Sakari Anttonen

KOODIGENERAATTORI TUKIASEMAOHJAIMEN OHJELMISTOLLE

Tietotekniikan koulutusohjelma

2014

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

KOODIGENERAATTORI TUKIASEMAOHJAIMEN OHJELMISTOLLE

Anttonen, Sakari Johannes
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Syyskuu 2014
Ohjaaja: Ismo Trast
Sivumäärä: 37
Liitteitä: 2

Asiasanat:
BSC, GSM, koodigeneraattori, Ohjelmistotuotanto,

Tämän opinnäytetyön tarkoituksena oli luoda Java-pohjainen ohjelmistokoodia generoiva työkalu Tieto Finland - yritykselle DX200 - tukiasemaohjaimen ohjelmiston tuotantokehitykseen. Samanlaista tai vastaavaa ohjelmaa yrityksessä ei ole saatavilla ja sen kehittäminen nähtiin tarpeellisena koodin kirjoitusasun sekä kommentoinnin standardoinnin kannalta.

Työkalu ja sen projektitiedostot tallennettiin yrityksen verkkolevylle. Työkalu kehitettiin Java-ohjelmointikielellä ja sen tarkoitus on luoda uusia koodirunkoja C, TNSDL sekä TDL kielillä. Työn ensimmäisessä vaiheessa työkalun asiakasvaatimukset määritettiin ja dokumentoitiin. Tärkeimmiksi vaatimuksiksi asetettiin käytön nopeus ja helppous. Toisessa vaiheessa kirjoitettiin vaatimusmäärittelydokumentti. Työkalua testattiin jatkuvasti toteutusvaiheen aikana.

Työkalu ja opinnäytetyön kirjallinen osuus tehtiin Tieto Finland Oyj tiloissa Porissa. Opinnäytetyön kirjallisessa osuudessa perehdyttiin yleisellä tasolla GSM-matkapuhelinjärjestelmään ja sen verkkoarkkitehtuuriin. Työssä tutustutaan myös Nokian DX200 tuoteperheeseen ja tukiasemaohjaimen rakenteeseen. Työssä tutkittiin ohjelmistotuotantoa, ohjelmistotuotannon elinkaarta ja ohjelmistotuotantoon yleisesti liittyviä vaiheita. Työn kirjallisen osan lopussa kuvattiin työkalun toteutusvaiheita aina aloituksesta käyttöönottoon asti ja pohdittiin sekä työkalun että opinnäytetyön onnistumista.

CODEGENERATOR FOR BASE STATION CONTROLLER SOFTWARE

Anttonen, Sakari Johannes
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
September 2014
Supervisor: Ismo Trast
Number of pages: 37
Appendices: 2

Keywords: BSC, GSM, code-generator, Software development

The purpose of this thesis was to create Java-based software code generating tool to Tieto Finland – company for DX200 Base Station Controller software product development. There is no equivalent or similar program available inside the company and its development was seen as necessary for standardizing code spelling and commenting.

The tool and its project files were stored to company's network drive. The tool was developed in Java programming language and its purpose is to create new code frames in C, TNSDL and TDL languages. Customer requirements were specified and documented for the tool in the first phase of the thesis. The most important requirements for the tool were speed and user-friendliness. In the second phase of the thesis an implementation specification document was written. The tool was tested continuously during implementation phase.

The tool and the written part of the thesis were done in Tieto Finland premises in Pori. In the written part of the thesis the basics of the GSM cellular network system and its network architecture was introduced in general. The Nokia DX200 product family and the basic infrastructure of the Base Station Controller were also introduced. Software engineering, its life cycle and all phases that are included to it were examined in general in this work. At the end of the written part development phases of the tool were described from the beginning to the deployment phase and success of both the tool and the thesis were evaluated.

ALKUSANAT

Opinnäytetyöni tein Tieto Finland Oyj:lle Porin toimipisteen tiloissa, joten haluan kiittää Tieto Finland Oyj:tä mahdollisuudesta tehdä opinnäytetyöni. Kiitän myös kaikkia työtovereitani ideoimisesta, testauksesta ja motivoivan ilmapiirin luomisesta.

Sakari Anttonen

LYHENTEET

Abis-interface	Abis-rajapinta, BSC:n ja MSC:N välinen rajapinta
A-interface	A-rajapinta, BTS:n ja BSC:n välinen rajapinta
Air-interface	Ilmarajapinta, MS:n ja BTS:n välinen rajapinta
AUC	Authentication Centre, autentikointikeskus
BCSU	BSC Signalling Unit, BSC:n signalointiyksikkö
BSC	Base Station Controller, tukiasemaohjain
BSS	Base Station Subsystem, tukiasema-alijärjestelmä
BTS	Base Transceiver Station, tukiasema
CEPT	Conférence Européenne des Postes et Télécommunications
CLS	Clock and Synchronisation Unit, kello ja synkronisointiyksikkö
DCN	Data Communications Networks, Tietoliikenne verkot
EIR	Equipment Identity Register, laitetunnisterekisteri
ET	Exchange Terminal, keskuspääte
FSC	Fixed Network Switching Centre, kiinteän verkon matkapuhelinkeskus
GGSN	Gateway GPRS Support Node, yhdyskäytäväsolmu
GMSC	Gateway Mobile services Switching Centre, porttimatkapuhelinkeskus
GPRS	General Packet Radio Service, yleinen pakettiradiopalvelu

GSM	Global System for Mobile communications, maailmanlaajuinen matkapuhelinjärjestelmä
GSWB	Group Switch, kytkentäkenttä
HLR	Home Location Register, kotirekisteri
ISDN	Integrated Service Digital Network, piirikytkentäinen puhelinverkkojärjestelmä
JRE	Java Runtime Environment
LAN	Local Area Network, lähiverkko
MB	Message Bus, sanomaväylä
MCMU	Marker and Cellular Management Unit, kytkentämatriisi- ja radioverkkoresursienhallintayksikkö
ME	Mobile Equipment, matkapuhelin laite
MML	Man-Machine Language, ohjelmointikieli
MS	Mobile Station, SIM-kortilla varustettu matkapuhelin tai muu vastaava päätelaite
MSC	Mobile services Switching Centre, matkapuhelinkeskus
NMS	Network Management Subsystem, verkonhallinta-alijärjestelmä
NSS	Network and Switching Subsystem, verkonkytkentäalijärjestelmä
O&M	Operations & Maintenance interface, BSS:n ja NMS:n välinen rajapinta
OMU	Operation and Maintenance Unit, käyttö- ja ylläpitoyksikkö

PCM	Pulse Code Modulation, pulssikoodausmodulaatio
PCU	Packet Control Unit, paketinhallintayksikkö
PSTN	Public Switched Telephone Network, yleinen puhelinverkko
SGSN	Service GPRS Support Node, palveleva GPRS-tukisolmu
SIM	Subscriber Identity Module, tilaajamoduuli
SS7	Signalling System No. 7, puhelinverkon merkinantoprotokolla
TC	Transcoder, transkooderi
TDL	Telenokia Database Language, ohjelmointikieli
TNSDL	TeleNokia Specification and Description Language, ohjelmointikieli
TRX	Transceiver, lähetinvastaanotin
VLR	Visitor Location Register, vierailijarekisteri

SISÄLLYS

1 JOHDANTO

Tätä lopputyötä lähdettiin toteuttamaan tilanteessa, jossa tarvitaan uusia ja nopeuttavia keinoja tuottamaan koodia Tieto Finland Oyj – yritykselle. Työn tarkoituksena on tuottaa ohjelma, joka luo automaattisesti ohjelmistorunkoja Nokian DX200 tukiasemaohjaimelle käyttäjän valitsemalla ohjelmointikielellä. Vastaavanlaista työkalua ei yrityksessä ole saatavilla.

Opinnäytetyö on jaettu neljään eri osioon. Ensimmäisessä osiossa (luku 2) käydään yleisesti läpi matkapuhelinverkkojen rakennetta. Toisessa osiossa (luku 3) käydään lyhyesti läpi DX200 ympäristöä, jolle työkalu luo koodirunkoja. Kolmannessa osiossa (luku 4) käydään läpi ohjelmistotuotantoa ja sen eri vaiheita, koska tutkin tätä osa-aluetta opinnäytetyötä tehdessäni. Opinnäytetyön osiossa (luku 5 ja luku 6) käydään läpi työkalun toteutusta, sen eri vaiheita ja lopputulosta. Työkalun vaiheet käydään läpi aloituksesta käyttöönottoon ja ylläpitoon saakka. Lisäksi opinnäytetyön viimeisessä osiossa (luku 7) pohditaan työkalun ja projektin onnistumista.

2 GSM MATKAPUHELINJÄRJESTELMÄ

1980-luvun alussa oli huomattu, että eri puolilla Eurooppaa oli käytössä monia keskenään yhteensopimattomia matkapuhelinjärjestelmiä. Samanaikaisesti tietoliikennepalveluiden tarve oli huomattavasti kasvanut (SYSTRA 2000, 13). Vuonna 1982 CEPT:n (Conférence Européenne des Postes et Télécommunications) perusti ryhmän, jolle annettiin tehtäväksi laatia yhteinen suositus eurooppalaiselle 900MHz alueella toimivalle puhelinjärjestelmälle. Ryhmän nimeksi tuli GSM (Groupe Spéciale Mobile) (Granlund 2001, 114). Myöhemmin GSM-lyhenne sai myös englanninkielisen vastineensa Global System for Mobile communications.

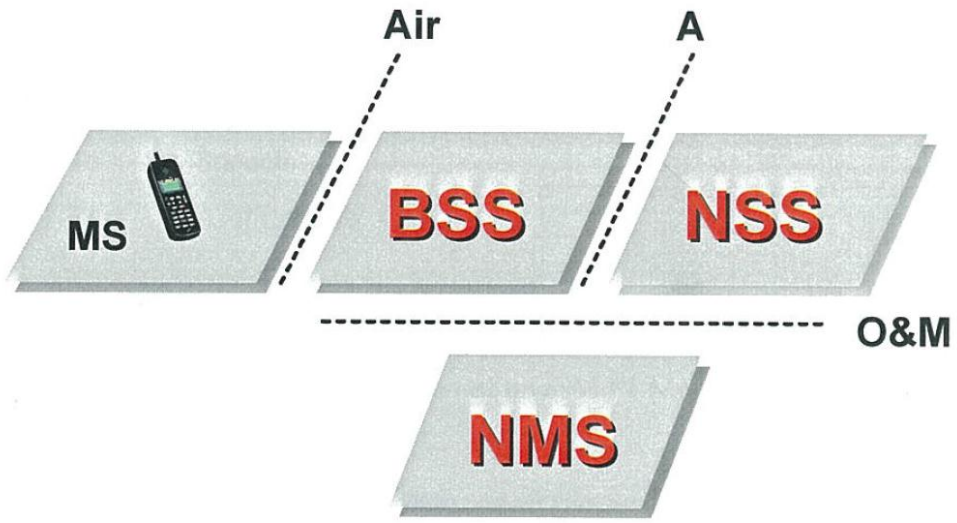
1990-luvun alussa yhteisen matkapuhelinjärjestelmän puuttuminen nähtiin maailmanlaajuisena ongelmana. Kuitenkin jo vuonna 1991 soitettiin ensimmäinen viralli-

nen GSM-puhelu ja vuonna 1992 maailman ensimmäinen GSM-verkko avattiin julkiseen käyttöön. (SYSTRA 2000, 10–11). GSM-järjestelmään kehitys ei kuitenkaan pysähtynyt, vaan siihen on ajan kuluessa liitetty lisää uusia ominaisuuksia (Granlund 2001, 114). Vuonna 1999 tehtiin ensimmäinen datapuhelu käyttäen yleistä pakettiradiopalvelua GPRS (General Packet Radio Service) ja saman vuoden lopussa GSM-käyttäjien määrä oli jo yli 250 miljoonaa 127:ssä eri maassa. (SYSTRA 2000, 13)

2.1 GSM-verkon arkkitehtuuri

GSM-verkko muodostuu neljästä kokonaisuudesta. Tilaajaa edustaa matkapuhelinlaite tai liikkuva asema MS (Mobile Station) (Granlund 2001, 120). MS on yhteydessä kolmeen alijärjestelmään, jotka on liitetty toisiinsa vaatimusmäärittelyn mukaisilla avoimilla ilma-, A- ja O&M käytönhallintarajapinnoilla (kuva 1).

Käyttökokemus analogisissa verkoissa oli osoittanut, että keskitetty älykkyys generoi liiallista kuormaa järjestelmälle ja se johti suorituskyvyn heikkenemiseen. Tästä syystä GSM-verkko on toteutettu siten, että älykkyys on jaettu kolmen alijärjestelmän kesken ja älykkyyttä pystytään jakamaan verkon läpi (SYSTRA 2000, 14 - 15). Verkkoälykkyyden kolme toteuttavaa alijärjestelmää ovat tukiasema-alijärjestelmään BSS (Base Station Subsystem), verkonhallinta-alijärjestelmään NMS (Network Management Subsystem) ja keskusalijärjestelmään NSS (Network Switching Subsystem).



Kuva 1. Yleinen matkapuhelinjärjestelmä (SYSTRA 2000, 15)

2.1.1 Matkapuhelin MS

Matkapuhelin MS (Mobile Station) koostuu matkapuhelinlaitteesta ME (Mobile Equipment) ja sen sisälle asennettavasta SIM (Subscriber Identity Module)– kortista. SIM eli tilaajatunnistemoduuli on pieni muistilaite, joka sisältää käyttäjäkohtaiset tunnistetiedot ja listan käytettävissä olevista GSM-verkoista. (SYSTRA 2000, 14, 27)

2.1.2 Tukiasema-alijärjestelmä BSS

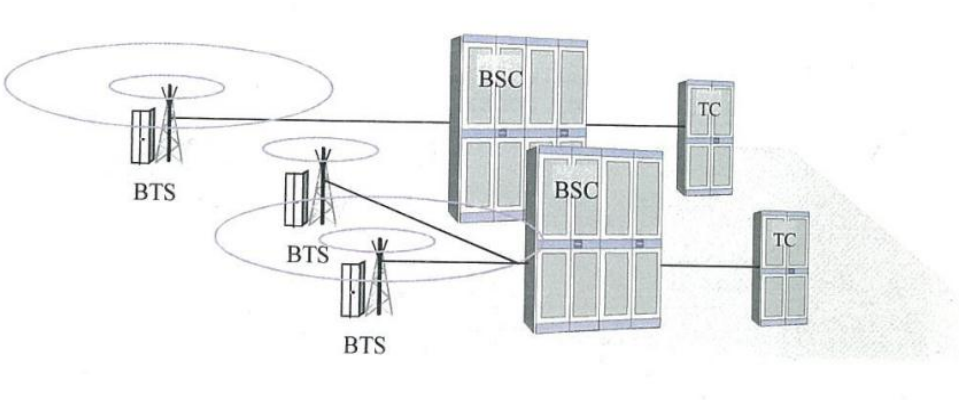
Tukiasema-alijärjestelmä BSS (Base Station Subsystem) on vastuussa GSM-verkon radio-resurssien hallinnasta ja käytöstä. BSS ohjaa, tarkkailee ja vapauttaa liikennettä sekä hallitsee yhteyksiä radiorajapinnalla (SYSTRA 2000, 94). BSS muodostuu kolmesta eri osasta:

- Tukiasemaohjaimesta BSC (Base Station Controller)
- Tukiasemasta BTS (Base Tranceiver Station)
- Transkooderista TC (TransCoder)

MS on suorassa yhteydessä BTS:ään lähetinvastaanottimen TRX (Transceiver) kautta ja BTS on suorassa yhteydessä BSC:hen. BTS on liitetty GSM-verkkoon joko kiinteällä yhteydellä tai erillisen radiolinkin avulla. BTS:n tehtäviin kuuluu digitaalisen tiedon modulointi radiotietä varten, siirtyvän tiedon koodaus tai purkaminen ja puheluiden salaaminen sekä salauksen purkaminen (Granlund 2001, 121).

Tukiasemaohjain BSC (Base Station Controller) on keskeinen elementti tukiasema-alijärjestelmässä ja sen tehtävänä on siirtoteiden ylläpito MS:ltä NSS:lle (SYSTRA 2000, 44). BSC valvoo radiosignaalin laatua, säätelee signaalien lähetystehoja ja huolehtii käyttäjän siirtymisestä solusta toiseen. (Granlund 2001, 122).

TC:n tehtävänä on muuntaa tiedonsiirtonopeuden sopiviksi langallisen ja langattoman verkon välillä. Siirtonopeuksien sovittaminen on tarpeellista, koska Abis rajapinnan kanavakohtaiset datanopeudet ovat 16kbps tai 64kbps ja ilma-rajapinnalla nopeudet ovat 9,6 – 56kbps(Granlund 2001, 122). Yleinen pakettiradiopalvelu GPRS (General Packet Radio Service) ei kuitenkaan mene TC:n läpi, sillä BSC:ssä on pakettinhallintayksikkö PCU (Packet Control Unit), jonka tehtävä on erottaa GPRS-datapaketit GSM-lähetteiden virrasta ja lähettää ne erilliseen GPRS-runkoverkkoon. (Penttinen, 2006, 128.) GPRS käydään tarkemmin läpi kappaleessa 2.1.5.



Kuva 2. Tukiasema-alijärjestelmä (SYSTRA 2000, 44)

2.1.3 Verkonkytkentäalijärjestelmä NSS

Verkonkytkentäalijärjestelmän NSS (Network Switching Subsystem) pääfunktioihin kuuluvat: puheluiden sekä liikkuvuuden hallinta, turvallisuudesta huolehtiminen ja monen tyyppisten GSM-tunnusnumeroiden ja GSM-funktioiden hallinta. Tämän lisäksi NSS:n vastuulla on myös laskutus ja signalointi BSS:än sekä muiden verkkojen kanssa (SYSTRA 2000, 93). NSS muodostuu seuraavista verkkoelementeistä:

- matkapuhelinkeskus MSC (Mobile-services Switching Centre)
- tunnistamiskeskus AUC (Authentication Centre)
- kotirekisteri HLR (Home Location Register)
- vierailijarekisteri VLR (Visitor Location Register)
- laitetunnisterekisteri EIR (Equipment Identity Register)

MSC on vastuussa puheluiden ohjauksesta GSM-verkosta seuraavalle MSC:lle ja sen tehtävä on tunnistaa puhelun tyyppi, alkuperä ja määränpää (SYSTRA 2000, 40). Puheluiden muodostuksessa GSM-verkon ja kiinteän verkon PSTN (Public Switched Telephone Network) välillä toimii porttimatkapuhelinkeskus GMSC (Gateway Mobile services Switching Centre), jonka avulla puheluita voidaan ohjata lankaverkosta GSM-verkkoon ja toisinpäin. MSC:n tehtävänä on myös huolehtia matkapuhelimen sijainnin seurannasta ja salausparametrien välityksestä. (Granlund 2001, 123)

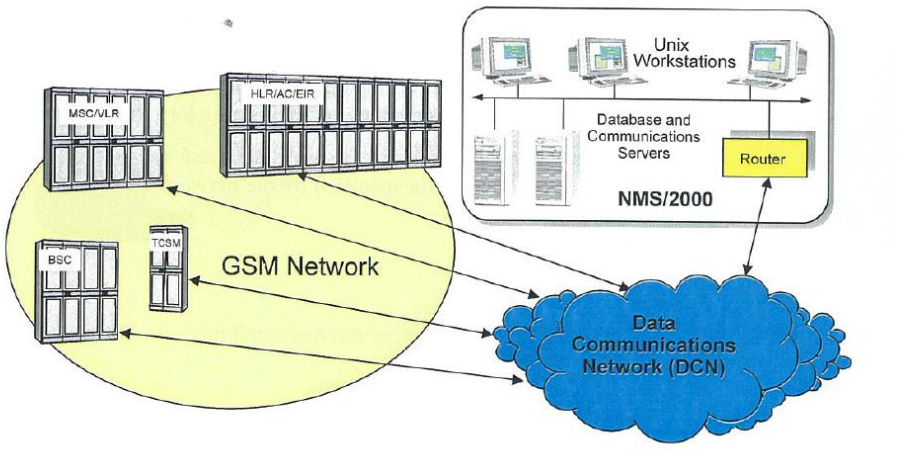
MSC:hen on liitettynä VLR tietokanta, jossa on tilapäiset tiedot alueella olevista tilaajista. VLR:ssä pidetään tiedot niin kauan kunnes käyttäjä poistuu sille määritetyltä alueelta. VLR:n lisäksi on HLR tietokanta, jossa on tieto käyttäjää tällä hetkellä palvelevasta VLR:stä. HLR:ssä on pysyvästi käyttäjän tilaajatiedot ja se päivittyy aina käyttäjää palvelevan VLR:n vaihtuessa. VLR tieto täytyy aina hakea HLR:stä puheluiden reitittämistä varten, sillä ilman sitä ei tiedetä mihin MSC:hen puhelu täytyy reitittää. AUC ja EIR ovat yleensä integroitu HLR:iin. (SYSTRA 2000, 40).

2.1.4 Verkonhallinta-alijärjestelmä NMS

Verkonhallinta-alijärjestelmä NMS (Network Management Subsystem) on kolmas GSM-verkon alijärjestelmistä. Sen tehtäviin kuuluu verkon lukuisten funktioiden ja elementtien valvonta ja se on toteutettu NMS/2000-järjestelmällä. NMS/2000-järjestelmä koostuu useista työasemista, jotka ovat yhdistettyinä tietokanta- ja tietoliikennepalvelimiin lähiverkon LAN (Local Area Network) kautta. NMS on yhdistetty GSM-verkkoon reitittimen avulla, joka muuntaa yhteyden X.25-tietoliikenneverkkoon DCN (Data Communications Network) sopivaksi. NMS:än tehtävät jaetaan kolmeen eri osa-alueeseen:

- Vikahallintaan
- Asetuksien hallitaan
- Suorituskyvyn hallitaan

NMS kattaa kaikki GSM-verkon elementit tukiasemista aina MSC:ihin ja HLR:iin asti. (SYSTRA 2000, 73–75)



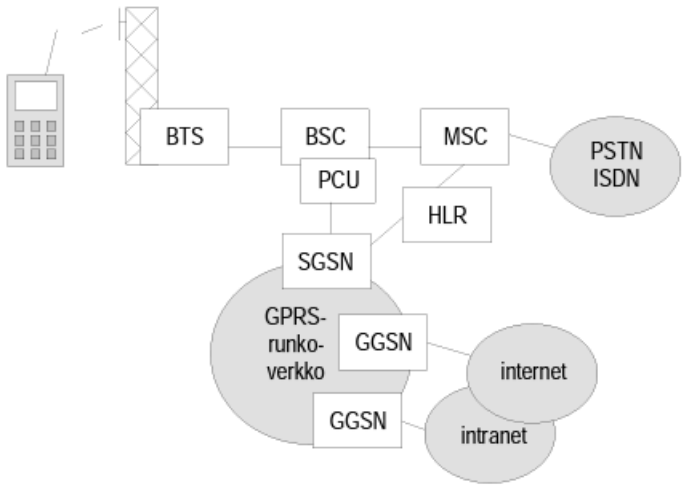
Kuva 3. Verkonhallinta-alijärjestelmä ja GSM-verkko (SYSTRA 2000, 73)

2.1.5 Yleinen pakettiradiopalvelu GPRS

Yleinen pakettiradiopalvelu GPRS (General Packet Radio Service) on GSM-verkon laajennus ja se käyttää datayhteyttä varten GSM-verkon infrastruktuuria. Datapalve-

lut ovat alusta asti olleet mukana GSM-verkossa, mutta ne ovat perustuneet piirikytkentäiseen tekniikkaan, jossa yhteys luodaan vain kerran ja istuntoa ylläpidetään niin kauan kunnes käyttäjä lopettaa istunnon. GPRS on pakettikytkentäinen datayhteys ja sen suurin etu on verkon ylimääräisen kapasiteetin hyödyntäminen sekä yhteyden jatkuvan ylläpidon mahdollisuus ilman erillistä yhteydenottoa (Granlund 2001,173). GPRS varaa vapaata fyysistä tiedonsiirtokaistaa ainoastaan dataa liikuttaessa, kun taas GSM-datayhteys varaa kokonaisen puhelun kaistan siitä huolimatta siirretäänkö dataa vai ei (Penttinen 2006, 159).

GPRS käyttää GSM-puheluiden kanssa samoja radioresursseja, mutta GPRS-yhteydet ohjataan erilliseen GPRS-runkoverkkoon BSC:n sisällä olevalla lisäelementillä, paketinohjausyksiköllä PCU (Packet Control Unit). PCU on mahdollista myös kytkeä muualle, mutta verkkosuunnittelun kannalta BSC on sille kaikista loogisin sijoituspaikka. PCU mahdollistaa kytkeytymisen GPRS-runkoverkon elementtiin palvelevaan GPRS-tukisolmuun SGSN (Serving GPRS Support Node), joka huolehtii GPRS-päätelaitteiden liikkuvuuden hallinnasta ja radiopinnan salauksesta. SGSN on yhdistetty GPRS-tukisolmun yhdyskäytävään GGSN (Gateway GPRS Support-Node), joka toimii GPRS-runkoverkkojen ja ulkopuolisten dataverkkojen välisenä reitittimenä. Tavallisesta reitittimestä GGSN poikkeaa siten, että GPRS-päätelaitteen liikkuessa GSM-verkon alueella, GGSN osaa ohjata yhteyden aikana datapaketit oikean SGSN:n alle (Penttinen 2006, 160–161).



Kuva 4. GPRS verkon peruselementit (Penttinen 2006, 161)

2.1.6 Rajapinnat

GSM-verkon määrittelyn tarkoituksena oli määrittää useita avoimia rajapintoja. Tästä johtuen verkkoa ylläpitävä matkapuhelinoperaattori voi käyttää eri laitevalmistajien tuotteita matkapuhelinverkkonsa rakentamiseen.

Nykyään GSM määrittelee kaksi täysin avointa rajapintaa. Ensimmäinen avoin rajapinta on ilma-rajapinta MS:n ja BTS:n välillä. Toinen avoin rajapinta on A- rajapinta, joka on BSC:n ja MSC:n välinen rajapinta.

Järjestelmän kuuluu enemmän kuin kaksi määriteltyä rajapintaa, mutta ne eivät ole täysin avoimia. Esimerkiksi BTS ja BSC välinen rajapinta Abis-rajapinta on ei-avoin rajapinta ja tästä johtuen eri laitevalmistajien BSC ja BTS eivät välttämättä ole yhteensopivia. (SYSTRA 2000, 14)

3 DX200 – JÄRJESTELMÄ

DX 200 on Nokian tuoteperhe, jota käytetään yleisnimityksenä erilaisille tietoliikenneverkon muodostamiseen tarkoitetuille verkkoelementeille. Ensimmäinen DX 200 -tuote oli kiinteä matkapuhelinkeskus FSC (Fixed Network Switching Centre), jonka myötä kiinteän verkon rinnalle on rakennettu lukuisia DX 200 – tuotteita, joista keskeisimmät ovat MSC, HLR ja BSC. Alustana kaikille DX 200 – tuoteperheen tuotteille toimii periaatteessa samanlainen vikasietokykyinen DX 200 – laitteisto (Silander 1999, 2, 3).

3.1 DX 200 – järjestelmän ominaisuuksia

DX 200 – järjestelmä on hajautettu, löyhästi kytketty moniprosessorijärjestelmä, joka on skaalautuva, suorituskykyinen ja vikasietoinen. DX 200 on periaatteessa yleiskäyttöinen tietokonejärjestelmä ja vasta ohjelmistolla se sidotaan palvelemaan tiettyä sovellusaluetta. Kuitenkin laitteistoa lähemmin tarkastellen voi todeta, että siinä on

laitteistokomponentteja, jotka ovat välttämättömiä puhelinkeskukselle mutta ei tavalliselle tietokoneelle (Silander 1999, 48 - 49).

DX 200 – järjestelmä on reaaliaikainen. Reaaliaikaisuus edellyttää sitä, että järjestelmä kykenee vastaamaan palvelupyyntöihin tietyssä määräajassa. Vasteajan pituus on järjestelmästä riippuvainen. Reaaliaikaisuus DX 200 – järjestelmässä on mahdollistettu keskusmuistipohjaisuudella, rinnakkaisyksiköillä sekä nopeilla tiedonsiirtoväylillä (Silander 1999, 66).

DX 200 – järjestelmän vikasietokyky perustuu laitteiston varmennukseen. Järjestelmässä on ylimääräisiä laitteistokomponentteja eli varayksikköjä, joiden tehtävänä on varmistaa laitteiston toiminta, vaikka jokin yksittäinen osa lakkaisi toimimasta. Käytettyjä varmennustapoja on kahta erilaista, 2N – varmennus ja N+1 – varmennus. 2N – varmennuksessa on tarkoin määritelty varayksikkö, joka toimii yleensä synkronisesti aktiivisen yksikön rinnalla ja se kykenee ottamaan milloin tahansa aktiiviyksikön tehtävät vastaan. Tätä kutsutaan puolenvaihdoksi ja se ei saa vaikuttaa puhelinkeskuksen toimintaan millään tavalla. N+1 – varmennuksessa voi olla käytössä yhteinen varayksikkö, joka pystyy ottamaan minkä tahansa yksikön toimintaa vastaan vikatilanteen sattuessa, tai sitten turvaudutaan johonkin muuhun jo aktiiviseen yksikköön, jolle tehtävä voidaan jakaa (Silander 1999, 50 - 51, 67 - 71).

3.2 DX 200 tukiasemaohjain

DX 200 BSC tukiasemaohjain on rakennettu samaan tapaan kuin muut DX 200 vikasietoiset tuotteet. DX 200 BSC koostuu kytkentäkentästä ja useasta tietokoneyksiköstä (kuva 5), joista jokaisella on omat tehtävänsä (SYSTRA 2000, 183 - 185).

DX 200 tukiasemaohjaimen osat:

- Kytkentäkenttä GSWB (Group Switch)
  GSWB:n päätehtävänä on kytkeä puhelut ja datayhteydet PCM-kanavien välillä.

- Keskuspääte ET (Exchange Terminal)
  ET sovittaa 2.048 Mbit/s PCM-yhteydet yhteensopiviksi MSC:n ja BTS:n välillä.

- Kello- ja synkronisointiyksikkö CLS (Clock and Synchronisation Unit)
  CLS generoi synkronisointisignaaleja BSC:n yksiköille ja se on vastuussa niiden tahdistamisesta.

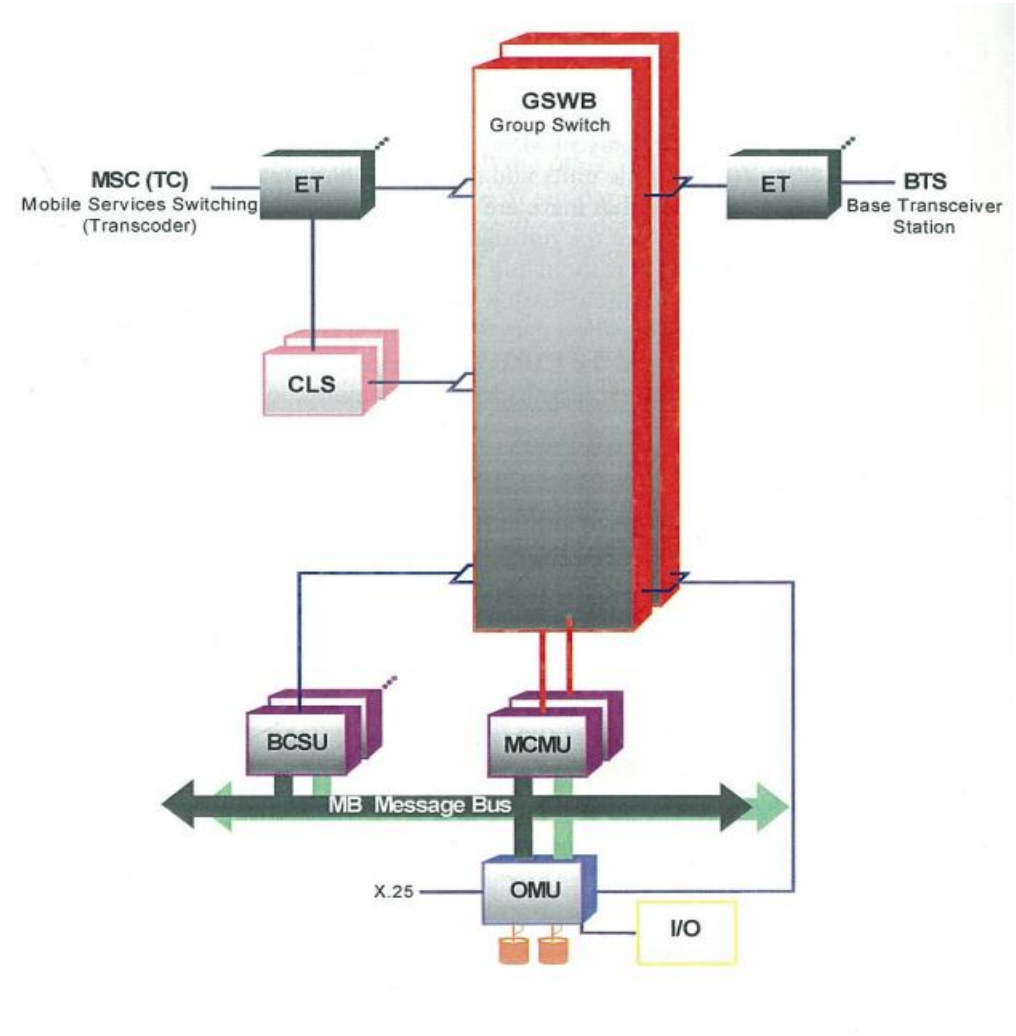
- BSC:n signalointi yksikkö BCSU (BSC Signalling Unit)
  BCSU on vastuussa SS7 (Signalling System No. 7) signaloinnista MSC:n ja BSC:n välillä sekä LAP-D (Link Access Protocol on the D-channel) signaloinnista BSC:n ja BTS:n välillä.

- Kytkentämatriisi- ja radioverkkoresurssienhallintayksikkö MCMU (Marker and Cellular Management Unit)
  MCMU:n Kytkentämatriisi ohjaa GSWB:n toimintaa ja Cellular Management Unit on vastuussa matkapuhelinverkon soluista ja radiokanavista.

- Käyttö- ja ylläpitoyksikkö OMU (Operations and Maintenance Unit)
  OMU toimii huoltopäätteenä, jonka kautta käyttäjä saa yhteyden tukiasemaan. OMU:n kautta voidaan suorittaa tarvittavat huoltotoimenpiteet ja sitä voidaan myös käyttää paikallisiin toimintoihin.

- Sanomaväylä MB (Message Bus)
  Kahdennettua sanomaväylää MB käytetään sanomien siirtämiseen ja vastaanottamiseen eri yksiköiden välillä BSC:n sisäisesti. (SYSTRA, 2000, 180–185)

- Paketinohjausyksikkö PCU (Packet Control Unit)
  PCU on BSC:n lisäelementti, jonka avulla BSC voidaan kytkeytyä GPRS-runkoverkon elementtiin nimeltään SGSN. (Penttinen 2006, 160).



Kuva 5. DX 200 BSC:n rakenne (SYSTRA 2000, 184).

4 OHJELMISTOTUOTANTO

Ohjelmistotuotantoa voidaan kuvata teknisten määrittelyiden sekä dokumenttien tuottamiseksi, joiden perusteella ohjelmisto lopulta toteutetaan. Mikäli ohjelmiston koodausvaihe voitaisiin ymmärtää dokumentaationa, ei ohjelmistotyö olisi muuta kuin dokumentointia.

Ohjelmiston tuotantoprosessi on vaiheittainen kuvaus järjestelmän vaatimuksista niiden toteuttamiseksi ja yleensä jokaisella tuotantoprosessin tasolla tuotetaan spesifikaatiodokumentti seuraavan tason syötteeksi. Kuvaamisen ylimmillä tasoilla uutta

spesifikaatiota voidaan vielä kuvata käyttäen puhekieltä, kun taas alimmilla tasoilla ohjelmiston toteutus yleensä esitetään määrittely tai spesifikaatio dokumentteina ja aivan loppujen lopuksi ohjelmointikielenä (Haikala & Märijärvi, 2002, 61 - 62).

4.1 Ohjelmistotuotannon elinkaarimallit

Tietojärjestelmän elinkaari eli ohjelmistotuotannon prosessi nähdään kokonaisvaltaisena mallintamiskohteena ja kokonaiskuvan hahmottamiseen varten on luotu erilaisia elinkaarimalleja. Elinkaarimalleissa tietojärjestelmä jaetaan useampaan vaiheeseen, joissa ensisijainen tehtävä on määritellä järjestelmän kehittämisen tehtävät, niiden ajoitus ja riippuvuudet toisistaan. Järjestelmän elinkaaren katsotaan alkavan esitutkimuksesta ja se jatkuu aina valmiin järjestelmän käyttöönottoon sekä ylläpitovaiheeseen, joka jatkuu niin kauan kunnes järjestelmästä päätetään luopua (Pohjonen, 2002, 26)

Ohjelmiston elinkaarimalleja sovellettaessa on muistettava, että mallin tarkoitus on ainoastaan antaa pohja ohjelmistotyölle. Ohjelmiston elinkaarimalleja on useita ja niistä yleensä löytyvät vähintään ohjelmistotuotannon määrittely-, suunnittelu- ja toteutusvaiheet. Laadunvarmistustoimenpiteet, esimerkiksi tarkastukset, katselmukset ja testaukset, liittyvät ohjelmistotuotannon elinkaaren kaikkiin vaiheisiin ja niillä pyritään vähentämään tai poistamaan ohjelmistossa esiintyviä vikoja jo mahdollisimman aikaisessa vaiheessa. (Haikala & Märijärvi, 2002, 37).
Yleisimmin käytössä olevat elinkaarimallit ovat:

- Vesiputousmalli (waterfall-model)
- V-malli (V-model)
- Prototyyppilähestymistapa (prototyping-model)
- Spiraalimalli (spiral-model)

4.1.1 Esitutkimus

Esitutkimus on ohjelmistotuotannon aloittamisen ensi-vaihe, jolla vastataan kysymyksiin: Onko ohjelmiston tekeminen mahdollista ja kannattaako ohjelmistoa tehdä?

Esitutkimuksen tavoitteena on selvittää yrityksen tarve suunnitteilla olevalle projektille ja miten projekti olisi mahdollista tehdä alusta loppuun. Esitutkimus tuottaa projektista päättäville henkilöille tietoa sekä lähtökohdat mahdolliselle projektin rakentamishankkeelle. Esitutkimuksen aloittaminen ei kuitenkaan tarkoita projektin automaattista käynnistämistä vaan se voi myös johtaa hankkeen hylkäämiseen tai hyllyttämiseen. Esitutkimuksesta syntyy dokumentteja ja raportteja, jotka on hyvä arkistoida myöhempää käyttöä varten, vaikka projektia ei aloitettaisikaan (Pohjonen 2002, 27).

Risto Pohjosen mukaan hyvässä esitutkimus raportissa tai dokumentissa olisi hyvä löytää seuraavat asiat: (Pohjonen, 2002, 27)

- Organisaation tietojenkäsittelyn nykytilanteen kuvaaminen siltä osin kuin se liittyy käsillä olevaan kehityshankkeeseen
- Niiden ongelmien kuvaukset, joihin järjestelmän oletetaan tuovan ratkaisut
- Kuvaukset niistä viite- ja sidosryhmistä, joita hanke koskee
- Alustavien järjestelmälle asetettavien tavoitteiden ja rajausten määritykset
- Uuden järjestelmän kehittämistavoitteiden määritykset
- Eri toimintavaihtoehtojen kuvaukset arvioineen ja perusteluineen
- Alustava suunnitelma tietojärjestelmän kehittämishankkeen läpiviemiseksi

4.1.2 Vaatimus- ja toteutusmäärittely

Vaatimusmäärittelyvaiheessa kerätään järjestelmälle asetetut asiakasvaatimukset ja niistä johdetaan vaatimusmäärittelydokumentti. Vaatimusmäärittelydokumenttiin kirjataan asiakkaan määrittelemät toiminnalliset ja ei-toiminnalliset vaatimukset. Toiminnalliset vaatimukset kertovat mitä järjestelmän oletetaan tekevän ja ei-toiminnalliset vaatimukset sanelevat tekemisen reunaehdot. Vaatimusmäärittelydokumentin tarkoitus on vastata kysymykseen mitä toteutetaan ja sen tulee olla helposti luettavissa ja ymmärrettävissä. Vaatimusmäärittelydokumenttiin kirjataan myös kaikki mahdolliset rajoitukset, esimerkiksi laitteiston muistin käytön rajoitus.

Vaatimusmäärittelyvaihe on tärkeä vaihe ohjelmistosuunnittelua. Asiakasvaatimusten kerääminen ja dokumentointi on haastavaa työtä, sillä asiakas ei aina ole täysin

tietoinen mitä hän haluaa ja miten. Tyypillisesti törmätään ongelmiin kuten vaatimusten keskeneräisyys tai ristiriitaisuus. Tästä johtuen asiakasvaatimuksia tuottaminen kirjalliseen muotoon saattaa kestää, sillä myöhemmin puutteellisten määrittelyjen korjaaminen tulee moninkertaisesti kalliimmaksi (Pohjonen, 2002, 28 - 30)

Risto Pohjosen mukaan vaatimusmäärittelydokumentista olisi hyvä löytää seuraavat asiat: (Pohjonen, 2002, 30 - 31)

- Kuvaus kehittämishankkeen toimeksiannosta
- Yleiskuvaus kohdejärjestelmä osalta organisaatiossa vallitsevasta nykytilanteesta
- Kuvaus kohdejärjestelmästä ja sille asetetuista tavoitteista pääpiirteissään
- Jokaisen toiminnallisen vaatimuksen kuvaus
- Jokaisen ei-toiminnallisen vaatimuksen kuvaus
- Jokaisen rajoitteen kuvaus
- Vaatimukset ja rajoitteet numeroituina ja priorisoituina
- Mahdolliset lisäselvitykset

Vaatimusmäärittelyt toimivat syötteenä ohjelmistotuotannon seuraavalle vaiheelle, toteutusmäärittelylle. Toteutusmäärittelyssä tavoitteena on analysoida vaatimusmäärittelyt yksityiskohtaisesti ja johtaa niistä järjestelmän toiminnallisen määrittelyn dokumentti. Toiminnallisen määrittelydokumentti on hyvä laatia selkeästi, sillä sen perusteella lähdetään tuottamaan järjestelmälle ohjelmistoa ja sen pohjalta voidaan alustavasti laatia testaus suunnitelma sekä käyttöohjeistus. (Pohjonen, 2002, 31 - 32)

Risto Pohjosen mukaan hyvästä toiminnallisesta määrittelydokumentista olisi hyvä löytyä seuraavat asiat: (Pohjonen, 2002, 31 – 32)

- Yleiskuvaus rakennettavan järjestelmän tarkoituksesta
- Kuvaus järjestelmän ympäristöstä
- Kuvaus järjestelmän toiminnasta yleisellä tasolla
- Kuvaus järjestelmän käyttäjistä
- Kuvaukset järjestelmän yleisistä rajoitteista
- Kuvaus järjestelmään ja sen käyttöön liittyvistä oletuksista ja riippuvuuksista
- Järjestelmän jokaisen toiminnon yksityiskohtainen kuvaus

- Järjestelmän käsittelemien tietojen ja tietokantojen käyttö
- Järjestelmän rajapintojen kuvaukset (käyttö-, ohjelmisto-, laitteisto- ja tietoliikenneliittymät)
- Määrittelyt järjestelmän suorituskyvyn, käytettävyyden, virhetilanteista toipumisen sekä turvallisuuden suhteen
- Kuvaukset järjestelmään tai sen toteuttamiseen liittyvistä rajoitteista (standardit, laitteisto- ja ohjelmistorajoitteet)

4.1.3 Suunnittelu

Suunnittelussa pyritään kuvaamaan määrittelyvaiheen toteutus. Suunnittelu voidaan vaiheistaa jakamalla se kahteen tai useampaan eri tasoon ja sen tarkoituksena on muuntaa toiminnallinen määrittely tekniseksi määrittelyksi, joka kuvaa järjestelmän toteutuksen. Suuremmat järjestelmät on hyvä aluksi jakaa mahdollisimman itsenäisiin, toisistaan riippumattomiin osiin, moduuleihin. Tätä vaihetta kutsutaan arkkitehtuurisuunnitteluksi. Suunnitteluvaiheessa tavoitteena on saada selville miten järjestelmä tulee suorittamaan tehtävänsä (Haikala & Märijärvi, 2002, 40).

Moduulisuunnittelussa puolestaan tavoitteena on määritellä tarkasti jokaisen pienin moduulin sisäinen rakenne, mitä moduuli sisältää. Moduuleja suunnitellessa tavoitteena on pitää moduulien koot ja toiminnot pieninä, sillä mitä suuremmaksi moduulit kasvavat sitä hankalampaa niitä on ylläpitää. Suunnittelun kannalta on hyvä asettaa tekninen määrittelyn tavoitteiksi selkeys, ymmärrettävyys, tehokkuus, luotettavuus, ylläpidettävyys ja siirrettävyys (Pohjonen, 2002, 32 – 34).

Tekninen määrittely on myös dokumentoitava vaihe. Risto Pohjosen mukaan dokumentaation tulisi kattaa ainakin seuraavat osa-alueet: (Pohjonen, 2002, 33)

- Tiivistelmä järjestelmän tarkoituksesta
- Kuvaus järjestelmän sovellusalueesta ja järjestelmän osuudesta siinä
- Kuvaus järjestelmän laitteisto- ja ohjelmistoympäristöstä
- Kuvaus järjestelmän toteutuksen keskeisistä reunaehdoista
- Kuvaus järjestelmän ja sen ympäristön välisistä vuorovaikutuksista
- Kuvaus järjestelmän ja sen ympäristön välisestä vuorovaikutuksesta

- Kuvaus järjestelmän arkkitehtuurista (yleiset ratkaisuperiaatteet, ohjelmisto- ja tietokanta-arkkitehtuuri, uudelleenkäytettävät komponentit)
- Kuvaus jokaisesta yksittäisestä järjestelmän moduulista ja alijärjestelmästä
- Toteutusrajoitteiden määrittelyt
- Mahdollisten erityisten teknisten ratkaisujen kuvaukset
- Kuvaukset mahdollisista vaihtoehtoisista tai hylätyistä ratkaisuista
- Mahdolliset muut toteutukseen vaikuttavat seikat

4.1.4 Toteutus

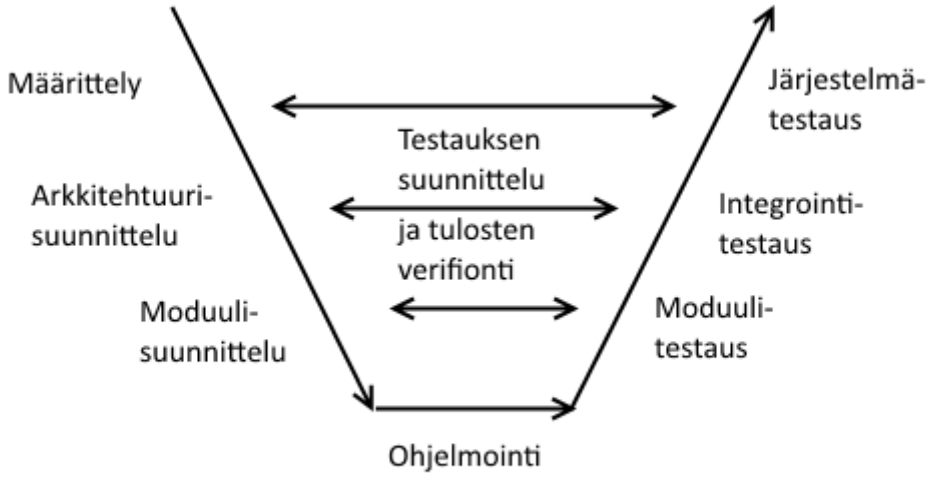
Toteutusvaiheessa toteutetaan itse ohjelma eli kirjoitetaan ohjelmakoodi. Ohjelmakoodien valmistuessa aiemmin moduuleihin pilkotut osat integroidaan eli yhdistetään toimivaksi kokonaisuudeksi. Toteutusvaiheen suoraviivaisuus riippuu hyvin paljon edellisten vaiheiden onnistumisesta sekä siitä miten ohjelmistoa toteuttaessa esiintyvät ongelmat ja mahdollisesti muuttuneet vaatimukset hoidetaan. Toteutuksen täytyy vastata sille asetettuja vaatimuksia ja sen on oltava toiminnallisen ja teknisen määrittelyn mukainen.

Toteutusvaiheessa pitää ottaa huomioon järjestelmän siirrettävyys ja ylläpidettävyys. Ohjelmistoa toteuttaessa on otettava huomioon erilaiset laite- ja käyttöympäristöt, joissa sitä voidaan käyttää. Mikäli ohjelmistosta tarvitsee tehdä laitteisto yksityiskohtaista, se kannattaa eristää omaksi moduulikseen, jolloin ohjelmaa siirrettäessä eri ympäristöön vain yksi moduuli tarvitsee kirjoittaa täysin uusiksi. Ylläpidettävyyttä parantaa standardoidut koodaus-, nimeämis- ja kommentointimenetelmät, jolloin koodista tulee helpompi lukuisempaa ja muokattavampaa. (Pohjonen, 2002, 34 – 35).

4.1.5 Testaus

Ohjelmistoa ei voida ottaa käyttöön ilman huolellista testausta. Testauksella pyritään paikallistamaan systemaattisesti kaikki mahdolliset ohjelmistovirheet. Ohjelmistotestaus jaetaan V-mallin (kuva 6.) mukaan moduuli-, integrointi- ja järjestelmätestaukseen. Moduulitestauksessa testataan yksittäisen moduulin toimivuutta mahdollisesti

simuloidussa ympäristössä, integrointitestauksessa testataan yhden tai useamman eri moduulin yhteistoimintaa ja järjestelmätestauksessa testataan koko systeemin toimintaa ja suorituskykyä. Ohjelmistotestauksen vaiheet suunnitellaan eri ohjelmistotuotannon vaiheiden yhteydessä V-mallin mukaisesti. Testauksessa on tärkeä muistaa, että testauksen pitää perustua määrittelyn sekä suunnittelun tuloksiin, eikä itse ohjelmakoodiin (Pohjonen, 2002, 35–36).



Kuva 6. Ohjelmistotestauksen V-malli

4.1.6 Käyttöönotto

Huolellisen ja kattavan testauksen jälkeen järjestelmä voidaan ottaa käyttöön. Käyttöönottoon täytyy varata aikaa, jotta se sujuisi ongelmitta. Käyttöönotossa täytyy ottaa huomioon edelliset vastaavanlaiset järjestelmät sekä niiden sisältämät tiedot ja uutta järjestelmää käyttävä henkilökunta täytyy kouluttaa (Pohjonen, 2002, 37).

4.1.7 Ylläpito

Käyttöönoton jälkeen alkaa ohjelmiston elinkaaren pisin vaihe, ylläpito. Ylläpidon tarkoituksena on huolehtia jo tuotantokehityksessä olevan ohjelmiston toimintakunnosta virheiden korjauksilla, jatkokehityksellä ja muutostoimenpiteillä. Ylläpidon

aikana tehdyistä muutoksista on yhtä tärkeätä pitää dokumentaatio kuin muistakin ohjelmistotuotannon vaiheista, sillä ylläpidon aiheuttamat toimenpiteet ovat muuten vaikeasti jäljiteltävissä. (Pohjonen, 2002, 37–38).

5 TYÖKALUN TOTEUTUS

Tässä kappaleessa kuvataan työkalun alkuvaiheet aina aloituksesta työkalun käyttöönottoon saakka.

Työn aloituspalaverissa käytiin läpi ohjelman vaadittavat ominaisuudet, jonka jälkeen hahmoteltiin työlle alustava aikataulu. Projektin aikataulutusta sovittiin, että ohjelma on viimeistään valmis heinäkuun lopussa vuonna 2014 ja aikaa työkalun tekemiselle varattiin vähintään 236 tuntia. Työkalu tulee käyttöön Tieto Finland Oyj - Porin toimipisteelle. Työkalun toteutusvaihe alkoi viikolla 19 ja työkalun luomiseen käytetyksi kieleksi valittiin Java.

5.1 Käytetyt ohjelmat

Java Runtime Environment (JRE):
Java koodia voidaan ajaa Java Runtime Environmentilla (JRE). Se sisältää Java virtuaalitietokoneen, Java alustan ydinluokat ja tuen Java alustan kirjastoihin. (Javan työkalun www-sivut 2014).

Eclipse Java development tools:
Eclipse on ohjelmointiympäristö, jota käytetään ohjelmien kehittämiseen. Eclipse on suurimmakseen osakseen kirjoitettu Java kielellä, jolloin se luonnollisesti soveltuu hyvin Java kielen kehitysympäristöksi. Eclipse työkalulla voidaan myös käyttää muiden kielien ohjelmoinnissa sen liitännäisten ohjelmien avulla (Eclipse työkalun www-sivut 2014).

Eclipse Add-on WindowBuilder:
WindowBuilder on laajennus Eclipse ohjelmointiympäristöön, jolla voidaan luoda graafisia käyttöliittymiä. Laajennus generoi käyttäjän määrittelemän graafisen käyttöliittymän mukaisen koodin (WindowBuilder projektin www-sivut 2014).

5.2 Vaatimusmäärittelyt

Työkalulle määriteltiin aloitusvaatimukset jo työn aloituspalaverissa, jota voidaan käytännössä katsoa työn esitutkinta vaiheeksi. Tärkeimmät vaatimukset työkalulle olivat helppokäyttöisyys ja nopeus. Valmiin työkalun tavoitteena on helpottaa uuden koodin luontia ja vähentää virheellisen koodin tuottamista, eikä päinvastoin. Yhtenä vaatimuksena oli myös erilaisten moduulien lisäämisen helppous, eli projektiin täytyy pystyä lisäämään ohjelmamoduuleja ilman toisten moduulien muokkaamista. Työkalun käyttöliittymälle ei vaatimusmäärittelyissä asetettu muita rajoituksia kuin helppokäyttöisyys.

Vaatimusmäärittelyvaiheen alussa rajattiin mitä työkalu voi toteuttaa, jotta projekti ei paisuisi liian suureksi. Ensimmäiseksi ja tärkeimmäksi vaatimukseksi asetettiin, että saadaan toimiva työkalu, jolla voidaan luoda valmiita koodimalleja edes yhdellä ohjelmointikielellä. Esitutkimusvaiheessa sovittiin myös, että työkalun vaatimusmäärittelyjä voidaan muokata koko projektin ajan. Vaatimusmäärittelyistä kirjoitettiin dokumentti, joka on kokonaisuudessaan liitteessä 1.

5.3 Toteutusmäärittelyt

Toteutusmäärittelyt kirjoitettiin vaatimusmäärittelydokumentin pohjalta, mutta kuitenkin normaalista ohjelmistoprojektista poiketen siten, että toteutusmäärittelyt kirjoitettiin myös toteutuksen aikana. Toteutusmäärittelydokumenttiin lisättiin asioita aina sitä mukaan, kun vaatimusmäärittelydokumenttiin lisättiin vaatimuksia. Toteutusmäärittely dokumentti on kokonaisuudessaan liitteessä 2.

5.4 Suunnittelu ja toteutus

Projektilla ei ollut varsinaista suunnitteluvaihetta, vaan sitä tehtiin suunnitellen sekä toteuttaen samanaikaisesti. Toteutuksen alussa kokeiltiin erilaisia ohjelmointitapoja, joista parhaimmaksi työhön todettiin prototyyppimenetelmä. Menetelmän mukaisesti ohjelmaa kehitettiin yksi asia kerrallaan eli uusi asia aina määriteltiin, jonka jälkeen se suunniteltiin, toteutettiin ja testattiin.

Aluksi luotiin ensimmäinen ohjelmamoduuli eli Java-paketti, jonka sisällä oli luokka, joka kykeni tekemään yksittäisen tiedoston ilman graafista käyttöliittymää. Koodaamisen ja testaamisen jälkeen kyseistä pakettia lähdettiin laajentamaan graafisella käyttöliittymällä. Aluksi käyttöliittymälle ei ollut suuria vaatimuksia vaan tavoitteena oli saada jotain näkyvää ja helposti jatkojalostettavaa. Käyttöliittymään määriteltiin jatkuvasti uusia ominaisuuksia projektin edetessä.

Aluksi ohjelmiston hierarkia muuttui huomattavan useasti, koska toteutusvaiheessa käytettiin prototyyppimenetelmää. Lopulta, kun Java luokkia ja paketteja oli luotu tarpeeksi, pystyttiin paketit sekä luokat lajittelemaan tarkasti omiin ryhmiinsä, jolloin niiden nimeäminen helpottui. Ohjelmiston hierarkian selventymisen myötä, toteutus muuttui suoraviivaisemmaksi ja uudet ominaisuudet saatiin lisättyä ilman suurempaa vaivaa. Lopulta hierarkian ylimmät paketit saatiin siihen muotoon, että niiden toimintaa ei tarvinnut enää muokata. Projektiin pystyttiin siis lisäämään paketteja ilman pelkoa toimivan koodin rikkomisesta.

5.5 Testaus

Ohjelman testaus suoritettiin toteutusvaiheen yhteydessä prototyyppimenetelmän mukaisesti. Ohjelma testattiin aina, kun uusi toiminta oli saatu lisättyä, sekä kaikkia käyttäjän toimesta johtuvia virhetilanteita pyrittiin etsimään ja estämään. Ohjelman testaamisen eduksi katsottiin yksinkertaisuus, jonka vuoksi yksittäisten luokkien testaus oli helppoa ja nopeaa.

Työkalun testaaminen tehtiin yrityksen sisäiseksi sijoittamalla aina uusin versio ohjelmasta yrityksen sisäiselle verkkolevylle, josta kaikki työntekijät pääsivät kokeilemaan ohjelmaa koko toteutusvaiheen ajan. Lopuksi ohjelmasta julkaistiin versio, jolle määrättiin ohjelmalohkokohtaisesti testaajat.

5.6 Käyttöönotto

Ohjelman käyttöönotto sovittiin alkuperäisen aikataulun mukaan viikolle 31. Projekti kuitenkin on aikataulusta kirjoittamishetkellä edellä ja tästä johtuen työkalu voidaan mahdollisesti ottaa käyttöön jo pari viikkoa aikaisemmin. Käyttöönoton helpottamista varten työkaluun oli toteutettu help näkymä, jossa on tiedot miten ohjelmaan määritetyille kielille tuotetaan ohjelmistorunkoja.

5.7 Ylläpito

Projektin toteutusvaiheessa otettiin ylläpito huomioon siten, että koodit kommentoitiin ja paketti-hierarkia tehtiin selkeäksi. Keskeisimmät asiat ylläpitoa varten kirjattiin määrittelydokumentteihin. Mikäli ohjelmaan tarvitsee tehdä muutoksia ylläpidon aikana, olisi tärkeätä kirjata tehdyt muutokset myös määrittelydokumentteihin.

Työkalun siirrettävyys huomioitiin työkalua luodessa. Tästä johtuen työkalun pitäisi olla siirrettävissä erilaisiin käyttöjärjestelmiin, joissa on toimiva Java Runtime Environment. Työkalun siirto tapahtuu käytännössä siten, että siirretään tai kopioidaan projektin ajettava jar-tiedosto käyttöjärjestelmästä toiseen. Työkalu on heti siirron jälkeen käyttövalmis.

5.8 Ongelmat

Opinnäytetyön aikana suurimmilta ongelmilta vältyttiin, mutta toteutusvaiheen aikana yhdeksi ongelmaksi koettiin käyttöliittymän luonti ja sen yhdenmukaisuus. Spesifikaatioon oli määritelty käyttöliittymän kannalta ainoastaan helppokäyttöisyys, joten työn toteuttajan vastuulle jäi käyttöliittymän asettelun mallintaminen. Kuitenkin tästä

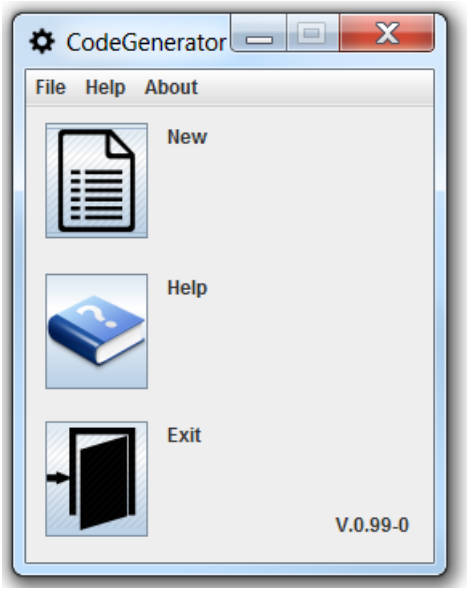
haasteesta selvittiin ja käyttöliittymä saatiin sen muotoiseksi, että ikkunasta riippumatta käyttäjä ymmärtää ikkunoiden toiminnan.

6 LOPPUTULOS

Projektin lopputuloksena saatiin työkalu koodimallien luomista varten. Tässä kappaleessa käydään läpi työkalun oleellisimpia näyttöjä ja toiminnallisuutta.

Ohjelman käynnistys ja aloitusvalikko

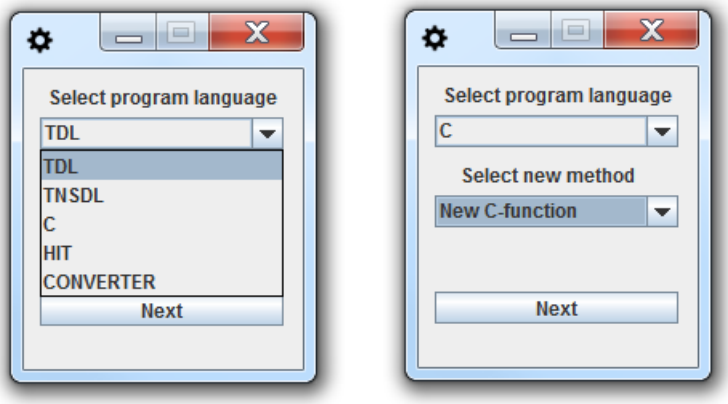
Ohjelma käynnistetään Codegenerator.jar tiedostosta, joka avaa käyttäjälle ensimmäisen näkymän eli aloitusvalikon (kuva 7). Aloitusvalikosta löytyvät uuden koodin luonti- ja help-painikkeet. Ylänavigointi paneelista About -kohdasta löytyy tietoa ohjelmasta sekä ohjelman versiohallinta.



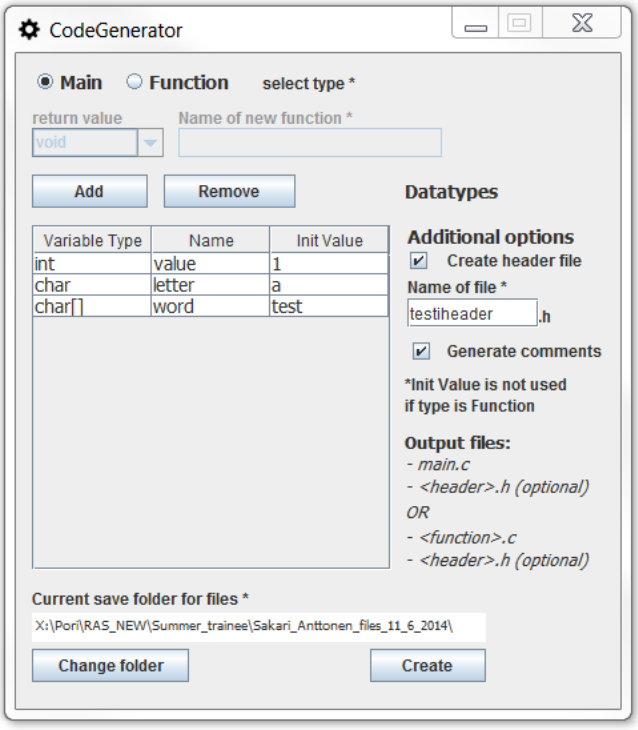
Kuva 7. Aloitusruutu

Uuden koodi-mallipojan luominen

Uuden koodi-mallipohjan luominen aloitetaan New-painiketta painamalla, joka avaa käyttäjälle valikon, jossa voidaan valita koodin kieli sekä sille luotava mallipohja (kuva 8). Next-painiketta painamalla käyttäjä pääsee määrittämään valitun kielen ohjelmistorunkoa (kuva 9).



Kuva 8. Valittava kieli tai toiminto ja sillä kielellä luotava mallipohja.



Kuva 9. Ohjelmistorungon määrittäminen

Ohjelmistorungon määrittämis-ikkunassa käyttäjä voi määritellä valitsemaansa kieleen tarvittavia komponentteja. Ikkunan määriteltävät asiat vaihtelevat kielen mukaan, esimerkiksi C-ohjelmointikielen määrittely-ikkunassa käyttäjä voi valita onko luotu tyyppi ajettava pääfunktio vai erillinen funktio. C-ohjelmistorunkoon voidaan myös määrittää muuttujat ja niiden arvot. Käyttäjä voi vielä valita, että tehdäänkö erillinen esittelytiedosto funktiokirjastoja varten, tehdäänkö automaattiset kommentit ohjelmistorunkoon ja mihin tiedostot tallennetaan levyllä. Create-painikkeella lopulta luodaan ohjelmistorunko (Kuva 10) sekä funktioesittelykirjasto (kuva 11).

```
/**************************************/
/* This file has been automatically   */
/* created by software CodeGenerator  */
/**************************************/
/* Copyright: Tieto Finland Oyj       */
/*    Author: Sakari Anttonen         */
/*   Version: 0.99-0                  */
/**************************************/
// User : anttosak
// Date : 2014/07/04 09:51:27

/**************************************
* This is the main class, all runnable *
***** contents shall be added here *****
**************************************/
#include <testiheader.h>
int main()
{
/******* Variable definitions **********/
  int value = 1;
  char letter = 'a';
  char[] word = "test";


/******** insert your code here ********/


return(0);
}
/**************************************
************* End of Main **************
**************************************/
```

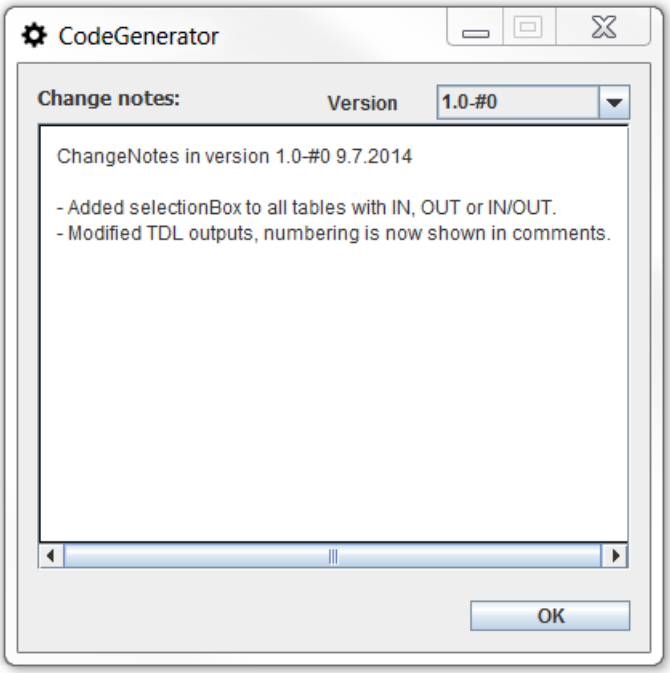
Kuva 10. Ohjelmalla luotu ohjelmistorunkotiedosto.

```
#ifndef TESTIHEADER_H_   /* Redefinition protection */
#define TESTIHEADER_H_

#include <assert.h>      /* Contains the assert macro, used to assist with detecting logical
                            errors and other types of bug in debugging versions of a program. */
#include <complex.h>     /* A set of functions for manipulating complex numbers. */
#include <ctype.h>       /* Defines set of functions used to classify characters by their types or to convert
                            between upper and lower case in a way that is independent
                            of the used character set (typically ASCII or one of its extensions,
                            although implementations utilizing EBCDIC are also known). */
#include <errno.h>       /* For testing error codes reported by library functions. */
#include <fenv.h>        /* Defines a set of functions for controlling floating-point environment. */
#include <float.h>       /* Defines macro constants specifying the implementation-specific
                            properties of the floating-point library. */
#include <inttypes.h>    /* Defines exact width integer types. */
#include <iso646.h>      /* Defines several macros that implement alternative ways to express several standard tokens.
                            For programming in ISO 646 variant character sets. */
#include <limits.h>      /* Defines macro constants specifying the implementation-specific properties of the integer types. */
#include <locale.h>      /* Defines localization functions. */
#include <math.h>        /* Defines common mathematical functions. */
#include <setjmp.h>      /* Declares the macros setjmp and longjmp, which are used for non-local exits. */
#include <signal.h>      /* Defines signal handling functions. */
#include <stdalign.h>    /* For querying and specifying the alignment of objects. */
#include <stdarg.h>      /* For accessing a varying number of arguments passed to functions. */
#include <stdatomic.h>   /* For atomic operations on data shared between threads. */
#include <stdbool.h>     /* Defines a boolean data type. */
#include <stddef.h>      /* Defines several useful types and macros. */
#include <stdint.h>      /* Defines exact width integer types. */
#include <stdio.h>       /* Defines core input and output functions */
#include <stdlib.h>      /* Defines numeric conversion functions, pseudo-random numbers generation functions,
                            memory allocation, process control functions */
#include <stdnoreturn.h> /* For specifying non-returning functions. */
#include <string.h>      /* Defines string handling functions. */
#include <tgmath.h>      /* Defines type-generic mathematical functions. */
#include <threads.h>     /* Defines functions for managing multiple Threads as well as mutexes and condition variables. */
#include <time.h>        /* Defines date and time handling functions */
#include <uchar.h>       /* Types and functions for manipulating Unicode characters. */
#include <wchar.h>       /* Defines wide string handling functions. */
#include <wctype.h>      /* Defines set of functions used to classify wide characters by their
                            types or to convert between upper and lower case */

#endif                   /* End of redefinition protection */
```

Kuva 11. Ohjelmalla luotu funktioiden esittelytiedosto.

Versiohallinta

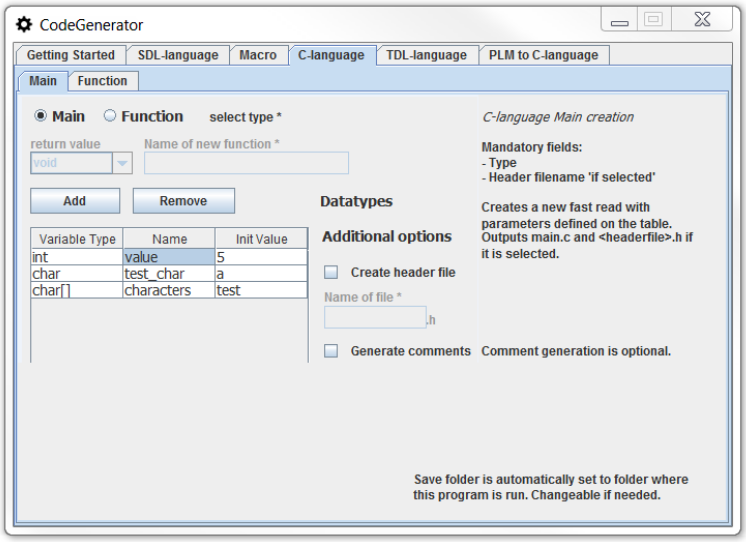
Ohjelmaan on luotu versiohallinta muutoksia varten. Versiohallinnassa pidetään tiedot kaikista sen eri versioista ja niihin tehdyistä muutoksista (kuva 12). Versiohallintaan pääsee päävalikon ylänavigointi paneelista löytyvästä About – Change notes painikkeesta.



Kuva 12. Versiohallinta

Help-valikko

Ohjelmassa on sisäinen help-valikko, jonka tarkoituksena on opastaa käyttäjää ohjelman nopeaan käyttöön. Help-valikkoon on kirjoitettu tärkeimmät tiedot jokaisesta toiminnosta (Kuva 13). Välilehtiin on liitetty käyttäjää opastavaksi esimerkeiksi kuvia.



Kuva 13. Help-valikko

7 YHTEENVETO

Tämän insinöörityön tavoitteena oli luoda työkalu koodirunkojen automaattista generointia varten. Työkalu rakentaminen alusta loppuun tehtiin projektina ja työkalun luominen onnistui hyvin, sillä se täytti kaikki sille esitutkimuksen aikana asetetut alkuvaatimukset. Työkalu onnistuttiin myös toteuttamaan siten, että sitä on mahdollista vielä kehittää erilaisiin projektitarpeisiin.

Projektissa kehitettyä työkalua lähdettiin kehittämään ohjelmistotuotannon yleisiä ohjeita hieman mukaillen, kuten työssä on jo aiemmin tullut ilmi. Ensimmäisenä vaiheena työlle oli esitutkimus, jonka jälkeen työkalua luomista alettiin tutkia ja asetettuja vaatimuksia dokumentoimaan. Vaatimusmäärittelyt kirjoitettiin suunnittelu ja toteutusvaiheessa kirjalliseen muotoon sitä mukaan kun ohjelma alkoi saada omaa graafista olomuotoansa.

Projektissa onnistuttiin hyvin sille määriteltyjen kielien, helppokäyttöisen käyttöliittymän ja aikataulun osalta. Työtä saatiin vielä laajennettua hieman lisäämällä siihen makrovalikkojen luominen ja testimielessä PL/M386-ohjelmointikielestä C- kieleksi kääntävä työkalu. Kuitenkaan kaikkia ohjelmointikieliä ei työhön voinut tai kannattanut lisätä. Työhön mietittiin muun muassa MML (Man-Machine Language) koodirunkojen tekemistä, mutta asiaa tutkittaessa todettiin kyseisen kielen koodirakenteiden olevan liian lyhyitä automaattisesti luotaviksi. Tämä koettiin hieman ongelmalliseksi, sillä työtä aloittaessa uskottiin, että työkaluun voisi liittää suurimman osan käytettävistä ohjelmointikielistä.

Opinnäytetyötä voidaan pitää moneltakin osin tekijäänsä kasvattavana kokemuksena. Työkalun luomisen aikana kokemusta karttui ohjelmistotuotannon eri vaiheista, Java-ohjelmoinnista ja käyttöliittymäsuunnittelusta. Opinnäytetyön aikana tutuksi tuli dokumentoinnin tärkeys, useat erilaiset internet-tietolähteet ja erilaiset ohjelmointiin käytettävät työkalut. Uutena haasteena tuli myös projektitiedostojen järjestys ja ylläpito. Myös alan kirjallisuus tuli tutuksi opinnäytetyötä tehdessä. Tärkeimmässä asemassa kuitenkin olivat työn aikana tehdyt vaatimus- ja toteutusmäärittelyt, joiden pohjalta tämäkin ohjelmisto saatiin valmiiksi.

LÄHTEET

Eclipsen www-sivut. 2014. Viitattu 24.6.2014. http://eclipse.org/org/

Granlund K. 2001, Langaton tiedonsiirto, Porvoo: Docendo Finland Oy.

Haikala I. Märijärvi J. 2002. Ohjelmistotuotanto. Helsinki: Talentum.

Javan www-sivut. 2014. Viitattu 24.6.2014.
http://www.java.com/en/download/faq/whatis_java.xml

Nokia Networks Oy.2000, SYSTRA GSM System Training.

Penttinen, J. 2006. Tietoliikenneverkot – Perusverkot ja GSM. Sanoma Pro Oy.

Pohjonen R. 2002. Tietojärjestelmien kehittäminen. 2. Painos. Jyväskylä: Docendo Finland Oy

Silander, S. 1999. DX-AAPINEN Johdatus DX 200–ohjelmistotyöhön. Helsinki: Nokia Telecommunications Oy.

WindowBuilder laajennuksen www-sivut. 2014. Viitattu 24.6.2014
https://projects.eclipse.org/projects/tools.windowbuilder

LIITE 1

Opinnäytetyön aikana laadittu vaatimusmäärittelydokumentti.

Koodigeneraattori tukiasemaohjaimen ohjelmistolle

Sakari Anttonen
11.6.2014
Koodigeneraattorin vaatimusmäärittelyt.doc

SISÄLLYS

VERSIOHISTORIA

Henkilö	Päiväys	Versio	Kommentit
SA	11.6	0.1	Dokumentin runko
SA	18.6	0.2	Dokumentin päivitys
SA	4.7	0.3	Dokumentin päivitys
SA	21.7	1.0	Dokumentti on katselmoitu ja korjaukset hyväksytty

1 JOHDANTO

Tämän vaatimusmäärittelydokumentin tarkoituksena on kuvata koodigeneraattori tukiasemaohjaimen ohjelmistolle – projektin vaatimuksia. Vaatimuksiin kuuluu niin toiminnalliset kuin ei toiminnalliset vaatimuksetkin.

2 YLEISKUVAUS

2.1 Kenelle?

Työkalu tehdään Tieto Finland Oyj:lle – Porin ryhmälle.

2.2 Käyttäjät

Työkalun käyttäjiä ovat BSC-ohjelmistoprojektin kehittäjät.

2.3 Käyttötarkoitus

Työkalua voidaan tarvittaessa käyttää arkipäivän ohjelmoinnin apuvälineenä.

2.4 Toimintaympäristö

Työkalun tulee toimia kaikissa Java-ohjelmia tukevissa ympäristöissä käyttöjärjestelmästä riippumatta.

3 TOIMINNALLISET VAATIMUKSET

3.1 Yleiskuvaus työkalun toiminnasta

Työkalun on tarkoitus helpottaa arkista koodaamista ja vähentää kopioinnista aiheutuvia virheitä. Työkalun tavoitteena on myös standardoida koodin kommentointia.

3.2 Toiminnalliset vaatimukset

Työkalussa on oltava vaihtoehtoina vähintään C, TNSDL (TeleNokia Specification and Description Language) ja TDL (TeleNokia Database Language) kielet ja sen on kyettävä tuottamaan niille toimivia ohjelmistorunkoja nimen, numeron sekä muutaman oleellisen parametrin kanssa. Käyttäjän on pystyttävä määrittelemään mihin tiedostoon kyseiset ohjelmarungot tehdään.

3.3 Kieli

Sovelluksen kielenä tulee olla englanti.

4 EI-TOIMINNALLISET VAATIMUKSET

4.1 Java-versio

Työkalu testataan Java Runtime Environment versiolla 1.7.0_45.

4.2 Toimintavarmuus

Työkalu ei saa kaatua käsittelemättömiin virhetilanteisiin. Työkalu ei saa kaatua myöskään käyttäjän aiheuttamiin virhetilanteisiin.

4.3 Laajennettavuus

Työkaluun tulee olla mahdollista lisätä uusia ominaisuuksia ja toiminnallisuuksia käyttöönoton jälkeenkin. Koodi tulee pitää selkeänä ja asianmukaisena sekä hyvin kommentoituna.

4.4 ”Helpit”

Ohjelmassa pitää olla asianmukainen ohjeistus tiettyjen kenttien vieressä, sekä opastava ”Help” – osio. Help – osiossa täytyy olla jokaisen koodin luomiseen erillinen ohjeistus.

5 MUUT VAATIMUKSET

5.1 Käyttöliittymä

Graafinen käyttöliittymä toteutetaan javax.swing GUI kirjastolla. Graafisen käyttöliittymän tulee olla yhdenmukainen, helppokäytettävä ja sen pitää noudattaa käyttöliittymäsuunnittelun periaatteita.

6 DOKUMENTIN MUUTOKSET

6.1 Vaatimusten muuttaminen

Vaatimuksia voidaan muuttaa koko toteutusvaiheen ajan: lisätä, poistaa sekä muokata, mikäli ne ovat oleellisia ohjelman kokonaisuuden kannalta.

LIITE 2

Opinnäytetyön aikana laadittu toteutusmäärittelydokumentti.

Koodigeneraattori tukiasemaohjaimen ohjelmistolle

Sakari Anttonen
11.6.2014
Koodigeneraattorin toteutusmäärittelyt.doc

SISÄLLYS

VERSIOHISTORIA

Henkilö	Päiväys	Versio	Kommentit
SA	11.6	0.1	Dokumentin runko
SA	18.6	0.2	Dokumentin päivitys
SA	1.7	0.3	Dokumentin päivitys
SA	4.7	0.4	Dokumentin päivitys
SA	21.7	1.0	Dokumentti on katselmoitu ja korjaukset hyväksytty

1 JOHDANTO

Tämän toteutusmäärittelydokumentin tarkoitus on määrittää vaatimukset toteutettaviksi. Vaatimukset on kuvattu projektin vaatimusmäärittelydokumentissa. Tässä dokumentissa pyritään kuvaamaan mahdollisimman tarkasti kaikki projektin toteutettavat asiat, jotta työkalun toteutus olisi dokumentin pohjalta sujuvaa ja määrittelyihin perustuvaa. Dokumentti on laadittu myös jatkokehitys huomioon ottaen.

2 TYÖKALU

2.1 Yleiskuvaus

Yleiskuvaus on määritelty vaatimusmäärittelydokumentissa kohdassa 2.

2.2 Kehitysympäristö

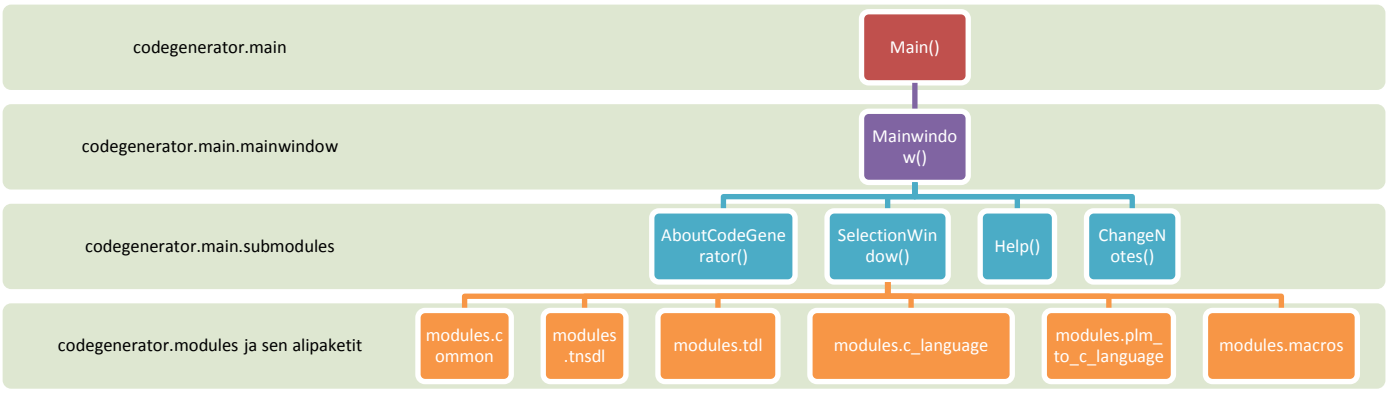
Käyttöjärjestelmä: Windows 7 Enterprise SP1

Java:	Java(TM) SE Runtime Environment (build 1.7.0_45-b18)
Toteutustyökalut:	Eclipse Java EE IDE for Web Developers
	WindowBuilder Pro v.1.5.0 Eclipse Add-on

3 TEKNISET MÄÄRITTELYT

3.1 Työkalun paketti-hierarkia

Alapuolella olevassa kuvassa kuvataan pakettien nimet sekä niiden sisältämät luokat. Pakettien sekä luokkien tiedot selitetään seuraavissa kappaleissa.



3.2 Pakettien määritykset sekä luokat

Java-paketteja käytetään luokkien järjestämiseen omiksi moduuleikseen. Toisin sanoen voidaan sanoa, että yksi paketti on aina oma moduulinsa. Paketit on järjestelty sen mukaan, mitä luokkia ne sisältävät. Samantyyppiset funktiot sekä luokat liitetään yleensä samaan pakettiin. Seuraavissa alakappaleissa käydään paketit lyhyesti läpi.

3.2.1 codegenerator.main

Paketti sisältää seuraavat luokat:

- **Main()**, jonka tehtävänä on luoda uusi ajettava ohjelma (New Runnable), käynnistää **MainWindow()** luokan.

3.2.2 codegenerator.main.mainwindow

Paketti sisältää seuraavat luokat:

- **MainWindow()**, joka luo graafisen käyttöliittymän käyttäjälle (Graphic User Interface). Käyttöliittymästä löytyy valikot ja painikkeet, joista käyttäjä voi ajaa **codegenerator.main.submodules**-paketin sisältämiä luokkia.

3.2.3 codegenerator.main.submodules

Paketti sisältää seuraavat luokat:

- **AboutCodeGenerator()**, avaa ikkunan, jossa on yleistä tietoa ohjelmasta.
- **ChangeNotes()**, avaa ikkunan, johon on kirjattu viimeisimmät muutokset mitä ohjelmaan on tehty.
- **Help()**, avaa ikkunan, jossa on tietoa miten ohjelmaa tulisi käyttää.
- **SelectionWindow()**, avaa ikkunan, josta käyttäjä voi valita minkä alamoduulin haluaa ajaa. Tällä luokalla päästään käsiksi **codegenerator.modules** sisältämiin luokkiin.

3.2.4 codegenerator.modules

Paketti sisältää seuraavat alipaketit:

- **codegenerator.modules.c_language**
- **codegenerator.modules.common**
- **codegenerator.modules.tdl**
- **codegenerator.modules.tnsdl**
- **codegenerator.modules.plm_to_c_language**

- **codegenerator.modules.macros**

Tällä tasolla yksittäiset ohjelmatoteutukset jaetaan erillisiin paketteihin, jotta pystytään toteuttamaan uutta koodia sotkematta vanhaa. Näin ohjelmaan pystytään lisäämään uutta koodia sitä mukaan, kun sille tarvetta tulee. Jokainen paketti, joka on tehty **codegenerator.modules** paketin alapaketiksi, tuottaa ohjelmistokoodia paketin nimen mukaisella kielellä. Ainoa poikkeus on **codegenerator.modules.common** -paketti, josta löytyy yleiskäyttöiset funktiot kaikille alipaketeille.

3.3 Tarkistukset

Käyttäjän tehdessä uutta koodia, ohjelma ei saa kaatua odottamattomiin tai käyttäjästä johtuvaan virheeseen, esimerkiksi siihen, ettei tiedoston nimeä ole kirjoitettu. Ohjelman täytyy varmistaa, että kaikissa kentissä mitkä johtavat käsittelemättömiin virhetilanteisiin, on sellaista dataa, että ohjelma ei kaadu. Jos käytetään sellaista funktioita, joka johtaa mahdollisesti virhetilanteeseen, on käytettävä try – catch menetelmää virhetilanteen estämiseksi.

3.4 Yleiset funktiot

Tähän alalukuun on kuvattu työkaluun toteutettavat yleiset luokat jotka sisältävät käytettävät funktiot. Yleiset luokat ja niiden funktiot löytyvät paketista **codegenerator.modules.common**

3.4.1 Error_codes()

Luokka	Error_codes()
INPUT(s)	-
OUTPUT(s)	-
Käyttötarkoitus	Käytetään muiden luokkien ja niiden funktioiden yhteydessä. Jos funktio palauttaa integer arvon, sitä käytetään tarkistukseen Error_codes() luokan arvojen avulla.

Virhekoodi	-

3.4.2 FetchTableContents()

Luokka	FetchTableContents()
INPUT(s)	JTable
OUTPUT(s)	StringBuffer
Käyttötarkoitus	Sisältää monta funktiota taulukon manipuloimista datan varten. Palauttaa taulukon datan haluttuna merkkijonona takaisin käyttäjälle.
Virhekoodi	-

3.4.3 FileMoveTool()

Luokka	FileMoveTool()
INPUT(s)	String Filesource, String Filedestination, String Filename
OUTPUT(s)	Int Error_code
Käyttötarkoitus	Käytetään tiedostojen siirtämiseen. Käyttäjä valitsee polun, johon tiedostot siirretään, jonka perusteella FileMoveTool() – luokka osaa siirtää tiedoston.
Virhekoodi	Palauttaa virhekoodin (2), mikäli tiedostoa ei voida siirtää, jonka avulla tulevat operaatiot keskeytetään. Tieto ilmoitetaan käyttäjälle MessageBox() – luokan avulla.

3.4.4 MessageBox()

Luokka	MessageBox()
INPUT(s)	String Infomessage, String location
OUTPUT(s)	-
Käyttötarkoitus	Käytetään virhetilanteiden ilmoittamiseen. Parametreina sisään tulee virhekoodin teksti, sekä sijainti missä virhe tapahtui.
Virhekoodi	-

3.4.5 PrintDataToFile()

Luokka	PrintDataToFile()
INPUT(s)	String filename, String data
OUTPUT(s)	Int Error_code

Käyttötarkoitus	Käytetään tiedostojen kirjoittamiseen. Käyttäjän kirjoittama data annetaan syötteenä tälle luokalle, joka kirjoittaa sen annetun nimiseksi tiedostoksi.
Virhekoodi	Palauttaa virhekoodin (1), mikäli tiedostoa ei voida kirjoittaa, jonka avulla tulevat operaatiot keskeytetään. Tieto ilmoitetaan käyttäjälle MessageBox() – luokan avulla.

3.5 Aikaleimat ja käyttäjät

Ohjelma kirjoittaa automaattisesti luotuihin tiedostoihin käyttäjänimen ja aikaleiman.

3.6 Helpit

Työkaluun tehdään valikkojen yhteyteen helpit kirjoittamalla viereen mitä parametreja mahdollisesti tarvitaan. Työkaluun tulee myös Help valikko, jossa on malliesimerkki siitä miten ohjelmaa tulisi käyttää.

Help – tekstien lisäksi työkalussa ilmoitetaan pakolliset kentät *merkillä. Toteutustapa voi olla esimerkiksi:

- Select type *
- Name of file*
- Create header file [ ]
    - Name of header file*
- Generate comments[ ]

4 KÄYTTÖLIITTYMÄ

4.1 Yleisrakenne

Käyttöliittymä koostuu useasta eri palasesta, jotka avaavat uusia ikkunoita. Kuitenkin jos MainWindow() - ikkuna sammutetaan, koko ohjelma sammuu sen mukana.

MainWindow() - ikkunassa on ylävalikot sekä painikkeet luontia, ohjeita ja sulkemista varten. Ylävalikossa on tietoa ohjelmasta sekä viimeiset päivitystiedot.

MainWindow() – ikkuna ei sulkeudu, kun käyttäjä avaa toisen ikkunan. Muut ikkuna sulkeutuvat, ilman että koko ohjelma sammuu.

4.2 Layout:n koko

Työkalun layout on kiinteän kokoinen ja se ei ole muokattavissa. Ikkunakoot sekä objektien koot määrätään tarkasti ja ne ovat kiinteitä. Jos esimerkiksi taulukko kasvaa liian isoksi, se tehdään automaattisesti vieritettäväksi.

4.3 Tapahtumakuuntelijat

Kaikkiin työkalun painikkeisiin laitetaan tapahtumakuuntelija. Työkalussa ei ole painikkeita, joille ei ole toimintoa. Tapahtumakuuntelijoiden avulla työkalua ohjataan haluttuihin luokkiin.

5 JATKOKEHITYS

Tähän osaan on koottu ajatukset työkalun mahdollisen jatkokehityksen kannalta.

- MML (Man-Machine Language) - Jätettiin toteuttamatta siitä syystä, että pienten, noin 10 rivin, koodin luontia ei nähty tarpeellisena. Ongelmaksi osoittautui myös se, että koodin runko ei ollut samanlainen eli toistuvuutta ei ollut riittävästi.