty kieli ilman, että koko ohjelmaa täytyy käynnistää uudestaan.

5. Välilehtipaneli. Tässä panelissa on oletuksena pikainen aloitusohje käyttäjälle, joka muuttuu kieltä muuttaessa. Ohjelmaa käyttäessä tähän välilehtipaneliin tulevat Drawer- ja Stringdrawer-luokkien piirtämät palkki- ja merkkitaulukot.

6. Lähtöajankohta. Näiden neljän alasvetovalikon avulla käyttäjä pystyy määrittämään ajankohdan, josta lähtien ohjelma lähtee lukemaan lokitiedostojen arvoja. Kun näihin valikkohin on asetettu halutut arvot, ohjelma luo alasvetovalikkojen alle tekstin, jossa kerrotaan, mikä on näillä valituilla asetuksilla, aloitusajalla ja aikavälillä, lokitiedoston arvojen lukemisen lopetusajankohta.

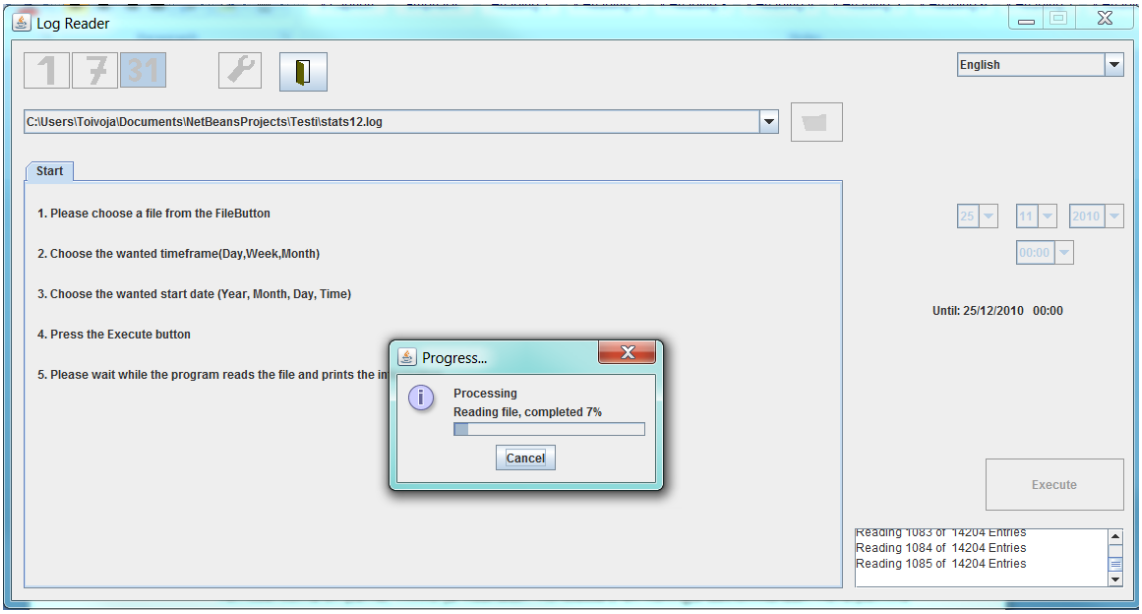
7. Lukijan käynnistyspainike. Tätä painiketta painettaessa ohjelma lähtee suorittamaan valitun lokitiedoston lukemista annetuilla arvoilla. Kuitenkin, jos tiedostoa ei ole vielä valittu, nuun painike vie käyttäjän valitsemaan luettavan lokitiedoston.

8. Yleinen tietoikkuna. Tämän ikkunan avulla ohjelma ilmoittaa missä kohdassa se on lokitiedoston lukemisen aikana.

3.1.2 Counter

Counter-luokan vastuulla on lukea lokitiedostossa olevat tiedot ja jakaa ne GUI-luokassa määritellyn ajankohdan mukaiseen Drawer- tai StringDrawer-luokkaan. Counter-luokka käynnistyy TaskManagerOwn-luokan kautta, joka käskee Counter-luokassa olevan swingworkerin lukemaan taustalla halutun lokin tietoja.

Samalla kun swingworker lähtee taustalla lukemaan tiedoston arvoja, tulee näytölle TaskManagerOwnissa luodun uuden progressmonitor-objektin ikkuna, jossa näytetään missä vaiheessa lukeminen on. Tämä näytetään kuviossa 7.



Kuvio 7: Lokitiedoston luku käynnissä

3.1.3 AltCounter

AltCounter-luokan tehtävänä on tukea sekä GUI- että Counter- luokkien toimintaa. GUI:ta varten luokassa suoritetaan valitun tiedoston ensimmäinen lukukerta. Tässä määritetään tiedoston aloitus ja lopetusarvot, jotka toimivat rajoituksena, kun määritetään mitä aikaväliä halutaan lukea.

Counter-luokka taas haluaa tässä luokassa olevalta metodilta tietoja kuinka monta merkintää halutulla aikavälillä on. Tätä tietoa tarvitaan, kun ohjelman muut osat lähtevät piirtämään taulukoita, sillä kyseisellä arvolla pystytään määrittämän pylvästaulukon leveys ja merkkitaulukon pituus.

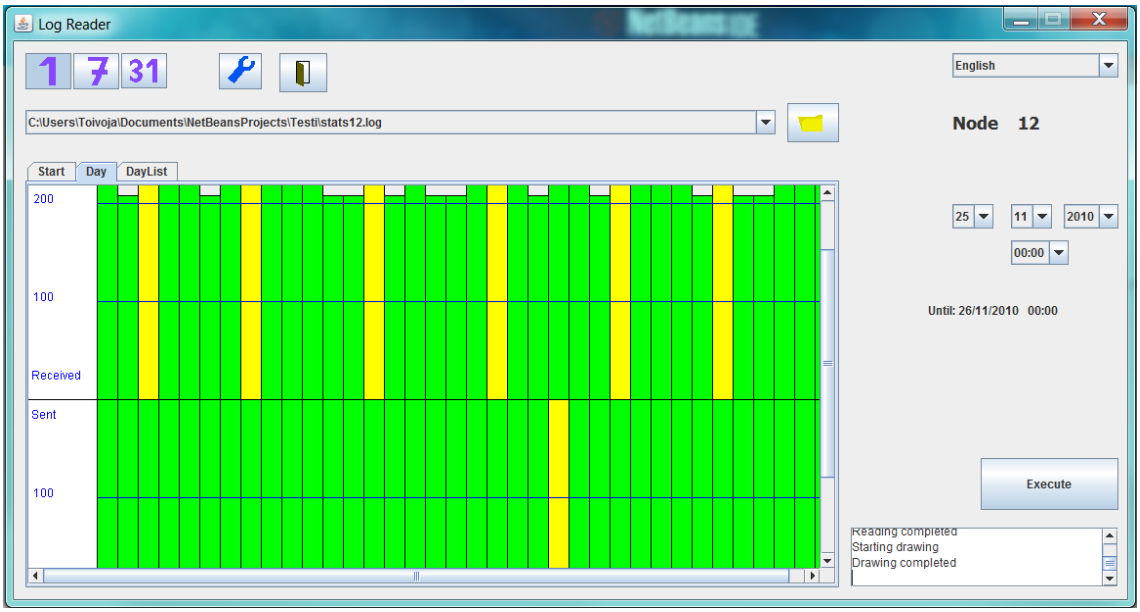
3.1.4 CompoUpdater

Toisin kuin AltCounter-luokka, on CompoUpdater luokka yksinomaan GUI:n käytössä. CompoUpdater-luokka sisältää useita eri metodeja, jotka kontrolloivat kuvion 6 kohdassa kuusi näkyviä, aikaa näyttäviä combolaatikoita. Tämän luokan ansiosta näiden ajanmääritys laatikoiden toiminta on sulavaa, varsinkin tilanteissa,joissa halutaan valita ajankohta läheltä tiedoston merkintöjen aikavälin alkua tai loppua. Tällä tavalla pystytään poistamaan riski, että käyttäjä valitsee aloituspäiväksi ajankohdan, joka ei ole tiedostossa olevien aikamääreiden sisällä.

3.1.5 Drawer

Drawer-luokkakokonaisuus sisältää viisi luokkaa, DrawerDay, -Week, -Month, -One ja – Gener luokat. Näistä Gener-luokka hallitsee muiden Drawer-luokkien yleisiä attribuutteja, toimien yleisenä luokkana tälle luokkakokonaisuudelle ja vähentäen muiden Drawer-luokkien koodin määrää ja siitä johtuvaa toistoa. Muut Drawer-luokat toimivat Gener-luokan aliluokkina ja toimien silloin kun niitä kutsutaan Counter-luokasta. Ohjelma käyttää Day-luokkaa, kun aikaväliksi on määritetty yksi päivä, Week-luokkaa, kun se on seitsemän päivää, ja Month-luokkaa, kun aikaväli on 31 päivää. DrawerOne-luokkaa käytetään silloin kuin halutulla aikavälillä on vain yksi arvo. Tämmöisiä tilanteita voi syntyä, kun lokitiedostoa luova node on ollut poissa päältä, ja aloitusajaksi määritetään juuri viimeinen hetki ennen noden sulkemista.

Drawer-luokkien tärkein tehtävä on luoda Lists-luokkaan tallennettujen arvojen pohjalta palkkeja kuvion 6 kohdassa viisi olevaan välilehtipaneliin ja luoda niille pieni tietoikkuna, jossa on oikeat arvot. Lisäksi luokissa on hiiren liikkeitä tutkivia metodeja, joiden avulla toimivat edellä mainitut tietoikkunat sekä siirto vastaavaan kohtaan merkkitaulukossa. Kuviossa 8 näkyy minkälaista tietoa luokka käyttäjälle piirtää.

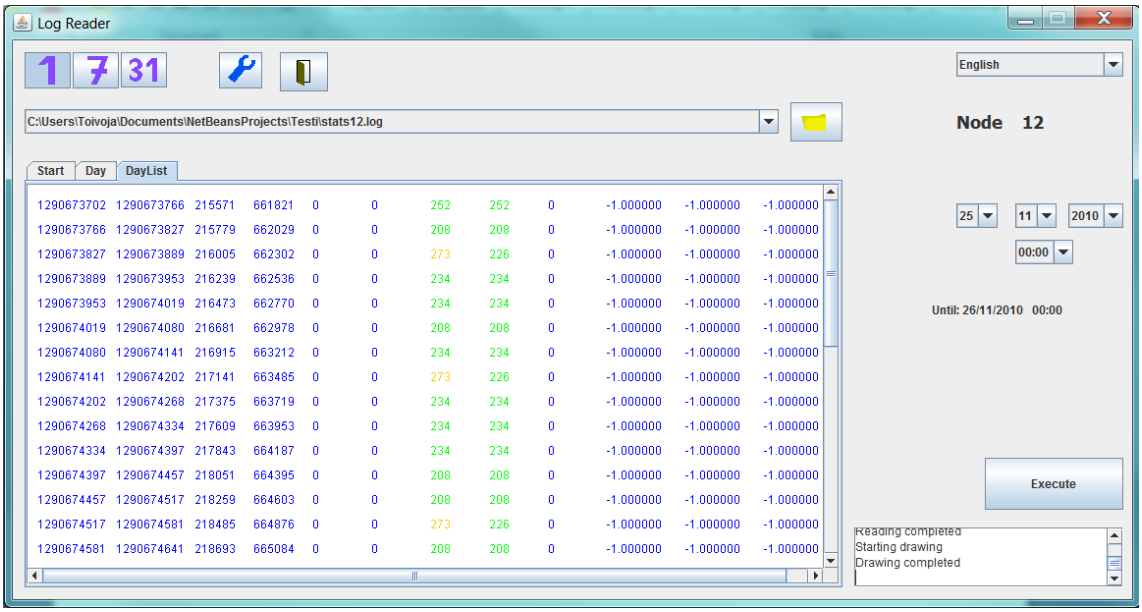


Kuvio 8: Piirrettävät arvot palkkeina

3.1.6 StringDrawer

StringDrawer-luokkien toimintaperiaate on sama kuin ylempänä mainittujen Drawer-luokkien, eli tässäkin on yksi yleisluokka, StringDrawerGener, ja neljä alaluokkaa, -Day, -Week, -Month ja –One. Kuitenkin näiden luokkien tärkein tehtävä on luoda, toisin kuin Drawer-luokkien,

merkkipohjainen taulukko Lists-luokan arvojen pohjalta. Luokassa on myös, Drawer-luokkien tapaan, hiiren liikkeitä tutkivia metodeja, joiden avulla toimivat myös tässä luokassa olevat tietoikkunat sekä siirto vastaavaan kohtaan pylvästaulukossa. Kuviossa 9 näkyy, että minkälaista tietoa luokka käyttäjälle piirtää.



Kuvio 9: Piirrettävät arvot tekstinä

3.1.7 Lists

Lists-luokka pitää sisällään jokaiselle Drawer- ja StringDrawer- luokalle taulukon, johon tallennetaan Counter-luokassa lokitiedostosta kerätty tieto. Nämä listat ovat jaettu vastaanotettujen ja lähetettyjen arvojen listoihin. Näin tehdään, jotta voidaan pitää arvot erillisenä, vaikka ohjelmaa pyydettäisiin hakemaan useita eri aikavälin palkkidiagrammeja välilehtipaneliin.Tällä varmistetaan palkkien pysyvyys, kun ohjelma piirtää sen uudelleen vaihdettaessa välilehteä sekä tietoikkunoiden arvojen paikkansapitävyys.

3.1.8 Muut luokat

Näiden pääluokkien lisäksi ohjelma pitää sisällään kahdeksan ohjelman toimintaa tukevaa, ja siihen lisää ominaisuuksia tuovaa luokkaa. Näitä ovat asetustiedostoja lukevat LanguageProperties ja ConfProps, ohjelman sisällä asetuksia varastoiva Configuration, asetuksen muokkaamisen mahdollistava ConfigurationFrame, tietynlaisia listoja luova FullListRecord, koko ohjelman käynnistävä Starter, progressimonitorin sisältävä TaskManagerOwn ja puuttuvien arvojen tilanteisiin vaadittu NoEntryPanel.

3.2 Ohjelmassa ongelmia tuottaneet vaiheet

Ongelmat ja esteet ovat osa kehitystä. Vain ongelmia ratkomalla ja esteitä ylittämällä voidaan saavuttaa jotain uutta ja kehittyä, oli se sitten yrityksen tietotaidon kehitystä tai vain ohjelman kirjoittajan kehittymistä. Myös tämän ohjelman kehityksen aikana kohdattiin monia ongelmia, joihin piti löytää ratkaisu. Nämä kuitenkin olivat enemmän yksilötason ongelmia, liittyen kiinteästi ohjelman kirjoittajan kehitykseen. Seuraavana käsitellään kolme ohjelman tärkeintä ongelmakohtaa.

3.2.1 Epoch

Ensimmäinen suuri ja mainittava ongelmakohta oli löytää oikea koodi, jolla pystyy muuttamaan epoch-aikaleiman helpommin luettavaan standardimuotoiseen aikaformaattiin. Epoch, tunnetaan myös paremmin nimillä Unix Epoch tai Unix-time, on ajanlaskennan tapa, joka on kehitetty täysin tietokoneiden ohjelmistojen tarpeisiin. Tämän ajanlaskennan perusideana on laskea sekunteja vuoden 1970 ensimmäisen päivän ensimmäisestä hetkestä lähtien ja tämän hetken aikajärjestelmänä käytetään koordinoitua yleisaikaa, UTC. Tämä mahdollistaa ajan käytön pelkästään yksinkertaisena kokonaislukuna ja antaa jokaiselle sekunnille oman arvonsa. Lisäksi tämä helpottaa ajan muuttamista erityyppisten kalentereiden, esim. Gregoriaanisten ja Juliaanisten, välillä sekä eliminoi kesäajan muutoksen tuoman ongelman, lukuunottamatta tilanteita, joissa aika pitää näyttää käyttäjälle. (Stover 2011)

Lokitiedostot sisältävät epoch aikaleiman, joilla pystyy ajallisesti seuraamaan lähetettyjen ja vastaanotettujen datojen määrää kronologisesti. Jotta näitä datoja pystyy järkevästi näyttämään, tulee nämä epoch-aikaleimat muuttaa ymmärrettävämpään standardimuotoiseen aikaformaattiin. Tämä onnistuu kuvion 10 kohdassa yksi olevalla koodilla. Kuitenkin ohjelman aloitusajankohdassa tarvitaan täysin vastakkaista, eli muunnosta standardimuotoisesta aikaformaatista epoch muotoiseen formaattiin. Kyseisen operaation vaatima koodi on kuvattu kuvion 10 kohdassa kaksi.

1
```
String date = new java.text.SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new java.util.Date (epoch*1000));
return date;
```

2
```
String date= MonthS + "/" +DayS + "/" +YearS+" "+TimeS+":00 GMT";
try{
SimpleDateFormat fmt = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss z");
      myDate = fmt.parse(date);
      epoch = myDate.getTime()/1000;
```

Kuvio 10: Epoch muutokseen tarvittavat koodit

3.2.2 Drawerit

Seuraavana ongelmakohta oli kuinka saada kerätty tieto näkymään oikein ohjelman näytöllä. Tähän kuuluu tiettyihin välilehtiin tulevat, datamääriä kuvaavat, värilliset palkit. Aluksi käytössä oli pelkkä JPanel, johon ohjelma piirsi lokien tiedoista kerättyjen datojen perusteella palkit. Tämä JPanel taas liitetään välilehtipaneliin, jonka avulla pystytään liittämään useita Jpaneleita samaan näkymään. Ongelmaksi tulee tilanne, jossa piirrettyjen palkkien yhteinen leveys ylittää välilehtipanelin leveyden.

Jscrollpane sisältää mahdollisuuden vierittää näkymää sekä horisontaalisesti että vertikaalisesti. Näin saadaan kaikki ohjelman piirtämät palkit näkymään tarvittaessa näytöllä. Jscrollpane toimii siten, että siihen liitetään JPanel ja tämä yhdistelmä taas liitetään välilehtipaneliin. JPanel on se, mikä toimii piirrettävänä näkymänä. Tämän panelin koko määräytyy piirrettävien palkkien määrän, koon ja korkeuden mukaan. Koska tämän panelin koko on suurempi kuin Jscrollpanen näkymäalueen koko, tekee Jscrollpane vierityspalkit sen mukaan, miten iso näkymässä olevan panelin koko on.

Kuitenkin eteen tulee useasti tilanteita, joissa piirrettäviä palkkeja on suuri määrä. Tämä aiheuttaa ongelman, kun koitetaan vierittää palkkeja. Koska näkymää vierittäessä JPanel jatkuvasti piirtää kaikki palkkinsa uudestaan, hidastaa se kohtuuttomasti ohjelman toimintaa. Ratkaisuna tähän, on rajoittaa Jpanelille piirrettävien palkkien määrää. Rajoittamalla tätä määrää saadaan ohjelman näkymä liikkumaan mahdollisimman sulavasti.

3.2.3 Counter

Ohjelman raskain työ on itse lokitiedostojen lukeminen. Käytettäessä suurta aikaväliä isossa lokitiedostossa tulee paljon luettavia tietoja. Ohjelman lukunopeus on kuitenkin vain noin 100 arvoa sekunnissa, jolloin jo yli tuhannen arvon sisältävässä aikavälissä käyttäjä tulee huomaamaan GUI:n jäätymisen näkymättömissä tehtävän työn takia.

Jotta käyttäjälle ei tule mielikuvaa että ohjelma ei tee mitään, on ohjelmaan kirjoitettu kaksi kätevää ominaisuutta. Ensimmäisenä on Swingworker-luokan toiminto. Tämän toiminnon avulla voidaan siirtää aikaa kuluttava laskentatyö pois Event Dispatch säikeestä, jossa GUI:n toiminnot tapahtuvat, taustalla toimivaan worker säikeeseen. Tällä saadaan GUI toiminnalliseksi, vaikka ohjelma työskenteleekin taustalla. Kuitenkaan tämä ei pakosti riitä, sillä pelkkä swingworker ei ilmoita käyttäjälle mitään toiminnastaan. Näin ollen pääsemme toiseen ominaisuuteen. Tämä on Progressmonitor-luokan toiminto, jonka avulla käyttäjä saa

eteensä pienen pop up-ikkunan. Tämän ikkunan avulla käyttäjä pystyy seuraamaan lukemisen etenemistä ja mahdollisesti peruttaa työn.

3.3 Ohjelman konfiguraatiotiedosto

Ohjelmaan on kirjoitettu toiminto lukemaan ennaltamääritettyjä konfiguraatiotiedostoja. Näiden tiedostojen avulla pystytään määrittämään käynnistyksen yhteydessä ohjelmalle arvoja, joita ohjelma käyttää ilman muita määrityksiä. Tämän ohjelman kohdalla konfiguraatiotiedostoissa on kahdenlaisia arvoja: kieliarvoja ja piirtämiseen tarvittavia arvoja.

Kieliarvot sisältävät määritykset, mitä käytetään, kun halutaan vaihtaa kieliä ohjelmassa. Jokaisella kielellä on oma konfiguraatiotiedosto ja niitä voidaan lisätä tarvittaessa. Konfiguraatiotiedosto, joka sisältää piirtämiseen tarvittavia arvoja, on toiminnaltaan erillainen kuin kieliarvoja sisältävä konfiguraatiotiedosto. Näitä piirtämiseen tarvittavia arvoja voi muokata ohjelman sisältä ja näillä arvoilla määritellään muun muuassa palkkien leveyttä sekä punaisen, keltaisen ja vihreän raja-arvoja.

3.4 Testaus

Ohjelman rakentamiseen kuului olennaisena osana sen jatkuva testaus. Jokaisen uuden komponentin jälkeen testattiin ohjelman toimivuus, jotta pystyttiin päättelemään, toimiiko komponentti oikealla tavalla, onko komponentista konkreettista hyötyä ja korjaamaan mahdolliset ongelmakohdat, joita komponentissa on tullut testauksessa esiin. Tämän avulla pystytään antamaan ohjelman tekijälle, ja tämän opinnäytetyön tekijälle, mahdollisuus testata epävarmuutta aiheuttavia kohtia koodissa. Tässä vaiheessa testaus tapahtui pääasiallisesti ohjelman luomiseen käytetyn integroidun ohjelmointiympäristön, Netbeans-ohjelman, avulla.

Kun ohjelman koodi oli kirjoitettu toiminnallisesti valmiiksi, oli aika testata ohjelmaa ja sen toiminnallisuutta. Testaus toteutettiin kahdella eri käyttöjärjestelmällä, Windowsilla ja Linuxin Ubuntu-jakelulla. Lisäksi testauksessa käytettiin suojaamatonta tekstitiedostoa, joissa on lokitiedoston tiedot. Tekstitiedosto on kuitenkin samanlainen, mitä DSIP-järjestelmä tuottaa. Tässä testausvaiheessa ei käytetty Netbeans-ohjelmaa, vaan se suoritettiin koneelle asennetun java-ohjelman kautta.

4 Arviointi

Ajeco Oy on määrittänyt haluamalleen ohjelmalle selkeät toiminnalisuuden edellytykset. Tämän ohjelman tavoitteet on yritys määrittänyt selvästi Mikko Bertlingin yritykselle lähetetyssä kyselyssä, joka on tässä työssä liitteenä (liite1). Ohjelman tulee toimia useammalla eri käyttöjärjestelmäpohjalla, jotta ei tarvitse rajata Network Management -serverin toimintaympäristöä. Lisäksi ohjelman pitää piirtää visuaalinen grafiikka halutusta lokitiedostosta tietyllä aikavälillä. Liitteessä olevassa haastattelussa on määritelty kolme aikaväliä, viimeiset 24 tuntia, viimeiset 7 päivää ja viimeinen kuukausi. (Holmström 2010).

Ohjelma tulee olla hyvin dokumentoitu. Siitä pitää olla yleiskuvaus, rakenne sekä konfiguraatiotiedostojen muuttujat ja merkitys selvästi dokumentoituna. Ohjelmoinnin, kommenttien ja dokumentoinnin kieleksi yritys toivoo olevan englanti. Lisäksi yritys pyytää, että ohjelmassa on mahdollisuus lisätä kieli pelkästään lisäämällä sen muuttujat konfiguraatiotiedostoon. (Holmström 2010).

4.1 Saavutetut vaatimukset

Ohjelma on saavuttanut kaikki siihen kohdistuneet vaatimukset. Ohjelma on kirjoitettu Javalla, jonka virtuaalikone JVM toimii usealla eri käyttöjärjestelmällä. Näin ollen ohjelmaa voidaan helposti siirtää käyttöjärjestelmien välillä, kunhan haluttuun tietokoneeseen on asennettu Java. Ohjelma myöskin pystyy suorittamaan sille annetun pääasiallisen tehtävän, eli piirtämään lokitiedostojen halutusta ajasta palkkeja, jotka kertovat siirtyneen datan määrän. Lisäksi siinä on määriteltynä kolme vaadittua aikaväliä, yksi päivä, seitsemän päivää sekä kolmekymmentäyksi päivää.

Ohjelmasta on luotu mallinnuskaavioiden avulla dokumentointi ja konfiguraatiotiedostojen muuttujat ja niiden tarkoitus on kirjattu ylös. Dokumentointikielenä on käytetty englantia. Samaista kieltä on käytetty myös itse koodissa sekä koodissa olevien kommenttien kanssa. Ohjelman viimeisessä ja valmiissa versiossa on mahdollista lisästä uusia kieliä vain lisäämällä siihen liittyvän kielen konfiguraatiotiedoston ja muokkamaalla yleistä konfiguraatiotiedostoa.

4.2 Toteutuksen onnistuminen ja ongelmat

Kun katsoo ohjelman toimintaa, on toteutus mielestäni onnistunut hyvin. Ohjelma on saavuttanut kaikki sille annetut vaatimukset, ja siitä on tehty niin sulava kuin on pystytty.

Toteutuksessa on ollut oikeastaan vain yksi ongelma, jos ei oteta huomioon ohjelmaan liittyviä ongelmia. Tämä ongelma on ollut aikataulun pitävyys. Alkuperäisenä suunnitelmana

oli, että ohjelma olisi ollut valmis jo keväällä 2011, mutta ohjelmantekijän motivaation puutteen vuoksi ohjelman ensimmäinen versio saatiin ulos vasta joulukuussa 2011.

4.3 Asiakkaan arvio toteutuksesta

Ohjelma on toimitettu arvioitavaksi Ajeco Oy:lle huhtikuussa 2012. Liitteessä 2 on kokonaisuudessaan Ajeco Oy:n toimitusjohtajan, John Holmströmin, arvio ohjelmasta. Ohjelman lisäksi Ajeco Oy:lle on toimitettu myös ohjelman HTML-dokumentaatio. ”Käsityksemme mukaan opinnäytetyö on selkeästi laadittu ja Alaverronen on perehtynyt hyvin sekä taustoihin että käytettyyn tekniikkaan. Ohjelmiston html-dokumentaatio on hyvä ja seikkaperäinen.” (Liite 2.)

Huolimatta ohjelmoinnin aikana tehdystä testauksesta, lopputestauksessa tuli ilmi muutamia virhekohtia. ” Testattuamme ohjelmistoa oikean lokitiedostodatan kanssa havaitsimme pieniä parennettavia kohtia mutta ottaen huomioon tehtävän määrittelyn emme katso, että nämä kehitettävät asiat olisivat esteenä kiitettävän arvosanan antamiseksi. Seikat, joihin kiinnitimme huomiota oli jos lokitiedosto sisälsi yli vuoden vanhaa dataa niin lähtöpäivämäärän valinnassa esiintyi pientä epämääräisyyttä. Toinen asia oli visualisointiohjelman piirtämien pylväitten värivaihtojen raja-arvojen asettelussa, joka mielestämme voisi olla suurempi.” (Liite 2.) Arviossa mainittujen parannuskohtien vuoksi ohjelmasta tehtiin versio 1.3, jossa edellä mainitut, parannusta vaativat kohdat, on korjattu. Tämä korjaus tehtiin samana päivänä, kun tieto parannettavista kohdista saatiin Ajeco Oy:n antamassa arviossa.

”Ottaen huomioon työn vaatimustason ja Alaverrosen itsenäisen työskentelyn esitämme omasta puolestamme kiitettävää arvosanaa ja pidämme Alaverrosta kehityskelpoisena henkilönä ohjelmistotekniselle alalle.”(Liite 2.) Huolimatta parannusta vaativista kohdista yritys on hyväksynnyt ohjelman ja määrittänyt sen kiitettäväksi.

4.4 Oman oppimisen arviointi

Oma oppimiskäyrä on ollut jyrkässä nousussa siitä lähtien, kun ohjelman kirjoitus alkoi. Olen mielestäni oppinut paljon Java-kielen toiminnasta sekä sen eri luokista. Myös olio-ohjelmointi on tullut minulle tämän opinnäytetyön aikana vahvasti tutuksi.Sovelluskehityksen periaatteita ja eri tekniikoita on tämän työn aikana käyty hyvin vähän läpi.
UML-tekniikoiden oppiminen on tämän opinnäytetyön aikana jäänyt vähäiseksi.

4.5 Tuotoksen jatkokehitysehdotukset

Koska lokitiedostoissa on myöskin latenssimäärät lueteltuina, olisi eräs mahdollinen kehitystyö, joko muokata kyseistä ohjelmaa siten, että se näyttää halutut latenssimäärät, tai luoda sille oma ohjelmansa. Tämän ohjelman muokkausta puoltaa se, että ohjelma tallentaa listoihin myöskin lokitiedostoissa olevat latenssimäärät. Niitä ei ole kuitenkaan implementoitu tämän ohjelman toimintaan. Toinen kehitysehdoitus olisi yhdistää jo valmis, Mikko Bertlingin luoma ohjelma tähän nykyiseen ohjelmaan, jolloin käyttäjä selviäisi yhdellä ohjelmalla.

4.6 Tuotoksen uudelleenkäyttö muissa projekteissa

Ohjelma on erittäin vahvasti muokattu Ajeco Oy:n vaatimusten mukaisesti. Tämä tekee tuotoksen jatkokäytön muissa projekteissa haastavaksi, mutta ei mahdottomaksi. Ohjelman jokainen luokka on suunniteltu toiminaan omana osa-alueenaan, pitämällä sisällään itselleen tärkeät metodit ja ominaisuudet. Tämä mahdollistaa luokkien uudelleenkäytön muissa ohjelmissa. Vaikka uudelleenkäyttö voi vaatia koodimuutoksia, on ne todennäköisesti vain minimaalisia.

Lähteet

Ajeco. Ajeco oy.
Viitattu 10.3.2011 http://www.ajeco.fi/dsip.php

Brick Marketing 2011. What is a log file? Viitattu 15.3.2011
http://www.brickmarketing.com/define-log-file.htm

Chonoles, M. & Schardt, J. 2003. UML 2 for Dummies. Indianapolis: Wiley Publishing.

Grimes R. 2009. Learn to love your log files. Viitattu 15.3.2011
https://www.infoworld.com/d/security-central/learn-love-your-log-files-251?page=0,0

Holmström, J. 2010 Toimitusjohtajan haastattelu 8.11.2010. Ajeco OY. Helsinki. Liite 1.

Holmström, J., Hämäläinen, P., Lehtonen, M., Nordman, M. & Ramstedt, K. 2003. A TCP/IP based communication architecture for distribution network operation and control. CIRED

Holmström, J., Knuutila, J. & Rajamäki, J. 2010. Robust mobile multichannel data communication for rescue and law enforcement authorities.

Hult, T. & Rajamäki, J. 2011. ICT integration of public protection and disaster relief (PPDR): mobile object bus interaction (MOBI) research and development project.

Madison, N. 2011. What is an operating system?
Viitattu 15.3.2011. http://www.wisegeek.com/what-is-an-operating-system.htm

Oracle. Viitattu 15.3.2011 http://www.java.com/en/download/faq/whatis_java.xml

Sun Microsystems 1997a. The Java language enviroment. Viitattu 15.3.2011.
http://java.sun.com/docs/white/langenv/Intro.doc1.html

Sun Microsystems 1997b. The Java language enviroment. Viitattu 15.3.2011.
http://java.sun.com/docs/white/langenv/Intro.doc2.html

Sun Microsystems 1997c. The Java language enviroment. Viitattu 15.3.2011.
http://java.sun.com/docs/white/langenv/Object.doc1.html#1354

Stover T. 2011 Advanced time considerations Viitattu 26.3.2012.
http://www.thomasstover.com/timearticle.html

Vilkka, H. & Airaksinen, T. 2003. Toiminnallinen opinnäytetyö. Jyväskylä: Gummerus Kirjapaino

Kuviot

Liitteet

Liite 1 John Holmströmin haastattelu

On 8.11.2010 16:55, Mikko Bertling wrote:
> Hei
>
> Olemme Samin kanssa päättäneet jakaa lokitiedostojen visualisoinnit niin, että minä toteuttaisin yhteysmäärien visualisoinnin opinnäytetyönäni. Koska Laureassa olevasta järjestelmästä ja komentolistauksista ei pysty päättelemään kaikkia tarvittavia tietoja, päätinkin ottaa yhteyttä suoraan teihin. Arvostaisin jos voisitte välittää kysymykset parhaiten vastaamaan kykenevälle asiantuntijallenne. Jos joitain tiedoista ei voida vielä tai ollenkaan kertoa, tietäisin siitä mielelläni mahdollisimman pian, jotta pystyn varautumaan siihen opinnäytettä tehdessäni.
>
> ----------
> Kysymykset:
>
> Onko asiakkaita joita voisi haastatella kytkentämäärien käytön kohteista/tarpeista?
>
> Onko tullut esille missä muodossa tietoja tahdottaisiin visualisoitavaksi? Kaikkien nodejen kytkennät? / Yhden palvelimen kytkennät? Millä aikavälillä nämä näytettäisiin?

Kun järjestelmää rakennetaan, kannattaa pitää mielessä se että olisi mahdollista kääntää se useammalle käyttöjärjestelmälle, tässä tapauksessa Linux ja Windows. Mikäli pitää käyttää alustariippuvaisia rutiineja kannattaa ne eristää välifunktion kautta omaan tiedostoon. Näin riittää kun tämän tiedoston porttaa, eikä tarvitse käydä etsimässä läpi koko ohjelmaa.

Mielestäni voisi toimia siten että valitaan yksi node tai router kerralla tarkasteluun olemassa olevista.

Näen ainakin kaksi tarkastelutarvetta:

1. Tutkitaan akuuttia ongelmaa. Eli lyhyt aikaväli
2. Seuraillaan yleistilannetta. Eli pitkä aikaväli

Olisi hyvä jos pystyisi valitsemaan, tai näytetään yhtä aikaa esim. viimeiset 24 tuntia, viimeiset 7 päivää ja viimeinen kuukausi.

Graafi olisi hyvä, helppo hahmottaa tilanne yhdellä silmäyksellä. Esim. voisi jakaa 24 tuntia 15 minuutin jaksoihin ja näyttää kytkeytymislukumäärä per 15 minuuttia janalla tai pylväällä tms. Lisäksi pitäisi keksiä joku hyvä tapa esittää milloin node on ollut järjestelmässä ja milloin ei. Voi olla tilanne jolloin node on poissa pidemmän aikaa, eli kytkentätapahtumia ei ole, mutta olisi hyvä tietää että tänä aikana ei ole ollut yhteyttä.

>
> Tämän lisäksi minua kiinnostaisi tietää miten lokitiedostojen resetoitumisväli vaikuttaa tietoihin (tyhjeneekö koko loki, alkaako luoda

alun päälle, uuteen tiedostoon)? Pystyykö tämän asetuksen tilaa tulkitsemaan järjestelmän ulkopuolelta ja tahdotaanko sen vaikuttavan jotenkin visuaalisen datan luontiin (esim. juuri aikaväliin)?

Kyseiset lokitiedoston ovat kumulatiivisia, eli eivät resetoidu automaattisesti.

>
> Olisiko mahdollista saada logiikka jolla lokitiedostot ja hakemistot rakennetaan? Ei tarvitse olla itse koodia, kunhan selkeästi kertoo minkälaisissa mahdollisissa muodoissa lokitietoja voi kohdata (ettei ohjelma joudu kohtaamaan tietoa jota ei ole sattunut generoitumaan Laurean verkkoympäristössä, mutta jota saattaa tulla vastaan muussa käytössä).

Lokihierarkia on:

```
DSiPNMServer
     |
-------------------------
  |                      |
Stats                   Logs
 |                       |
Statsnn                Nodenn
Statsmm                Nodemm
….                     Routernn
                       Routermm
                       …
```

Nodexxx.log tiedostossa on kolmen tyyppisiä rivejä:
Solmun kytkeytyminen verkkoon
05.10.2010 12:48:36:021 -Node connected to 192.168.1.81
Kahden tyyppisiä verkosta poistumista
05.10.2010 13:26:24:162 -Node disconnected from the router
05.10.2010 13:57:22:615 -Node disconnected (router-router disconnection)
jossa ensimmäinen on solmun yhteyden lopetus, ja toinen on solmun poistuminen siitä syystä että yhteyttä reitittimeen johon solmu on kytkeytynyt on katkennut.
Routerxxx.log tiedostossa on kahden tyyppisiä rivejä
Kytkeytyminen verkkoon
12.08.2010 16:01:41:664 -Router connected
Poidstuminen verkosta
12.08.2010 16:03:22:023 -Router disconnected
On myös olemassa NMAlarm.log joka sisältää aikaleimattuja tekstimuotoisia viestejä. Tätä sisältöä voi näyttää jos sovellukseen tekee jonkilaisen ”View log” toiminnon.
Statsxxx.log
Miittajakson alku - Mittajakson loppu - Summa lähetety tavut - Lähetys virhheet - Summa vastaanotetut tavut - Vastaanotto virheet - Vastaanotetut tavut mittajakson aikana - Lahetetyt tavut mittajakson aikana - Vastaamattomat keepalivet - Minimi latenssi - Keskimääräinen latenssi - Maksimi latenssi
1282050424 1282050488 144072 0 12771 0 60 1295 0 0.000701 0.000707 0.000711

Alku ja loppu aikaleima on sekunteja keskiyöstä 1 tammikuuta 1970

Summatavut ovat router kohtaiset ja vaihtuvat jos solmu kytkeytyy toiseen routteriin.

Latenssi mitataan keepaliven yhteydessä, koska keepalivejä lähetetään vain silloin kun ei ole somusta routeriin päin tulevaa liikennettä niin saattaa olla että mittajakson aikana ei ole lähetetty yhtään keepaliveä jolloin arvot ovat -1.0

Liikennöintiä voisi kuvata graafilla siten että se kuvaa tavuja / aikayksikössä. Aikajanat kuten kytkeytymis graafeissa.

Jotkut nodet lähettää myös tietoa liikenteestä eri interfaceilla, CommStatsnn_interfacename, esim. CommStats1_Eth.log. Ne sisältävät rivejä :
02.11.1970 21:38:35 338066 387028
Eli aikaleima , lähetetyt tavut , vastaanotetut tavut.

```
> Onko  Ajecolla joitain
ohjelmistonkehittämis/koodaus/nimeämis/dokumentointi käytänteitä joita
toivottaisiin käytettävän tässä projektissa? Ja tahdotaanko koodin
kommentoinnit mielummin englanniksi vai suomeksi? Jos ohjelmasta tahdotan
monikielinen, onko monikielisyyden toteuttamiseen valmista mallia
yrityksen sisältä?
> ----------------
```

Kuten aikaisemmin todettiin, kannattaa varautua siihen että kyseeseen tullee useampi alusta. Ohjelmoinnissa, funktio- ja muuttujanimissä olemme käyttäneet englannin kieltä. Kommentit myös mieluiten englanniksi mutta voi olla myös suomeksi. Dokumenttina pitäisi olla toteutuksen yleiskuvaus ja rakenne, sekä kofiguraatiotiedostojen formaatti muuttujat ja niiden merkitys. Funktion kuvaus voi laittaa koodiin funktion alkuun. Funktio ja muuttujanimet tulee valita siten että kuvaavat muuttuja sisältöä ja funktion tarkoitusta. Kielisyyskäsittelyyn ei ole ehdotonta toteuttamismallia, yleensä ne on kerätty ladattavaan tiedostoon jossa on avain – arvo pareja. Esim.
LANG_STORE ”Talleta”
LANG_OPEN ”Avaa”
Tällöin voidaan kääntää tämä tiedosto halutulle kielelle.

Liite 2 Ajeco Oy:n arvio ohjelmasta

Hei,

Olemme tutustuneet Sami Alaverrosen opinnäytetyöhön, jonka otsikko on "Liikennemäärien visualisointi
DSiP-järjestelmässä", huhtikuu 2012.

Käsityksemme mukaan opinnäytetyö on selkeästi laadittu ja Alaverronen on perehtynyt hyvin sekä taustoihin
että käytettyyn tekniikkaan. Ohjelmiston html-dokumentaatio on hyvä ja seikkaperäinen. Koska käytössämme
ei ole lähdekielistä koodia emme pysty lausumaan ohjelman teknisestä rakenteesta tarkemmin, mutta
koekäytössämme lokitiedostojen visualisointiohjelma toimi virheettä.

Testattuamme ohjelmistoa oikean lokitiedostodatan kanssa havaitsimme pieniä parennettavia kohtia
mutta ottaen huomioon tehtävän määrittelyn emme katso, että nämä kehitettävät asiat olisivat esteenä
kiitettävän arvosanan antamiseksi. Seikat, joihin kiinnitimme huomiota oli jos lokitiedosto sisälsi yli
vuoden vanhaa dataa niin lähtöpäivämäärän valinnassa esiintyi pientä epämääräisyyttä. Toinen asia oli
visualisointiohjelman piirtämien pylväitten värivaihtojen raja-arvojen asettelussa, joka mielestämme voisi
olla suurempi.

Ottaen huomioon työn vaatimustason ja Alaverrosen itsenäisen työskentelyn esitämme omasta puolestamme
kiitettävää arvosanaa ja pidämme Alaverrosta kehityskelpoisena henkilönä ohjelmistotekniselle alalle.

Ystävällisin terveisin,
John Holmström
Toimitusjohtaja.