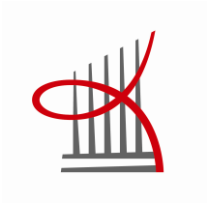


GSM-verkon tukiasemaohjaimen pake-tinohjausyksikön integraatiotestauksen automaatio

Markus Juopperi

Opinnäytetyö
Joulukuu 2014
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

JUOPPERI, MARKUS:
GSM-verkon tukiasemaohjaimen paketinohjausyksikön integraatiotestauksen automaatio

Opinnäytetyö 36 sivua
Joulukuu 2014

Tässä opinnäytetyössä käsitellään Sub System Integration(SSI) – testausjärjestelmää, jonka toteutin työskennellessäni Nokia Oyj:lla. Opinnäytetyössä käydään myös läpi toisen sukupolven matkapuhelinverkon keskeisimmät periaatteet ja tekniikat keskittyen pakettiliikenteeseen eli GPRS ja EDGE – tekniikoihin.

Erityisen tarkasti kerrotaan testauksen alla olevasta verkkoelementistä eli tukiasemaohjaimesta(BSC) ja varsinkin tukiasemaohjaimessa sijaitsevasta paketinohjausyksiköstä(PCU). Kerrotaan eri PCU – korttivarianttien ominaisuuksista ja eroista, sekä käydään yleisesti läpi PCU:n tehtävää GSM-verkossa.

Kerrotaan myös yleisesti ohjelmistotestauksen eri vaiheista ja periaatteista sekä siitä miten niitä sovelletaan tukiasemaohjaimen paketinhallintayksikön testauksessa. Työn pääasiallisena tarkoituksena on kuitenkin antaa selkeä kuva siitä miten SSI - testausjärjestelmä ja testausympäristöt on toteutettu. Kerrotaan miksi tiettyihin ratkaisuihin on päädytty, mitä ongelmia järjestelmän kehityksessä kohdattiin ja miten järjestelmää tulevaisuudessa voidaan kehittää.

Asiasanat: GSM, GPRS, EDGE, BSC, PCU, tukiasemaohjain, paketinohjausyksikkö

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in Information Technology
Option of Software Technology

MARKUS JUOPPERI:
Test Automation of GSM Network Base Station Controller's Packet Control Unit Integration

Bachelor's thesis 36 pages
December 2014

This thesis will explain the development of the packet control unit integration testing system. In 2G theory section basic fundamentals and techniques of the GSM – network will be explained. Theory section focuses especially in packet switched data transfer protocols: GPRS and EDGE.

Different phases of software testing are explained in theory level, but also told how these phases are implemented in Packet Control Unit software testing. Continuous Integration practice is handled, and explained how that practice is implemented in Packet Control Unit software development.

Tools and techniques used in the integration test automation are told in practical part of the thesis. Main tool is Holistic Integration Testing(HIT). HIT is both Integrated Development Environment and programming language. The Architecture of the Integration test automation framework is described and the usage of the framework is illustrated by handling one testcase in function level.

Key words: Base Station Controller, BSC, Packet Control Unit, PCU, GPRS, EDGE, GSM

SISÄLLYS

LYHENTEET JA TERMIT

Abis	Rajapinta tukiaseman ja tukiasemaohjaimen välillä
API	Application Programming Interface
BCSU	Base Control Signalling Unit
BCF	Base Station Control Function
BSC	Base Station Controller, tukiasemaohjain
BSS	Base Station Subsystem, tukiasemajärjestelmä
BTS	Base Station, Tukiasema
CI	Continuous Integration, jatkuva integraatio
CVOPS	C-bases Virtual Operating System
DSP	Digital Signal Processor
EDGE	Enhanched Data rates for GSM Evolution
ETP	Exchange Terminal Packet
FTP	File Transfer protocol
Gb	Rajapinta tukiasemaohjaimen ja SGSN:n välillä
GEMU	GPRS - emulaattori
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HIT	Holistic Integration Tester
IDE	Integrated Development Environment, ohjelmointiympäristö
IP	Internet Protocol
LA	Legacy Abis
LLC	Logical Link Channel
mcBSC	Multi Controller Base Station Controller
MS	Mobile Station, matkapuhelin
MSC	Message Sequence Chart
MCMU	Marker and Cellular Management Unit
OMU	Operation and Maintenance Unit
OFL	Offline logger
PA	Packet Abis
PDE	Public Definition Environment, Sack, Viestisäkki
PCI	Peripheral Component Interconnect

PCU	Packet Control Unit, Paketinohjausyksikkö
RLC	Radio Link Channel, radiolinkki kanava
SGSN	Serving GPRS Support Node
SSH	Secure Shell, tietoliikenne protokolla
SSI	SubSystem Integration
SVN	Apache Subversion, versionhallinta
TFTP	Trivial File Transfer Protocol, yksinkertainen tiedostosiirto-protokolla
XML	Extensible Markup Language, tiedon rakenteellinen kuvaus-kieli
XSL	Extensible Stylesheet Language,

1 JOHDANTO

Tämän työn tarkoituksena on kertoa kuinka paketinohjausyksikön integraatiotestauksen automaatio on toteutettu. Opinnäytetyön alussa kerrotaan teoriaan GSM – verkon toiminnasta ja verkkoelementeistä. Opinnäytetyö on kuitenkin rajattu koskemaan vain pakettikytkentäistä datansiirtoa käsitteleväksi, koska paketinohjausyksikkö käsittelee ainoastaan pakettidatakanavia. Verkkoelementeistä tarkemmin käsitellään tukiasemaohjainta, koska paketinohjausyksikkö on nimenomaan tukiasemaohjaimessa oleva erillinen yksikkö.

Työssä kerrotaan ohjelmistotestauksen teoriaa ja eri vaiheita, jotta lukijalle muodostuu kokonaiskuva paketinohjausyksikön testausprosessista, jonka yhtenä osana integraatiotestaus on. Kerrotaan myös kuinka testausta on automatisoitu ja millainen jatkuva integraatiojärjestelmä on rakennettu paketinohjausyksikön ohjelmistotestausta varten.

Käytännön osuudessa kerrotaan integraatiotestauksessa käytetyistä työkaluista ja menetelmistä sekä kerrotaan millainen testausympäristö on rakennettu. Suurin osa näistä työkaluista on kehitetty nimenomaan Nokian verkkotuotteiden testausta varten, eikä niitä ole saatavilla yrityksen ulkopuolelta. Keskeisimpänä työkaluna käytetään Holistic Integration Tester(HIT) – ohjelmointiympäristöä ja siihen kehitettyä HIT – ohjelmointikieltä.

2 2G

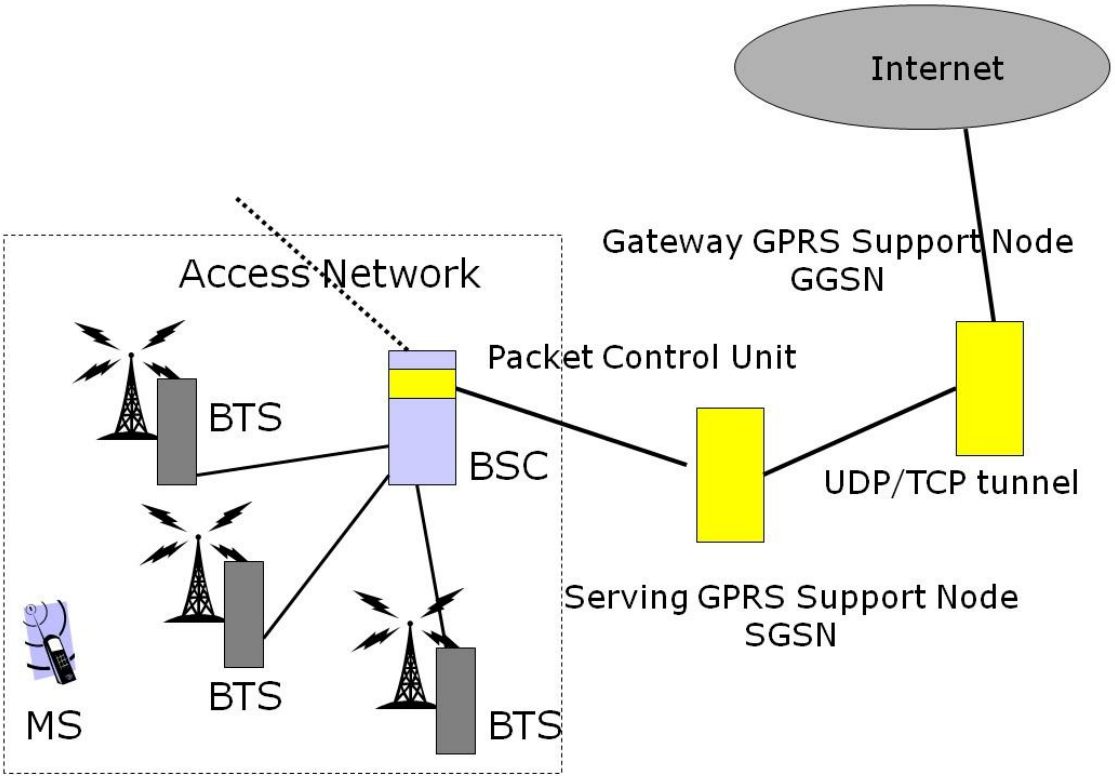
2G:stä eli toisen sukupolven matkapuhelinverkoista puhuttaessa tarkotetaan piirikytkentäistä GSM-verkkoa ja siihen myöhemmin lisättyjä pakettiliikennepalveluja GPRS:ää ja EDGE:ä. Maailman ensimmäinen kaupallinen GSM-verkko rakennettiin Suomeen vuonna 1991 ja sen rakennutti operaattori nimeltä Radiolinja joka on nykyisin osa Elisa Oyj:tä.

2.1 GSM

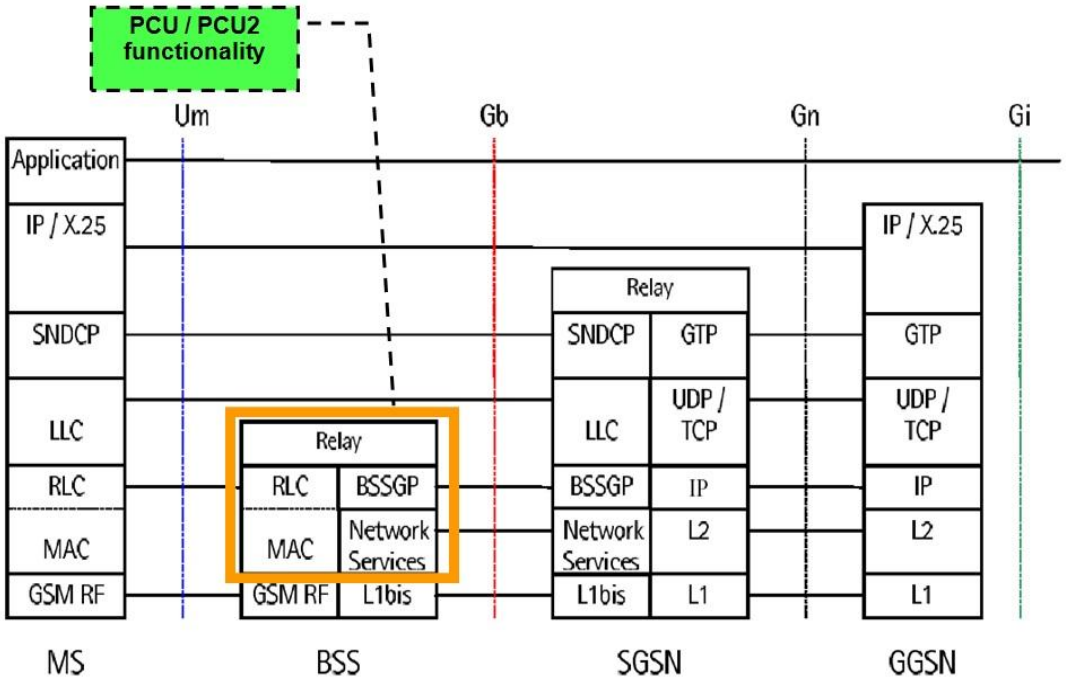
GSM on European Telecommunications Standard Institute(ETSI):n kehittämä standardi, joka määrittelee protokollat toisen sukupolven matkapuhelinverkolle. Alun perin standardi sisälsi vain piirikytkentäisen liikenteen standardin, mutta sitä on laajennettu sisältämään myös pakettiliikenteen eli GPRS- ja EDGE-standardin. Tässä opinnäytetyössä keskitytään pelkästään pakettiliikenteeseen.

GSM-verkko on ensimmäinen täysin digitaalinen matkapuhelinverkko, lisäksi se toi paljon uusia ominaisuuksia, joita ei edeltäneessä NMT-verkossa ollut. Tällaisia ominaisuuksia ovat esimerkiksi puhelun salaus, verkkovierailut ja tekstiviestit.

GSM-verkko koostuu useista eri verkkoelementeistä ja jokaisella niistä on omat tehtävänsä verkon toiminnassa. Kaikki verkkoelementit ovat yhteydessä toisiinsa joko suoraan tai välillisesti jonkin rajapinnan kautta.



KUVA 1. GSM-verkon pakettiliikenteen infrastruktuuri (Montelius, 2002)



KUVA 2. GPRS protokollakerrokset (Abou Aun, Karam & Peignot, 2002)

2.2 GPRS

GPRS (General Packet Radio Service) on tiedonsiirtopalvelu, jota käytetään pääasiassa internet-yhteyden muodostamiseen matkapuhelimen tai GPRS-sovittimen avulla. GPRS-yhteyden teoreettinen maksiminopeus on 171,2kbps, mutta tyypillisesti nopeus on n. 50-60kbps Nykyaikaiset matkapuhelimet kuitenkin usein käyttävät uudempia ja nopeampia 3G- ja 4G-tekniikoita Internet-yhteyden muodostamiseen, mutta GPRS-yhteyttä käytetään edelleen monissa laitteissa. Esimerkiksi seuraavat solaitteet toimivat usein GPRS:n avulla: maksukorttipäätelaitteet, limsa-automaatit, riistakamerat ja etäluettavat sähkömittarit. (Abou Aun, Karam & Peignot, 2002.)

2.3 EDGE

EDGE (Enhanced Data rates for GSM Evolution) – tekniikka toi monia uusia parannuksia GPRS-tekniikkaan, jotka paransivat datansiirtonopeuksia huomattavasti. Uusi modulaatiomenetelmä ja virheitä kestävä siirtomenetelmä yhdistettynä parannettuihin linkkiadaptaatiomenetelmiin nosti teoreettisen maksimi siirtonopeuden jopa 473,6kbps:iin. Tyypillisesti siirtonopeuden kuitenkin ovat noin 200kbps luokkaa. Etenkin uusi modulaatiomenetelmä paransi radiotaajuuksien käytön tehokkuutta. Alla olevassa taulukossa on vertailtu GPRS:n ja EDGE:n datansiirtonopeuksia. (EDGE technical white paper)

TAULUKKO 1. Tiedonsiirtonopeudet

	GPRS	EDGE
Modulation	GMSK	8-PSK/GMSK
Symbol rate	270 ksym/s	270 ksym/s
Modulation bit rate	**270** kb/s	**810** kb/s
Radio data rate per time slot	22,8 kb/s	69,2 kb/s
User data rate per time slot	**20** kb/s (CS4)	**59,2** kb/s (MCS9)
User data rate (8 time slots)	160 kb/s	473,6 kb/s
	(182,4 kb/s)	(553,6 kb/s)

3 TUKIASEMAOHJAIN

Tässä työssä perehdytään tukiasemaohjaimeen eli BSC:hen (Base Station Controller). Tukiasemaohjaimen tehtävä GSM-verkossa on huolehtia oman alueensa radioresurssien hallinnasta. Tämä tarkoittaa sitä että tukiasemaohjain määrittelee mihin tukiasemaan matkapuhelin on yhteydessä ja kuinka paljon radioresursseja sille allokoidaan. Tukiasemaohjain esimerkiksi pitää huolen siitä, ettei matkapuhelinverkko tukkeudu, jos tietyllä alueella on paljon matkapuhelimia. Jos tietyllä tukiasemalla ei ole tarpeeksi radioresursseja vapaana tai signaali matkapuhelimen ja tukiaseman välillä on heikko, tukiasemaohjain yhdistää matkapuhelimen toiseen tukiasemaan. Tätä toimintoa kutsutaan handover:ksi. (Rantala 1997)

Tukiasemaohjain myös tarjoaa palveluja rajapinnoille, jotka ovat tukiasemaohjaimen ja muiden verkkoelementtien välillä. Nämä rajapinnat ovat Abis-rajapinta tukiasemaohjaimen ja tukiaseman välillä, Gb-rajapinta tukiasemaohjaimen ja runkoverkon välillä.
(Rantala 1997)

Tukiasemaohjaimen toiminta on jaettu eri yksiköille, joilla jokaisella on oma tehtävänsä. Tukiasemaohjaimen sisällä olevien yksiköiden välistä viestintää kutsutaan DX-viestinnäksi. Esimerkiksi BCSU hoitaa piirikytkentäisten puheluiden signalointiin liittyviä toimintoja. BCSU:ssa on 8 korttipaikkaa, joihin voidaan kiinnittää erilaisia pistoyksiköitä, kuten paketinohjausyksiköitä.
(Hartikainen 2007)

3.1 Paketinohjausyksikkö

Paketinohjausyksikkö(PCU) lisättiin tukiasemaohjaimeen vuonna 1999, kun tarvittiin erillinen yksikkö hoitamaan pakettikytkentäliikennettä tukiasemaohjaimessa. Jos tukiasemaohjain huomaa kanavan olevan pakettikytkentäinen dataliikennekanava, se allokoi kanavan paketinohjausyksikölle, joka siitä lähtien ohjaa kanavan toimintaa. Tämä tarkoittaa sitä, että paketinohjausyksikkö hoitaa kanavan radioresurssien hallinnan. Paketinohjausyksikkö siis hoitaa radioresurssien allokoinnin matkapuhelimelle. Tämän lisäksi paketinohjainyksikkö hoitaa Abis- ja Gb-rajapinnan toimintoja. PCUSIG – raja-

pinta on paketinohjausyksikön ja DX:n välinen rajapinta eli tämän rajapinnan kautta paketinohjausyksikkö viestii muiden tukiasemaohjaimessa olevien yksiköiden kanssa. (Surana, Kakkar, Goyal, Gandhi, Jain, Das & Malhotra 2011)

Nokia networks:lla on tuoteportfoliossaan useita eri paketinohjausyksikkö malleja. Vanhempaa arkkitehtuuria edustavat pcu2-d ja pcu2-e mallit, joita käytetään flexi BSC:ssä. Uudempaa arkkitehtuuria ovat pcum-a ja pcum-b malit, joita käytetään mcBSC:ssä. Eri korttimallit eroavat toisistaan sekä piiriarkkitehtuuriltaan että ohjelmistoltaan. Esimerkiksi prosessoriydinten ja sisäisen muistin määrä on huomattavasti suurempi uusissa pcum korttivarianteissa, jolloin ne myös ovat tehokkaampia kuin vanhemmat pcu2 korttivariantit. (Kumar 2013)

3.2 Abis – rajapinta

Tukiasemaohjaimen ja tukiaseman välistä rajapintaa kutsutaan Abis – rajapinnaksi. Nokialla on käytössää kaksi erilaista ratkaisua Abis – rajapinnan toteutukseen. Alkuperäinen Legacy Abis – rajapinnan signalointi, on toteutettu käyttäen Link Access Protocol – protokollaa. Myöhemmin julkaistun Packet Abis – rajapinnan signalointi on toteutettu käyttäen IP – protokollaa.

3.3 Gb – rajapinta

Myös Gb – rajapinnasta Nokialle on olemassa kaksi erilaista toteutusta. Alun perin Gb – rajapinta toteutettiin käyttäen Frame Relay(FR) – protokollaa, mutta myöhemmin myös Gb – rajapinta toteutettiin käyttäen IP – protokollaa. Nykyään Frame Relay:lla toteutettuja rajapintoja ei enää asiakkaille ole käytössä harvoja poikkeuksia lukuun ottamatta. Nykyään Nokialla on myös tehty mahdolliseksi yhdistää paketinohjausyksikkö moneen eri SGSN:ään, jolloin syntyy monta rinnakkaista Gb –rajapintaa, tätä ominaisuutta kutsutaan nimellä Multipoint Gb over IP. (Mustajärvi 2002a)

4 PAKETINOHJAUSYKSIKÖN OHJELMISTOTESTAUS

Paketinohjausyksikön testaus on monivaiheinen prosessi, jolla varmistetaan ettei ohjelmistoon jää virheitä. Testaus on tärkeä osa ohjelmistonkehitystä ja testauksella voidaan varmistaa ohjelmiston laatu ja toimivuus. Ohjelmistoa testataan mahdollisimman kattavasti eri kehitysvaiheissa. Kattavuus saavutetaan usealla eri testivaiheella ja tarpeeksi suurella määrällä erilaisia testitapauksia.

4.1 Yksikkötestaus

Paketinohjausyksikön ensimmäinen testivaihe on yksikkötestaus. Yksikkötestauksella tarkoitetaan ohjelmiston pienimpien mahdollisten toiminnallisuuksien testaamista. Tämä tarkoittaa esimerkiksi yhden funktion testaamista. (Vihavainen 2011.) Paketinohjausyksikön yksikkötestaus on toteutettu CppUTest – sovelluskehyksen avulla rakennetulla omalla sovelluskehyksellä. Vaikka itse ohjelmakoodi on kirjoitettu C-kielellä, voidaan yksikkötestauksessa hyödyntää C++ - kieltä. Yksikkötestaus on kiinteästi sidoksissa itse ohjelmakoodiin, tästä syystä yksikkötestit tehdään yleensä yhdessä ohjelmakoodin kanssa, ja jokainen ohjelmoija tekee omalle koodille yksikkötestit. (Räty 2014)

Yksikkötestit voidaan myös kirjoittaa ennen varsinaista ohjelmakoodia. Tällaista ohjelmistokehitysmenetelmää kutsutaan testivetoiseksi kehitykseksi eli test-driven developement:ksi(TDD). Tällöin ensin kirjoitetaan yksikkötesti joka ei mene läpi, koska tarvittava toteutus puuttuu. Sen jälkeen ohjelmoidaan varsinainen toiminnallinen ohjelmakoodi, jonka jälkeen yksikkötestin tulisi mennä läpi. (Vihavainen 2011)

4.2 Integraatiotestaus

Yksikkötestauksen jälkeen seuraava testausvaihe on integraatiotestaus, tämä opinnäyte painottuu integraatiotestaukseen ja sen automaatioon. Integraatiotestauksen tarkoituksena on varmistaa että eri ohjelmamoduulit toimivat yhdessä. (Nikula 2014)
Integraatiotestaus on tärkeää etenkin suuremmissa järjestelmissä, jotka koostuvat useasta pienemmästä järjestelmästä. Paketinohjausyksikön ohjelmisto on nimenomaan tämän

tyyppinen kokonaisuus. Tästä syystä paketinohjausyksikön integraatiotestausta kutsutaan nimellä SubSytem Integration(SSI) – testaus. Paketinohjausyksikön ohjelmisto on loogisesti jaoteltu alustaohjemistoon(Platform Software) ja aplikaatio-ohjelmistoon(Application software). Aplikaatio-ohjelmisto on jaoteltu erikseen seuraaviin moduuliryhmiin: Gb-rajapinnan moduulit, Abis-rajapinnan moduulit ja resurssienhallintamoduulit. Integraatiotestauksessa testataan moduulien välisiä rajapintoja paketinohjausyksikön ohjelmiston sisällä, mutta myös PCUSIG – rajapintaa. (Surana ym. 2011)

4.3 Järjestelmätestaus

Integraatiotestauksen jälkeen seuraava testivaihe on järjestelmätestaus eli systeemitestaus. Järjestelmätestauksessa testataan kokonaista asiakkaalle toimitettavaa järjestelmää kokonaisuutena. Usein järjestelmätestaukseen yhdistetään myös kuormatestausta ja stabiilisuustestausta. Tukiasemaohjaimen järjestelmätestausta suoritetaan useassa eri paikassa erilaisilla testiympäristöillä. Systeemitestausta tehdään sekä simuloiduissa ympäristöissä kuin oikeillakin verkkoelementeillä. Koska tukiasemaohjaimen systeemitestaus on jaoteltu todella moneen rinnakkaiseen ja peräkkäiseen testivaiheeseen, ja sitä tehdään eri puolella Suomea, en siihen tässä työssä perehdy. (Kautto 1997)

4.4 Testausautomaatio

Koska testaus vie paljon aikaa ja on suurimmaksi osaksi samojen asioiden toistamista, pyritään testausta automatisoimaan niin paljon kuin mahdollista. Tämä tarkoittaa sitä, että tietokoneohjelmisto ajaa testit ja kerää testitulokset automaattisesti. Tällöin käyttäjän tarvitsee vain tehdä testitapauksia, lisätä ne testausjärjestelmään ja tarkistaa että testit menevät läpi. Automaatio myös varmistaa sen, että testit ajetaan joka kerta samalla tavalla eikä inhimillisiä virheitä pääse syntymään.

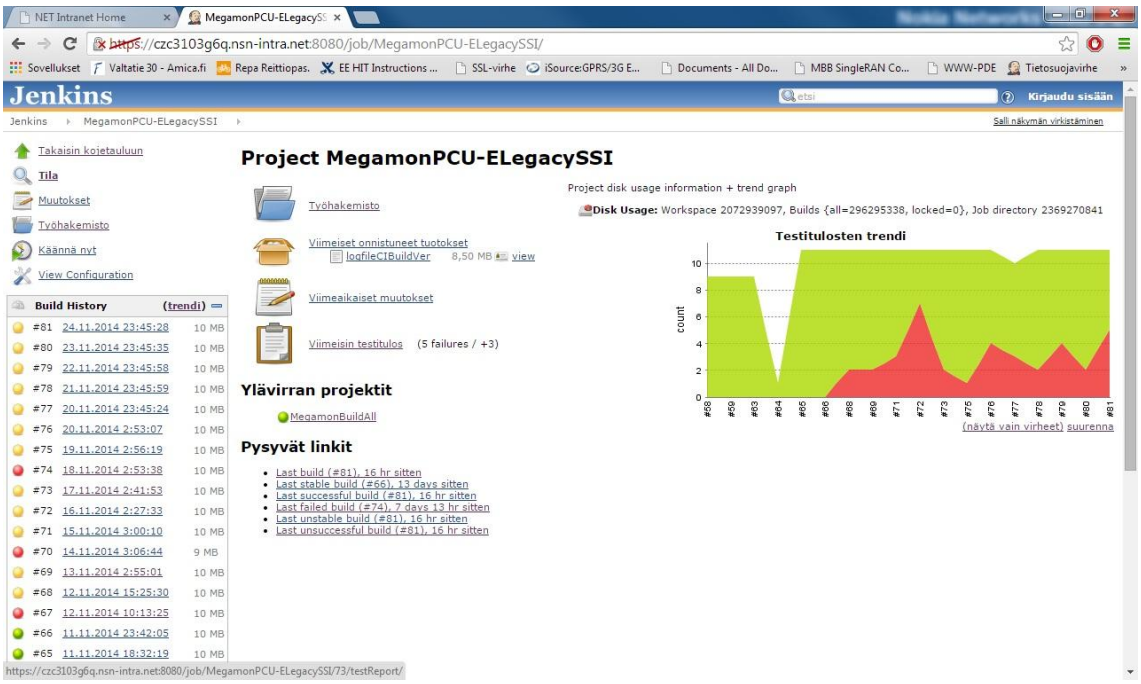
4.5 Jatkuva integraatio

Jatkuva integraatio eli Continuous Integration(CI) on prosessi, jossa ohjelmistoa koostetaan, integroidaan ja testataan jatkuvasti. Perinteisissä ohjelmistokehitysmalleissa kuten vesiputousmallissa ohjelmisto koostetaan ja integroidaan vasta projektin lopussa. Ketterissä ohjelmistokehitysmenetelmissä kuten scrum:ssa käytetään kuitenkin jatkuvaa integraatiota. Jatkuva integraatio ja varmistaa sen, että ohjelmisto ei pirstaloidu ja että ohjelmistovirheet huomataan niin aikaisessa vaiheessa kuin mahdollista, jolloin ne on helpompaa korjata. Samojen testien ajamista uudelleen jokaisen ohjelmistomuutoksen jälkeen kutsutaan regressiotestaukseksi. (Poimala & Tolvanen)

Paketinohjausyksikön ohjelmiston jatkuvassa integroinnissa käytämme Javalla toteutettua avoimen lähdekoodin Jenkins CI – järjestelmää. Käytännössä järjestelmään luodaan erilaisia toimintoja eli jobeja, jotka suoritetaan tietyin väliajoin tai tietyn tapahtuman jälkeen automaattisesti. Jobi on usein jokin itse tehty skripti jollain skriptikielellä, joka toteuttaa tietyn toiminnallisuuden. (Kawaguchi, 2014) Me olemme käyttäneet Pythonia näiden skriptien tekemiseen. Meidän CI - järjestelmämme on toteutettu siten, että se suorittaa integraation joka yö, jolloin se tekee seuraavat toimenpiteet automaattisesti:

1. Hakee versionhallinnasta(SVN) uusimman version ohjelmiston lähdekoodeista.
2. Kääntää ohjelmiston eli luo lähdekoodeista konekielisen ohjelman.
3. Ajaa ohjelmistolle yksikkötestit.
4. Siirtää käännetyt levykuvat FTP-palvelimelle ja ajaa ohjelmistolle integraatiotestit, tässä käytetään ohjelmoimaani integraatiotestausjärjestelmää
5. Hakee tulokset FTP-palvelimelta Jenkins – serverille

Nämä samat toimenpiteet tehdään kaikille eri korttimallien ohjelmistoille.



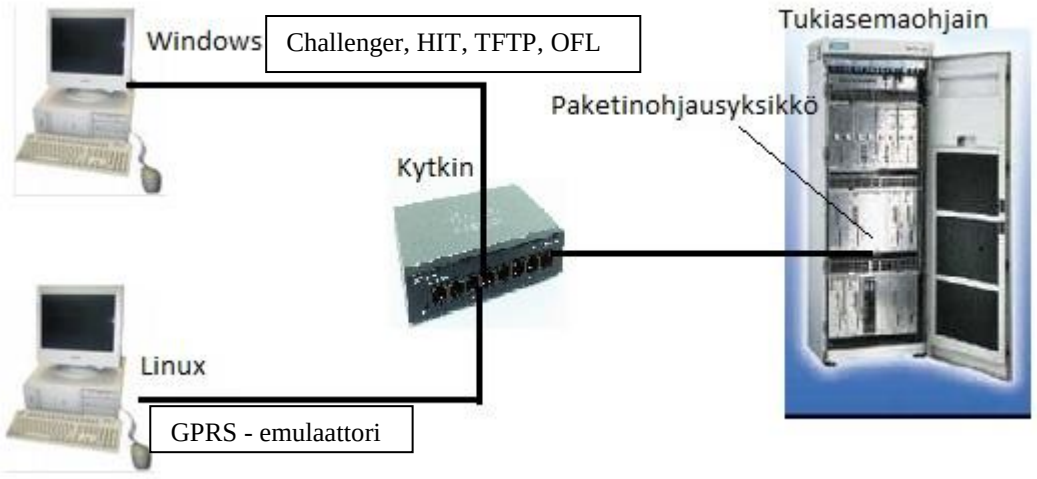
KUVA 3. Kuvakaappaus Jenkinsistä PCU2-e kortin SSI – testituloksista.

5 PAKETINOHJAUSYKSIKÖN INTEGRAATIOTESTAUKSESSA KÄYTETYT TYÖKALUT

Paketinohjausyksikön integraatiotestauksessa käytetään täysin simuloitua testausympäristöä. Testausympäristössä simuloidaan paketinohjausyksikön muita yksiköitä, joiden kanssa paketinohjausyksikkö viestii käyttäen PCUSIG – rajapintaa. Testausympäristössä simuloidaan myös tukiasemia, joiden kanssa paketinohjausyksikkö viestii Abis – rajapinnan kautta ja runkoverkon SGSN – verkkoelementtiä, jolle paketinohjausyksikkö viestii Gb – rajapinnan kautta. Tämän lisäksi simuloidaan matkapuhelinta, jolloin voidaan simuloida oikeata matkapuhelimen ja runkoverkon välistä tiedonsiirtoa, joka kulkee paketinohjausyksikön kautta.

5.1 Testausympäristöt

Yksi testausympäristö koostuu yhdestä windows - käyttöjärjestelmällä varustetusta tietokoneesta, yhdestä linux fedora 9 – käyttöjärjestelmällä varustetusta tietokoneesta sekä paketinohjausyksiköstä. Käytössämme on yksi flexi-BSC, jossa on neljä kappaletta pcu2-d paketinohjausyksikköä ja neljä pcu2-e paketinohjausyksikköä. Näistä ympäristöistä puolet on Legacy Abis – rajapinnalla toteutettu ja puolet on Packet Abis - toteutuksia. Tämän lisäksi käytössämme on yksi mcBSC, jossa on kaksi kappaletta pcum-a paketinohjausyksikköä ja kaksi kappaletta pcum-b paketinohjausyksikköä. Yhteensä olemme siis rakentaneet tähän mennessä kaksitoista integraatiotestausympäristöä.



KUVA 4. Testausympäristö

5.2 GPRS – emulaattori

GPRS – emulaattori(GEMU) on ohjelmisto, jolla voidaan emuloida GPRS – verkkoelementtejä. GEMU toimii ainoastaan linux – käyttöjärjestelmällä. GEMU on kehitetty nimenomaan tukemaan pakettikytkentäisen liikenteen emulointia ja piirikytkentäisen liikenteen ominaisuudet ovat vähäiset. GEMU tarjoaa oikeat rajapinnat ja protokollat emuloiduista verkkoelementeistä ulospäin. Emuloitujen verkkoelementtien protokollat ja rajapinnat toteuttavat Nokian oikeiden verkkoelementtien toteutuksen niin täydellisesti, että niitä voidaan jopa käyttää oikeissa toimivissa GPRS - verkoissa. (Mustajärvi 2002b)

GEMU on toteutettu käyttäen C-kieleen pohjautuvaa virtuaalista käyttöjärjestelmää, CVOPS:ia. CVOPS:n ensimmäisen version on kehittänyt teknologian tutkimuskeskus VTT 80-luvun puolivälissä. CVOPS on protokollaohjelmistojen kehitystyökalu ja ajoympäristö. CVOPS tarjoaa funktiokirjaston ja kommunikaatiopalvelut, joita voidaan käyttää kuten perinteistä ohjelmointirajapintaa(API). CVOPS:n tärkein ominaisuus on muuntaa tekstimuodossa määritelty protokolla tietokoneohjelmaksi käyttäen C-kääntäjää. (Pesola 1998)

GEMU:a käytetään CVOPSIN tarjoaman alkeellisen käyttöliittymän kautta. Käytännössä GEMU:a käytetään siten, että eri GEMU prosessit käynnistetään omiin komentoikkunoihinsa, joihin voidaan antaa ennalta määriteltyjä komentoja. Minkäänlaista graafista käyttöliittymää ei ole, mikä saa GEMU:n käyttämisen alkuun tuntumaan melko haastavalta. Haastavuutta lisää se, että prosesseilla on tekstimuotoinen konfiguraatiotiedosto, joissa saattaa olla monta sataa parametria liittyen kyseisen prosessin emuloiman verkkoelementin toimintaan. Usein tilanne on se, että jos yksikin parametri on väärin, eivät GEMU prosessit toimi ollenkaan. Juurikin GEMU:n käyttämisen haasteellisuus on yksi syy miksi integraatiotestausta halutaan automatisoida.

5.2.1 ETP – emulaattori

ETP – emulaattori ei varsinaisesti ole osa GEMU – ohjelmistoa, mutta sitä käytetään samalla tietokoneella kun GEMU:a ja se viestii GEMU – prosessien kanssa.

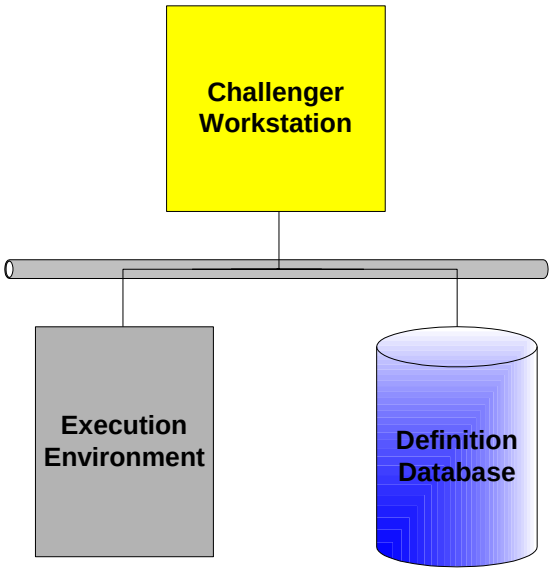
ETP on yksi tukiasemaohjaimen yksikkö, joka hoitaa viestien lähettämisen ja vastaanottamisen Packer Abis – rajapinnalla. ETP:tä voidaankin periaatteessa ajatella tukiasemaohjaimen verkkokorttina Abis – rajapinnan suuntaan. ETP – emulaattori siis käsittelee ja välittää paketinohjausyksiköltä GEMU:n tukiasemaprosessille menevät viestit, ja toisaalta välittää myös tukiasemaohjain prosessilta tulevat viestit paketinohjausyksiköl-le. ETP - emulaattoria tarvitaan ainoastaan ympäristöissä, joissa on käytössä Packet Abis – rajapinta, Legacy Abis – rajapinnalla toteutetuissa ympäristöissä on käytössä tukiasemaohjaimen oikea ET – yksikkö, jolloin emulaattoria ei tarvita. (Bajaj 2009)

5.2.2 PTI – kortti

Kaikkien ympäristöjen linux - tietokoneeseen on asennettuna ns. PTI – kortti, joka saa nimensä valmistajan Performance Technology Incorporated mukaan. PTI – kortti asennetaan tietokoneen emolevyn PCI – väylään. PTI – kortin tehtävä on synkronoida Abis – rajapintaa emuloivien GEMU – prosessien ja paketinohjausyksikön välinen signalointi. Käytännössä tämä tapahtuu siten, että PTI – kortti lähettää 20 millisekunnin välein synkronointisanomaa paketinohjausyksikölle, jolloin GEMU – prosessit ja paketinohjausyksikkö pysyvät samassa syklissä. (Elovaara 2007)

5.3 Challenger - työkalu

Challenger on graafinen työkalu, jolla voidaan simuloida paketinohjausyksikön ja muiden tukiasemaohjaimen yksiköiden välistä viestintää eli ns. DX - signalointia. Tämä tarkoittaa sitä, että Challengerin avulla voidaan lähettää ja vastaanottaa viestejä paketinohjausyksikön kanssa, tarkastella viestien sisältöä ja tarkastaa millaisia vasteita paketinohjausyksikkö lähettää eri viesteihin. Challenger toimii ainoastaan windows – käyttöjärjestelmällä. Challenger tarvitsee toimikseen myös erillisen Challenger Agentin ja VMB:n. Nämä ohjelmat ohjaavat Challengerilla luodut viestit paketinohjausyksikölle sekä paketinohjausyksiköltä tulevat viestit Challenger – työkalulle. Ohjelmat myös myös muokkaavat lähtevät viestit paketinohjausyksikön ymmärtämään formaattiin.
(Olkku 2007)

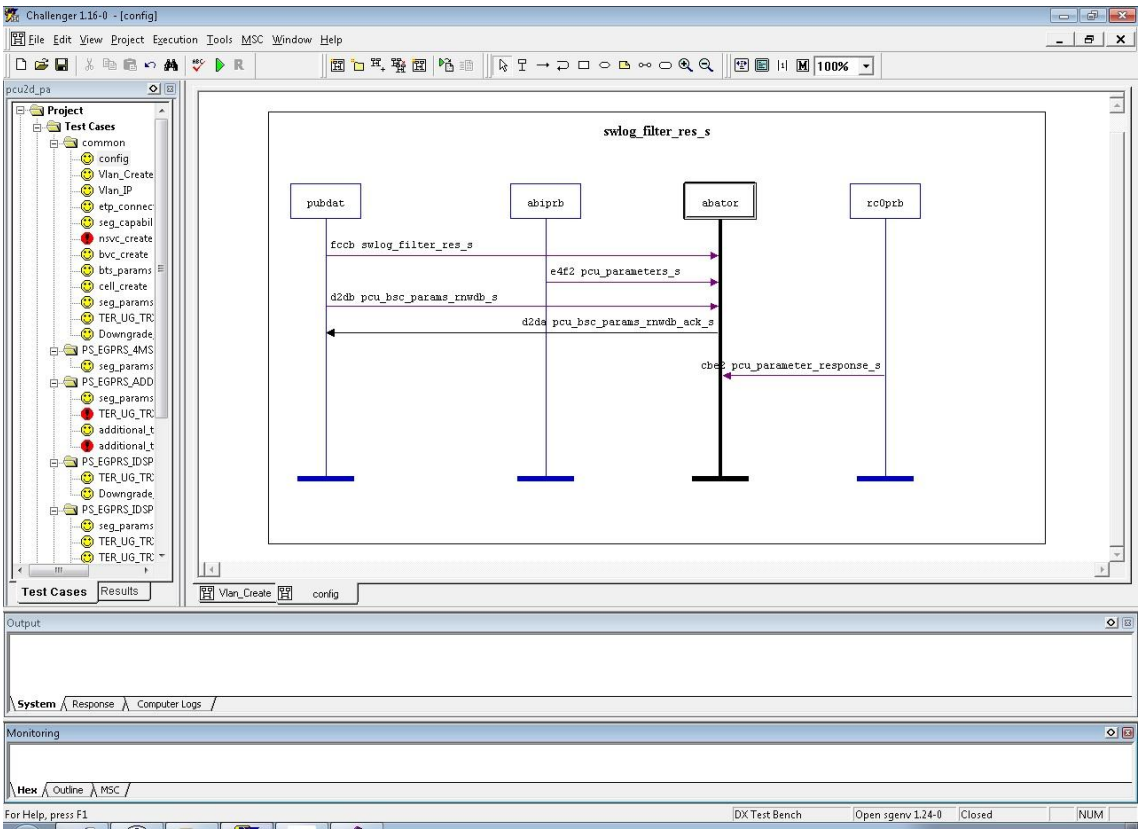


KUVA 5. Challenger arkkitehtuuri

Challenger on tärkein testauksessa käytettävä työkalu, koska periaatteessa testitapaukset tehdään Challengerilla, vaikka ne koostuvat myös muista osista. Näistä kerrotaan seuraavassa luvussa lisää. Challenger on hyvä työkalu paketinohjausyksikön testaamiseen, koska se on kehitetty Nokian sisällä ja suunniteltu nimenomaan Nokian tuotteiden testaamiseen. (Olkku 2007)

5.3.1 Challenger - työkalun käyttäminen

Challenger testitapaukset tehdään graafisen käyttöliittymän avulla käyttäen Message Sequence Chart(MSC) – notaatiota. Piirretään prosessit joiden välistä viestintää halutaan mallintaa, sen jälkeen piirretään viestit prosessien välille ja täytetään viestien sisältö. Käytännössä paketinohjausyksikköä kuvataan yhdellä prosessilla ja muissa tukiasemaohjaimen yksiköissä olevia prosesseja niin monta kuin testitapauksessa on tarvetta. (Olkku 2007)



KUVA 6. Challengerin käyttöliittymä.

Viestit on kuitenkin automaattisesti täytetty oletusarvoilla. Nämä oletusarvot Challenger lukee PDE:stä eli sack:sta. Käytännössä sack on .dat – muotoinen tiedosto, johon määriteltynä viestien nimet, viestissä olevien kenttien nimet, viestissä olevien kenttien tyypit ja viestissä oleville kentille oletusarvot. Sack – tiedosto on määriteltynä Challengerin asetuksiin. Myös paketinohjausyksikön IP – osoite on määritelty Challengerin asetuksiin, vastaavasti Challenger – tietokoneen IP – osoite on asetettu paketinohjausyksikön flash – muistiin. Näin ollen Challenger ja paketinohjausyksikkö voivat viestiä toisilleen lähiverkossa. (Olkku 2007)

Vaikka Challengeria käytetään graafisen käyttöliittymän avulla ja testitapauksen tehdään graafisesti, on Challengeriin toteutettu myös komentorivipohjainen käyttöliittymä. Tämän käyttöliittymän avulla, voidaan Challenger – testitapauksia ajaa komentoriviltä. Tämä toiminnallisuus on toteutettu, jotta Challengeria voidaan ohjata HIT – makrojen avulla. (Olkku 2007)

5.4 Offline Logger

Offline Logger eli OFL on työkalu, jolla paketinohjausyksikön logeja voidaan kerätä. Normaalisti paketinohjausyksikön logitiedostot menevät tukiasemaohjaimessa olevan OMU – yksikön kovalevylle, mutta koska integraatiotestausympäristöissä ei OMU:a ole asennettuna, logit kerätään OFL-työkalun avulla Windows tietokoneella. OFL on Windows komentoriviltä toimiva ohjelma, joka vastaanottaa paketinohjausyksiköltä tulevaa logia ja kirjoittaa sen tietokoneen levylle tekstitiedostoon. Käytännössä paketinohjausyksikölle kerrotaan IP – osoite ja portti, johon sen halutaan logit lähettävän reaaliajassa, ja OFL ohjelmalle kerrotaan portti, johon se logia odottaa.

5.5 TFTP - palvelin

TFTP eli Trivial File Transfer Protocol on yksinkertainen tiedostojensiirtoprotokolla, jota käytetään useimmiten esimerkiksi käynnistyslevykuvan siirtämiseen laitteeseen. TFTP on todella yksinkertainen toteuttaa pienellä määrällä koodia, jonka vuoksi sitä suositaan laitteissa, joissa muistia ei ole paljon käytössä. Toisaalta TFTP on niin yksinkertainen protokolla, että siitä puuttuu paljon ominaisuuksia, joita muissa tiedostonsiirtoprotokollissa on. TFTP:ssä liikennettä ei esimerkiksi salata mitenkään, jonka takia sitä nykyään käytetään lähinnä tiedostojen siirtoon lähiverkossa. (Sollins 1992)

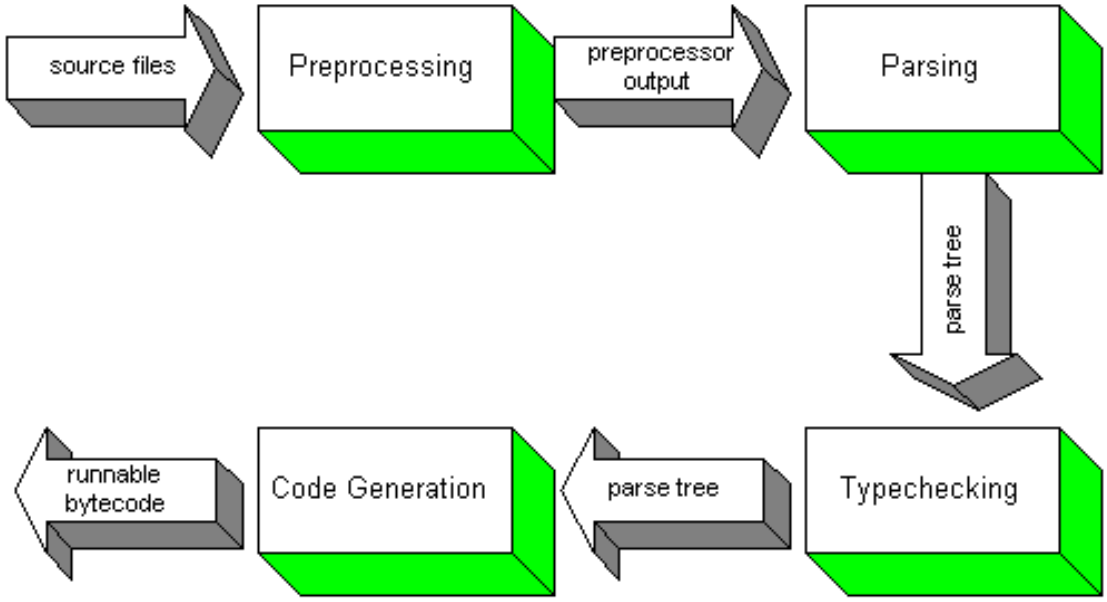
Testausympäristön Windows tietokoneelle on asennettuna Solarwinds TFTP - serveri. Kun paketinohjausyksikkö uudelleen käynnistetään, se lataa ohjelmiston levykuvat tältä TFTP – serveriltä. TFTP – serverin IP-osoite on tallennettuna paketinohjausyksikön Flash – muistiin, jolloin serverin osoite säilyy paketinohjausyksikön muistissa, vaikka siitä katkaistaisiin virta.

5.6 HIT – ohjelmointiympäristö ja ohjelmointikieli

HIT eli Holistic Integration Tester on Nokian kehittämä ohjelmointiympäristö(IDE) ja ohjelmointikieli. HIT – kieli on C-kielen kaltainen ohjelmointikieli, mutta siinä on myös monia merkittäviä eroja C-kieleen verrattuna. HIT – kielessä ei ole osoittimia, vaan muuttujien ainoat tietotyypit ovat kokonaisluku int, liukuluku float ja merkkijono string.

HIT – kielessä muistia ei tarvitse tai voi varata dynaamisesti, vaan muistinkäsittely on staattista. (Aartoaho 2011)

HIT – kieli on makrokieli, mikä tarkoittaa sitä että lähdekoodia ei käännetä konekielelle vaan se tulkataan tavukielelle toisin kuin C-kieli. HIT – kielen syntaksi myös eroaa osittan C-kielen syntaksista, HIT – kielessä ei esimerkiksi käytetä hakasulkeita. HIT - makrojen tiedostopääte on on .tel. (Aartoaho 2011)



KUVA 7. HIT – kielen tulkkausprosessi(Aartoaho 2011)

HIT – kieli noudattaa proseduraalista ohjelmointiparadigmaa, mikä tarkoittaa sitä että ohjelma jaetaan useiksi aliohjelmiksi eli funktioiksi. Yksi HIT – makro on siis vain lista erilaisia funktioita, joista ensimmäinen eli ylimpänä oleva funktio on main – funktio, jossa kutsutaan muita funktioita. Kun HIT – makro ajetaan, suoritetaan siitä aina vain main – funktio. HIT – kielessä ei ole minkäänlaista tukea olio-ohjelmoinnille. HIT – kieleen on kehitelty oma standardikirjasto, joka sisältää satoja valmiita funktioita käytettäväksi HIT – makroissa. HIT:ssä on myös listatoiminnallisuus, joka tarkoittaa sitä että eri makroja voidaan koota yhteen listatiedostoon, kun lista ajetaan, ajetaan kaikki listassa olevat makrot järjestyksessä. (Aartoaho 2011)

HIT – kielen yksi tärkeimpiä ominaisuuksia on device – toiminnallisuus ja -funktiokirjasto, mikä helpottaa testiympäristön eri laitteiden hallintaa. Voit HIT:n ohjelmointiympäristön graafisen käyttöliittymän kautta määritellä eri laitteita, joihin voidaan HIT – makrosta ottaa helposti yhdellä funktiokutsulla. Sinun täytyy vain määritellä

laitteelle IP – osoite, käyttäjätunnus ja salasana, otatko yhteyden telnet vai SSH – protokollan avulla ja laitteen nimi. Otettuasi yhteyden laitteeseen, voit helposti lähettää sille komentoja mikä vastaa sitä kuin käyttäisit laitetta lokaalisti komentoriviltä. Integraatiotestauksen sovelluskehyksessä laitteiksi on määritelty paketinohjausyksikkö sekä linux – tietokone johon asennettuna gprs - emulaattori.

HIT – ohjelmointiympäristöön on myös integroituna HIT – serveri. HIT – serveriin voi ottaa yhteyden käyttäen telnet - protokollaa. HIT – serverille voi antaa tiettyjä ennalta määritettyjä komentoja kuten esimerkiksi ”RM makron_nimi”, joka ajaa makron_nimi nimisen makron.

6 PAKETINOHJAUSYKSIKÖN INTEGRAATIOTESTAUKSEN SOVELLUSKEHYS

Paketinohjausyksikön sovelluskehys on toteutettu HIT – makroilla ja perl – skripteillä. HIT – makrojen avulla ohjataan linux – koneella olevan gprs – emulaattorin toimintaa, paketinohjausyksikön toimintaa ja HIT:n kanssa samalla koneella olevan OFL- ja Challenger – työkalun toimintaa. Linux – koneelle yhteys otetaan HIT – makrolla käyttäen SSH – prokollaa ja paketinohjausyksikköön yhteys käyttäen telnet – prokokollaa. Vaikka telnet – protokollalla toteutettua yhteyttä ei ole salattu, se ei haittaa, koska laboratorioverkkomme on eristetty Internetistä.

Paketinohjausyksikön testaaminen on nyt täysin automatisoitu, mikä tarkoittaa sitä että testit ajetaan automaattisesti joka yö Continuous Integration – järjestelmästä. Mutta käyttäjä voi myös ajaa testi päälle manuaalisesti telnet - yhteyden kautta yhdellä komennolla komentoriviltä tai vaihtoehtoisesti ottamalla yhteyden testiympäristön Windows tietokoneeseen käyttäen Windowsin etätyöpöytäominaisuutta. Tällöin saat omalle koneellesi näkymään testiympäristön Windows- tietokoneen työpöydän samaan tapaan kuin käyttäisit konetta lokaalisti.

6.1 Konfiguraatiotiedosto

Paketinohjausyksikön sovelluskehys sisältää general_ssi_automation_config.tel tiedoston, johon määriteltynä ympäristökohtaiset vakiomuuttujat, sekä jatkuvaan integraatioon liittyviä parametrejä. Konfiguraatiotiedostoon on määritelty parametri ”CI”, jonka arvoa vaihtamalla voi määrittää onko ympäristö konfiguroituna Jenkis CI – järjestelmään, josta on kerrottu tarkemmin luvussa 4.5. Mikäli parametrin arvon on ”y” tai ”Y”, lataa HIT – makro testiajon alussa ohjelmiston levykuvat FTP – palvelimelta ja testiajon lopuksi lähettää tulokset samalle FTP – palvelimelle. FTP – palvelin siis toimii Jenkins CI - järjestelmän ja integraatiotestaussovelluskehyksen välisenä rajapintana. Alla on esimerkki yhden testausympäristön konfiguraatiotiedostosta, IP – osoitteet on sensuroitu tietoturvasyistä

```
//*-------------------------------------------------------------------------------
//*
//* Filename        :  ssi_automation_config.tel
//*
//* Description     :  This file contains configuration parameters to run ssi_automation hit scr
//*
//*

//-----------------environment specific-----------------
#define GEMU_IP
#define CHALLENGER_IP
#define CHALLENGER_PROJECT_PATH         "C:\\challenger\\pcu2e_la.mtp"

#define CHALLENGER_EVENT_LOG_FILE       "C:\\challenger\\eventlogs\\eventlog.txt"
#define CHALLENGER_PATH                 "C:\\Program Files (x86)\\Challenger\\bin"
#define HIT_EXE_PATH                    "C:\\Program Files (x86)\\HIT\\SYSTEM\\HIT2.exe"
#define CHALLENGER_AGENT_PATH           "C:\\challenger_agent\\"
#define VMB_PATH                        "C:\\challenger_vmb\\src\\"
#define CHALLENGER_AGENT_EXE            "challenger_agent.exe"
#define STRUCT_FILE_PATH                "gprs3gemulator-5.3.9/system/testcases/"
#define MACRO_FILE_PATH                 "/root/gprs3gemulator-5.3.9/system/macros/testcases/"
#define M2GELGD_PATH                    "/root/gprs3gemulator-5.3.9/system/m2gelgd/"
#define LOG_PATH                        "C:\\logs"

//-------------for Continuous integration use------------

#define CI                              "n" //Change to "y" if environment is in CI use
#define CARD_TYPE                       "pcu2e"
#define FTP_ADDRESS
#define FTP_USERNAME                    "ftpserver"
#define FTP_PASSWORD                    "password"
```

KUVA 8. ssi_automation_config.tel

6.2 Sovelluskehyksen funktiokirjasto

Suurin osa sovelluskehyksen ohjelmakoodista on kirjoitettuna general_ssi_automation.tel nimiseen tiedostoon. Tiedostossa on kymmeniä erilaisia funktioita, joita käytetään jokaisessa yksittäisessä testitapausmakrossa. Jokainen testitapaus on siis yksi HIT – makro johon sisällytetään tämä funktiokirjastotiedosto sekä konfiguraatiotiedosto. Tämän johdosta testitapauksia on helpompi tehdä, koska voidaan käyttää valmiita funktioita, jotka ovat kaikille testitapauksille yhteisiä. Näin ollen koodia on myös helppo ylläpitää, kun muutokset tarvitsee tehdä vain yhteen tiedostoon, eikä jokaiseen testitapaukseen erikseen. Alla olevassa kuvassa on yksi HIT – kielellä tehty funkiokirjaston funktio, joka tarkastaa viimeksi ajetun Challenger testitapauksen tuloksen Challengerin kirjoittamasta logitiedostosta.

```
//*-------------------------------------------------------------------------------
//* Functionname  :  check_challenger_script_status
//*
//* Description      : Checks status of the last executed challenger script
//*                      return 1 OK
//*                      return 0 NOT OK
//*-------------------------------------------------------------------------------
integer function check_challenger_script_status()
  integer status = 0;
  string log_line = "";
  integer file_handle;
  string status_text = "";
  string ok = "\"OK\"";

  file_handle=fileopen(CHALLENGER_EVENT_LOG_FILE,READ);
  if(file_handle<0)
   abort(0,"file not found");
  endif
  fileseek(file_handle,FILE_END);
  if(filegetlineback(file_handle,log_line,"'STATUS='"))
    strfetch(log_line,"'STATUS=':8-11", status_text);
    status = strstr(status_text, ok);
  endif
  fileclose(file_handle);
  return status;
endfunction
```

KUVA 9. HIT – funktio.

6.3 Testiympäristön alustaminen

Jokaisen testiajon alussa paketinhallintayksikkö käynnistetään uudelleen, jolloin se lataa ohjelmiston levykuvat Windows – tietokoneen TFTP – palvelimelta. Tämän jälkeen lähetetään Challengerilla konfiguraatioviestit paketinohjausyksikölle. Näissä viesteissä esimerkiksi konfiguroidaan paketinohjausyksikölle sen käyttämät rajapinnat ja annetaan muitakin kortin toimintaan liittyviä parametrejä. Testiajon alustus ja lopetus toimintoon liittyvät funktiot on ohjelmoituna test_execution.tel nimiseen tiedostoon.

6.4 Testitapauksen rakenne ja toiminnallisuus

Kun testausympäristö on alustettu, eli paketinohjausyksikölle on ladattu testattava ohjelmisto, ja paketinohjausyksikkö on konfiguroitu toiminta valmiuteen, voidaan aloittaa varsinaisten testitapausten ajaminen. Yleisesti testitapauksen HIT - makro koostuu kahdesta funktiosta: main - funktiosta ja make_call – funktiosta, mutta monimutkaisemmissa testitapauksissa voi olla myös muita funktioita. Main – funktion rakenne on jokaisessa testitapauksessa samanlainen, siinä kutsutaan general_ssi_automation.tel – funktiokirjastossa määriteltyjä funktiota, sekä testitapauskohtaista make_call – funktiota. Testitapausten yhtenäinen rakenne helpottaa uusien testitapausten tekemistä sekä testi-

tapausten toiminnallisuuden ymmärtämistä. Alla on kuvakaappaus yhden testitapauksen main – funktiosta ja testitapauksen alussa määritellyistä Challenger testitapauksista. Challenger - testitapaukset on määriteltynä vakiomerkkijonotaulukoihin.

```
const string challenger_config_cases[] =
{
                                        "bts_params",
                                        "seg_params_MCS9",
                                        "edap_egprs"
};

const string challenger_upgrade_case[] = {"TER_UG_egprs"};
const string challenger_downgrade_case[] = {"Downgrade_new"};

function main(string testcase_name)
   copy_macros_testcaseswise(testcase_name);
   start_emulators();
   start_ofl();
   pcu_start_logging();
   stop_and_start_agent_vmb();
   set_bsc_simulator_address(CHALLENGER_IP);
   open_run_challenger(challenger_config_cases);
   take_initial_mem_output();
   open_run_challenger(challenger_upgrade_case);
   set_bsc_simulator_address(GEMU_IP);
   make_call();
   read_through_put_elgdr();
   set_bsc_simulator_address(CHALLENGER_IP);
   open_run_challenger(challenger_downgrade_case);
   take_final_mem_output();
   finish(testcase_name,0);
endfunction
```

KUVA 10. Testitapauksen main – funktio.

Jokaisella testitapauksella on myös linux – tietokoneella oma struct_2g.db_testcase_name - konfiguraatiotiedosto, johon määritelty GPRS – emulaattorin prosesseihin liittyviä parametrejä. Esimerkiksi tietyllä testitapauksella voi olla eri määrä tukiasemia, testitapaus voi olla GPRS- tai EGPRS – testitapaus jne. Tämän lisäksi linux – tietokoneella on elgms_macro.mac_testcase_name niminen konfiguraatiotiedosto, johon määritelty matkapuhelinta emuloivan prosessin toimintaan ja dataliikenteeseen liittyviä parametrejä. Tällaisia parametrejä on esimerkiksi datansiirrossa käytettävä pakettikoko, pakettien lähetysväli, matkapuhelinten määrä ja dataluokka yms.

6.4.1 Yhden testitapauksen analysointi funktiotasolla

Alla on kuvattuna yhdessä testitapauksessa käytettyjen funktioiden toiminta.

- Testimakron copy_macros_testcaseswise(string testcase_name) – funktio asettaa yllä kuvatut testitapauksen konfiguraatiotiedostot aktiiviseksi GPRS – emulaattorille.
- start_emulators() - funktio käynnistää GPRS – emulaattorin prosessit. Käytännössä Linux koneella on lyhyt shell – skripti, joka ajetaan tällä funktiolla.
- start_ofl() – funktio käynnistää offline logger – ohjelman windows - tietokoneelle.
- pcu_start_logging() – funktio antaa paketiohjausyksikölle komennot, joilla käynnistetään logikirjoutuksen ohjaaminen offline loggerille.
- stop_and_start_agent_vmb() – funktio käynnistää challenger – työkalun käytössä tarvittavat challenger agent- ja VMB – ohjelmat.
- set_bsc_simulator_address(string ip_address) – funktio antaa paketinohjausyksikölle komennon, jolla konfiguroidaan paketinohjausyksikön PCUSIG – rajapinnan IP – osoite. Tätä IP – osoitetta vaihdellaan sen mukaan käytetäänkö Challengeria vai GPRS – emulaattoria PCUSIG – rajapinnan vastapuolena.
- take_initial_mem_output() – antaa paketinohjausyksikölle komennot, joilla paketinohjausyksikkö tulostaa näytölle varattujen muistipuskurien koon. Tämä tulostus myös tallennetaan tekstitiedostoon myöhempään analysointia varten.
- open_run_challenger(string challenger_cases[ ]) – funktio ajaa ne challenger testitapaukset, joiden nimi on määriteltynä funktiolle parametrina annetussa taulukossa. Koska HIT – ympäristö ja Challenger – työkalu on asennettuna samalle tietokoneelle, HIT – makro ohjaa Challengeria Windows DOS – komentorivin kautta.
- make_call() – funktio käynnistää GPRS – emulaattorin m2gelgd – prosessin, joka simuloi matkapuhelinten toimintaa. Prosessille annetaan komennot, jotka yhdistävät matkapuhelimet verkkoon ja aloittavat pakettidatansiirron verkossa. Tämän prosessin elgms_macros.mac tiedostoon on määritelty matkapuhelinten ominaisuudet. Jokaisella testitapauksella on siis oma konfiguraatiotiedosto, joka on otettu käyttöön testitapauksen copy copy_macros_testcaseswise() - funktiossa.

- read_through_put_elgdr() – funktio tarkastaa edellä mainitusta m2gelgd - prosessista matkapuhelinten tekemän datansiirron määrän ja laskee sen avulla datansiirtonopeuden sekä tallentaa tämän tiedostoon.
- take_final_mem_output() – antaa paketinohjausyksiköllle komennot, joilla paketinohjausyksikkö tulostaa näytölle varattujen muistipuskurien koon. Tämä tulostus myös tallennetaan tekstitiedostoon myöhempään analysointia varten.

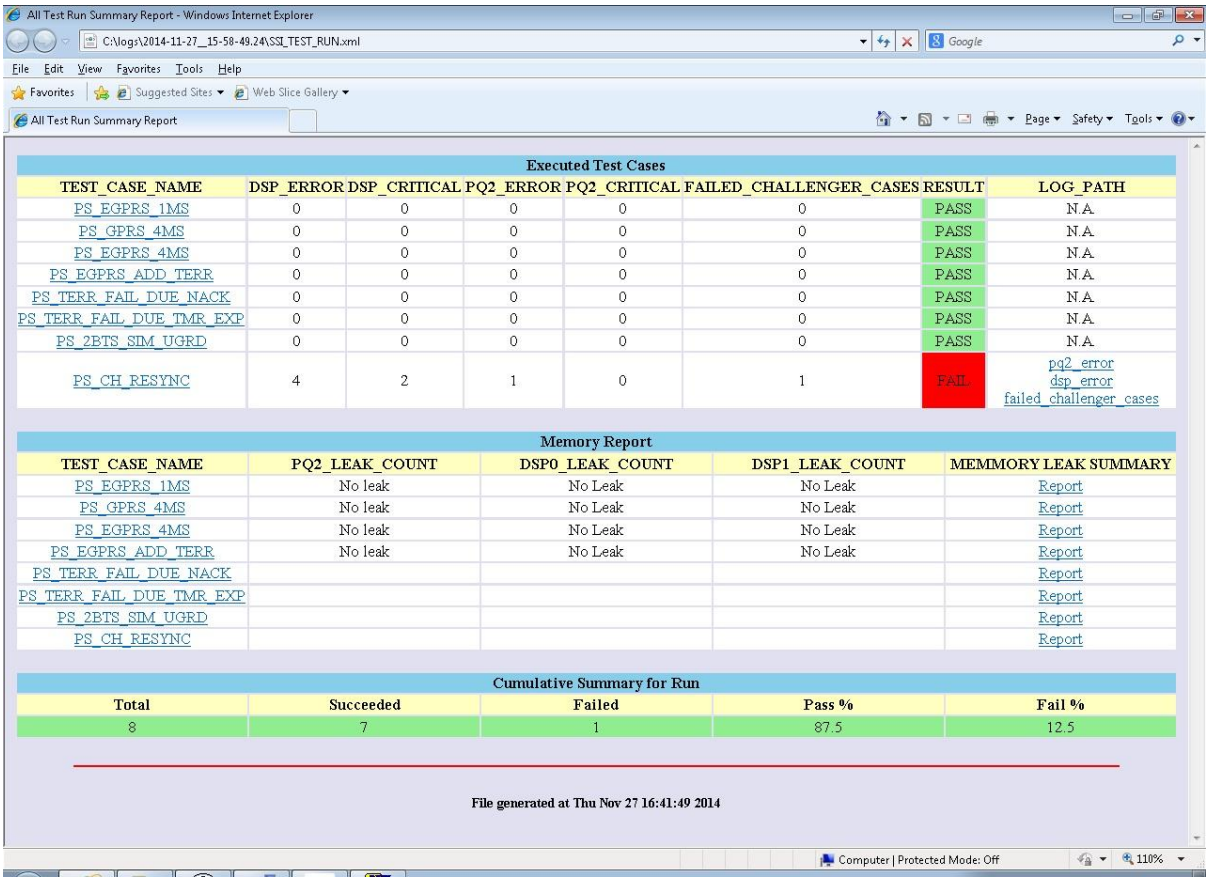
6.4.2 Testitapauksen lopetus ja tulosten luominen

finish() – funktio suoritetaan aina testitapauksen lopuksi. Tämä funktio sulkee kaikki käytetyt työkalut sekä siirtää testitapaukseen liittyvät tiedostot oikeaan kansioon ja poistaa väliaikaiset tiedostot. Tämän funktio myös suorittaa Windows komentorivin kautta perl – kielellä toteutetun skriptin, joka luo testitapaukselle XML – tiedostomuotoa olevan tulostiedoston. Tämän XML – tiedoston luonti on toteutettu erillisellä perl – skriptillä, koska HIT – kielelle ei ole luotu XML – funktiokirjastoa. Tähän XML – tulostiedostoon kirjoitetaan paketinohjausyksikön logitiedostoissa olevien virheilmoitusten määrä, sekä epäonnistuineiden Challenger – testitapausten määrä. Jotta testitapaus menee läpi, eli sen tulokseksi kirjoitetaan ”PASS”, ei lokitiedostoissa saa olla yhtään virheilmoitusta, eikä yhtään testitapaukseen liittyvää Challenger – testitapausta saa epäonnistua. Tulostiedostoon myös kirjoitetaan muistipuskureiden analysoinnista saadut tulokset.

6.4.3 Kokonaisen testiajon lopetus ja tulosten kerääminen

Yhden testiajon lopuksi erillisistä testitapausten XML – tulostiedostoista luodaan yksi XML – tulostiedosto, jossa on koko testiajon tulosten yhteenveto sekä testitapauskohtaiset tulokset. Myös tämän XML – tiedoston luontiin ja kokoamiseen käytetään perl – skriptiä, koska HIT – kielessä ei ole valmista XML – funktiokirjastoa. Lopulliselle tulostiedostolle on myös määritelty tyylitiedosto käyttäen XSL – kieltä. Tällä kielellä voidaan XML – tiedostolle määritellä tietty tyyli, eli se millaiselta tämä tiedosto näyttää kun sen avaa esimerkiksi Internet – selaimessa. (W3Schools)
Alla on kuvakaappaus yhden testiajon tuloksista avattuna Internet Explorer - selaimeen. Joissain testitapauksissa ei ole katsottu tarpeelliseksi tutkia muistipuskureita, koska niis-

sä testataan ainoastaan PCUSIG - rajapinnan signalointia, eikä lainkaan radioresurssienhallintaa, jossa muistinkäyttö on oleellisempaa. Kuten kuvakaappauksesta näkyy niin testitapausten nimet sekä logitiedostopolut on määritelty linkeiksi, jolloin tuloksia voi kätevästi selata suoraan Internet - selaimesta. Testitapauksen nimeä painamalla aukeaa Windowsin tiedostoselaimeen kansio, jossa testitapauksen logitiedostot ovat. Logitiedostopolkulinkkiä painamalla avautuu selaimeen tekstitiedosto, johon esimerkiksi tietyn login kaikki virheilmoitukset on kerätty.



Kuva 11. Testiajon XML - tulostiedosto Internet Explorer - selaimessa.

```
P <initialize> C:\scripts\HIT_scripts\test_execution.tel::pre_testcase_list_execution()
P <tc1> C:\scripts\HIT_scripts\PS_EGPRS_1MS\PCU2_SSI.tel::main("PS_EGPRS_1MS")
P <tc2> C:\scripts\HIT_scripts\PS_GPRS_4MS\PCU2_SSI.tel::main("PS_GPRS_4MS")
P <tc3> C:\scripts\HIT_scripts\PS_EGPRS_4MS\PCU2_SSI.tel::main("PS_EGPRS_4MS")
P <tc4> C:\scripts\HIT_scripts\PS_GPRS_4_DEL\PCU2_SSI.tel::main("PS_GPRS_4_DEL")
P <tc5> C:\scripts\HIT_scripts\PS_GPRS_IDSP\PCU2_SSI.tel::main("PS_GPRS_IDSP")
P <tc6> C:\scripts\HIT_scripts\PS_EGPRS_ADD_TERR\PCU2_SSI.tel::main("PS_EGPRS_ADD_TERR")
P <tc7> C:\scripts\HIT_scripts\PS_GPRS_IDSP_TER_DGRD\PCU2_SSI.tel::main("PS_GPRS_IDSP_TER_DGRD")
P <tc8> C:\scripts\HIT_scripts\PS_TERR_FAIL_DUE_NACK\PCU2_SSI.tel::main("PS_TERR_FAIL_DUE_NACK")
P <tc9> C:\scripts\HIT_scripts\PS_TERR_FAIL_DUE_TMR_EXP\PCU2_SSI.tel::main("PS_TERR_FAIL_DUE_TMR_EXP")
P <tc10> C:\scripts\HIT_scripts\PS_2BTS_SIM_UGRD\PCU2_SSI.tel::main("PS_2BTS_SIM_UGRD")
P <tc11> C:\scripts\HIT_scripts\PS_CH_RESYNC\PCU2_SSI.tel::main("PS_CH_RESYNC")
P <finalize> C:\scripts\HIT_scripts\test_execution.tel::post_testcase_list_execution()
```

Kuva 12. Testitapauslista, johon on määritelty yhdessä testiajossa ajettavat HIT-makrot.

7 POHDINTA

Integraatiotestausjärjestelmän toteutus onnistui hyvin ja se täytti sille asetetut vaatimukset. Järjestelmä on tehostanut ja nopeuttanut integraatiotestausprosessia huomattavasti, sillä aiemmin kaikkia testauksessa käytettyjä työkaluja käytettiin manuaalisesti joko graafisen käyttöliittymän avulla tai komentoriviltä. Tällöin testaus on kuitenkin hidasta, koska käytettäviä työkaluja on paljon ja yhden testitapauksen ajon aikana tulee käyttää testiympäristön Windows – tietokonetta, Linux – tietokonetta ja testauksen kohteena olevaan paketinohjausyksikköä. Näin ollen yhden testitapauksen ajaminen saattoi kestää esimerkiksi 30 minuuttia. Tämän jälkeen täytyi vielä manuaalisesti analysoida testin tulos logitiedostoja tarkastelemalla. Nyt yhden testitapauksen ajo kestää tyypillisesti 2-3 minuuttia, jonka jälkeen saat automaattisesti testituloksen XML – muodossa, josta näkee nopeasti ja helposti mikäli joitain virheitä esiintyi.

Järjestelmä myös mahdollistaa integraatiotestauksen liittämisen osaksi Continuous Integration – järjestelmää. Tällöin testit ajetaan automaattisesti, joka yö ja käyttäjän tarvitsee ainoastaan tarkastaan testiajon tulokset. Tällaista ei olisi voinut tehdä ennen integraatiotestausjärjestelmää ja siihen tehtyä rajapintaa Jenkins – järjestelmään.

Testausjärjestelmän toteutus vaati monien työkalujen ja tekniikoiden yhdistämistä, koska GSM – verkon pakettikytkentäinen datapalvelu on todella laaja ja monimutkainen järjestelmä. Pelkästään paketinohjausyksikön ohjelmisto koostuu kymmenistä eri moduuleista, jotka sisältävät lähes miljoona riviä ohjelmakoodia. Järjestelmän toteutus vaati testausympäristöjen suunnittelua, testauksessa käytettävien työkalujen asentamista ja konfiguroimista testausympäristöihin, testauksessa käytettävän sovelluskehyksen suunnittelua ja toteuttamista. Sovelluskehyksen toteutus vaati myös uuden HIT - ohjelmointikielen opiskelua.

Ohjelmistotestauksesta puhuttaessa on myös oleellista analysoida testauksen tehokkuutta ja kattavuutta. Testauksen tehokkuus syntyy nimenomaan siitä, että testausprosessia on automatisoitu mahdollisimman pitkälle, jolloin siihen ei tarvitse kuluttaa henkilöstöresursseja. Testauksessa on myös oleellista löytää ohjelmavirheet mahdollisimman varhaisessa vaiheessa testausprosessia, jolloin ne on helpompaa ja halvempaa korjata. Testauksen kattavuus taas syntyy tarpeeksi suuresta määrästä erilaisia testitapauksia ja testi-

tapausten laadusta. Integraatiotestausjärjestelmää voidaan pitää tehokkaana järjestelmänä, koska testausprosessi on automatisoitu todella pitkälle ja loppukäyttäjän on helppo käyttää sitä. Myös testitapausten tekeminen ja lisääminen järjestelmään on helppoa.

Integraatiotestaus on ensimmäinen testivaihe, jossa ohjelmakoodia ajetaan oikeassa paketinohjausyksikössä, eli koodia suoritetaan oikealla raudalla. Pelkästään tästä syystä testivaiheessa voi tulla esiin sellaisia ohjelmistovirheitä, joita ei yksikkötestauksessa ole huomattu. Integraatiotestauksen pääasiallinen tarkoitus on kuitenkin testata PCUSIG –rajapintaa, eli sitä kuinka paketinohjauskortti toimii muiden tukiasemaohjaimen yksiköiden kanssa. Integraatiotestausjärjestelmä toimii hyvin tämän rajapinnan testaukseen. Koska järjestelmää on käytetty vain vähän aikaa eikä testauksen kattavuus ole vielä kovin suuri, on mahdotonta arvioida kuinka paljon ohjelmavirheitä tässä testivaiheessa saadaan esiin.

Usein virhetilanteet kuitenkin esiintyvät vain tietynlaisissa verkkoinfrastruktuureissa ja monen samanaikaisen toiminnan seurauksena, on tällaisia virheitä mahdotonta löytää integraatiotestausvaiheessa. Tämä johtuu siitä, että integraatiotestausympäristö ei vastaa suuria verkkorakenteita, jollaisia oikeassa maailmassa olevat GPRS- ja EDGE-verkot ovat. Tämän vuoksi integraatiotestauksen jälkeen tulee useita erilaisia systeemitestausvaiheita, joissa ohjelmiston toimintaa testataan oikeilla verkkoelementeillä ja monipuolisemmilla testausympäristöillä.

Jatkossa järjestelmää voidaan kehittää eteenpäin lisäämällä uusia ominaisuuksia esimerkiksi tulosten tarkempaa analyysiä varten. Myös testiympäristöjen hallintaa ja päivittämistä varten voidaan kehittää uusia työkaluja. Tällä hetkellä uudet testitapaukset ja mahdolliset muutokset integraatiojärjestelmään täytyy tehdä manuaalisesti jokaiseen testiympäristöön erikseen, tähän tulisi kehittää työkaluja, jolla tätä manuaalista työtä voidaan nopeuttaa. Tärkeintä kuitenkin olisi tehdä paljon uusia testitapauksia, jotta integraatiotestauksen testikattavuutta saadaan kasvatettua.

LÄHTEET

Aartoaho, J. 2011. Introduction to HIT. Nokia sisäinen dokumentti.

Abou Aun, M., Karam, J. & Peignot, E. 2002. GPRS General Packet Radio Service EDGE Enhanced Data for GSM Evolution. Nokia sisäinen dokumentti.

Bajaj, H. 2009. ETP – emulator user quide. Nokia sisäinen dokumentti.

Elovaara, K. 2007. PTI user document. Nokia sisäinen dokumentti.

Hartikainen, J. 2007. BSC SW Architecture. Nokia sisäinen dokumentti.

Kautto, T., 1997. Ohjelmistotestaus ja siinä käytettävät työkalut. Jyväskylän yliopisto. Luettu 19.11.2014. http://www.mit.jyu.fi/opiskelu/seminaarit/bak/testaus/

Kawaguchi, K. 2014. What is jenkins. Luettu 20.11.2014. https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Kumar, T. 2013. PCU introduction. Nokia sisäinen dokumentti.

Montelius, J. 2009. GPRS and EDGE. Nokia sisäinen dokumentti.

Mustajärvi, J. 2002a. Gb over IP configuration v. 02. Nokia sisäinen dokumentti.

Mustajärvi, J. 2002b. GPRS Emulator overview. Nokia sisäinen dokumentti.

Nikula, U., 2014. Integraatio- ja systeemitestaus. Lappeenrannan teknillinen yliopisto. Luettu 20.11.2014. http://www2.it.lut.fi/kurssit/CT60A0210/L12Luento.pdf

Nokia, PCU introduction. n. d. Nokia sisäinen dokumentti.

Olkku, J. 2007. Challenger basics course material. Nokia sisäinen dokumentti.

Pesola, J. 1998. CVOPS - A Tool for Portable Communication Software. Luettu 21.11.2014. http://www.ercim.eu/publication/Ercim_News/enw32/pesola.html

Poimala, S. & Tolvanen, P. Yleisimmät ketterät käytännöt. Luettu 21.11.2014. http://www.meteoriitti.com/Artikkelisarjat/Ketteryys-haltuun/Ketteryys-haltuun-Yleisimmat-ketterat-kaytannot/

Rantala, T. 1997. RAS/BSC BBSOM. Nokia sisäinen dokumentti.

Räty, E. 2014. UT Frameworks. Nokia sisäinen dokumentti.

Sollins, K. 1992. THE TFTP PROTOCOL. Luettu 22.11.2014. https://tools.ietf.org/html/rfc1350

Surana, N., Kakkar, S., Goyal, K., Gandhi, H., Jain, S., Das, S. & Malhotra, S. 2011. PCUM application high-level design. Nokia sisäinen dokumentti.

Vihavainen, A. 2011. Yksikkötestaus. Helsingin yliopisto. Luettu 9.11.2014. http://www.cs.helsinki.fi/u/avihavai/edutainment/2011/ohma/7-yksikkotestaamisesta.pdf

W3Schools. The world's largest web development site. Luettu 28.11.2014. http://www.w3schools.com/xsl/