Opinnäytetyö (AMK) / (YAMK)

Tietotekniikan koulutusohjelma

Sulautetut järjestelmät

2010

Toni Kinnunen

SULAUTETUN JÄRJESTELMÄN TESTIAUTOMATISOINTI

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

Turun ammattikorkeakoulu

Tietotekniikan koulutusohjelma | Sulautetut järjestelmät

Joulukuu 2010 | Sivumäärä 44

Ohjaaja: TkL Jari-Pekka Paalassalo

Tekijä: Toni Kinnunen

SULAUTETUN JÄRJESTELMÄN TESTIAUTOMATISOINTI

Tämän työn tarkoituksena oli rakentaa automaattinen testausympäristö Viola Systems Oy:n tuoteperheelle. Työssä käytettiin ohjelmistona Robot Framework -automatisointiohjelmistoa, jonka ympärille rakennettiin fyysisistä ja virtuaalisista järjestelmistä ympäristö, johon voitiin lisätä eri Viola Systems Oy:n laitteita testaamista varten.

Automatisoinnin pääasiallisena tarkoituksena oli auttaa tuotekehitystä huomaamaan mahdolliset fyysiset, sekä ohjelmistolliset ongelmat ajoissa. Tällöin niihin pystyttiin reagoimaan mahdollisimman nopeasti ennen tuotteen luovuttamista asiakkaan käyttöön.

Automaattitestauksen roolia ohjelmistokehityksessä käydään läpi V-mallin ja vesiputousmallin osalta. Testauksen tavoitteita, riittävyyttä, virheiden etsintää sekä automaattitestauksen kannattavuutta pohditaan yleisellä tasolla.

Testausympäristön rakenne koostui eri ohjelmistoista sekä fyysisistä laitteista. Fyysisten laitteiden, kuten palvelimen, verkkokytkimen ja UPS:n valitseminen ja asentaminen oli osa testausympäristön fyysistä rakennetta. Ohjelmistojen, kuten Xenin ja Robot Frameworkin tarkoituksena oli mahdollistaa testien suorittaminen testausympäristössä erilaisten lisäohjelmistojen avulla, jotka rakennettiin Robot Frameworkin ympärille. Robot Frameworkin avulla rakennettiin perustestejä testattaville laitteille.

ASIASANAT:

ohjelmistotestaus, testausautomatisointi, robot framework, testausympäristö, automatisointi

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Systems

December 2010| Total number of pages: 44

Instructor: Jari-Pekka Paalassalo, Lic.Tech., Principal Lecturer

Author: Toni Kinnunen

TEST AUTOMATION IN EMBEDDED SYSTEMS

The goal of this thesis was to build an automatic testing environment for the product family of Viola Systems Ltd. The system used Robot Framework automation program surrounded with physical and virtual systems. It was possible to add different Viola System Ltd products for testing.

The main purpose of automation was to help product development to detect hardware and software issues in time so that there would be time to react as quickly as possible before delivering the product to clients.

The role of the automatic testing in software developing models (V-model and waterfall model) was considered in this thesis. Testing goals, searching errors, cost-effectiveness and the amount of testing in software developing were generally considered.

The construction of the testing environment consisted of different software programs and hardware devices. The main parts of the physical construction were the server, switch and uninterrupted power supply. To be able to run different automatic tests in the testing environment, a XEN virtualization program and a Robot Framework test automation tool were installed. There was also a need to build some extra functionality around the Robot Framework. Basic automatic tests were created using the Robot Framework.

KEYWORDS:

software testing, test automation, robot framework, test environment, automation

SISÄLTÖ

SYMBOLI- JA LYHENNELUETTELO

3G	Kolmannen sukupolven matkapuhelinteknologia
EDGE	Matkapuhelinten pakettikytkentäiseen tiedonsiirtoon suunniteltu tekniikka (Enhanced Data rates for Global Evolution)
GPRS	GSM-verkossa toimiva pakettikytkentäinen tiedonsiirtopalvelu (Ge-neral Packet Radio Service)
IP	TCP/IP-mallin Internet-kerroksen protokolla (Internet Protocol)
RAID	Tekniikka, jolla tietokoneiden vikasietoisuutta ja nopeutta kasvate-taan käyttämällä useita erillisiä kiintolevyjä, jotka yhdistetään yh-deksi loogiseksi levyksi (Redundant Array of Independent Disks)
SSH	Turvalliseen tiedonsiirtoon tarkoitettu järjestelmä (secure shell)
TCP	Tietoliikenneprotokolla, jolla luodaan yhteyksiä tietokoneiden välille (Transmission Control Protocol)
TCP/IP	Usean Internet-liikennöinnissä käytettävän tietoverkkoprotokollan yhdistelmä (Transmission Control Protocol / Internet Protocol)
UNIX	Unix on laitteistoriippumaton käyttöjärjestelmä
VLAN	Virtuaalilähiverkko (Virtual Local Area Network)
VPN	Tapa, jolla kaksi tai useampia yrityksen verkkoja voidaan yhdistää julkisen verkon yli muodostaen näennäisesti yksityisen verkon (Virtual Private Network)

1 JOHDANTO

Työn tarkoituksena oli rakentaa Viola Systems Oy:lle automaattinen testausjärjestelmä, jolla voitiin suorittaa erilaisia testejä heidän tuoteperheensä laitteille. Viola Systems Oy kehittää tietoliikennelaitteita teollisuuden tarpeisiin. Tuotteiden pääasiallisena tarkoituksena on toimia sähköverkkojen etäohjaamiseen tarvittavien 3G- ja GPRS-laitteiden tuottaminen. Laitteita voidaan käyttää myös mihin tahansa tietoliikennejärjestelmään, jossa tarvitaan VPN yhteyksiä langattomasti tai langallisesti. Tuote, johon tässä työssä tehdään testauksen suunnittelua, on Arctic 3G Gateway.

Arctic 3G Gateway eli lyhennettynä A2 on laite, jonka pääasiallisena tehtävänä on välittää erilaisten tiedonsiirtoprotokollien välistä liikennettä. Laite muun muassa mahdollistaa sarjaporttiin liitettävien teollisuuslaitteiden etäohjaamisen kiinteän tai langattoman internetyhteyden avulla. Langattomia yhteyksiä varten laite tukee matkapuhelinlaitteista tutuksi tulleita tiedonsiirtoprotokollia kuten GPRS:ää, EDGEä ja 3G:tä. Laite käyttää erilaisia VPN-tekniikoita yhteyksien suojaamiseen.

Työssä käsitellään yleisesti testausta ohjelmistokehityksessä sekä automatisointia testauksen osana. Työn varsinaisessa työosassa käsitellään testausympäristön rakentamista fyysisten laitteiden hankkimisesta ja asentamisesta, toimivan testausohjelmiston rakentamiseen asti. Valmiina testiohjelmistona käytettiin Robot Frameworkia, johon erilaisten lisäsovellusten avulla rakennettiin toimiva kokonaisuus, jonka jälkeen oli mahdollisuus lisätä laitteita erilaisia testaustarkoituksia varten.

2 TYÖN RAJAUS JA TAVOITTEET

Työn rajattiin ympäristön toimintakuntoon saamiseen sekä perustestien suunnitteluun ja toteutukseen. Tavoitteena oli tehdä järjestelmä, jota voitaisiin laajentaa tarvittaessa eri tuotealustoille Arctic 3G Gatewayn lisäksi. Työstä rajattiin pois pitkänajantestaus, jolle rakennettiin vastaavanlainen ympäristö. Työssä käydään läpi ympäristön perusrakenne sekä muutamia automaattisia testejä ja niiden rakentamista.

3 TESTAAMINEN

Testaaminen on prosessi, jonka tarkoituksena on löytää virheitä suoritettavasta ohjelmasta. Tätä kutsutaan negatiiviseksi testaukseksi. Testaus on mitä tahansa toimintaa, joka on suunniteltu arvioimaan ohjelman tai järjestelmän ominaisuutta tai kykyä kohdata sille asetetut tulokset. Tätä taas kutsutaan positiiviseksi testaukseksi. Nämä kaksi testaustyyliä määrittelevät testausprosessin. Tehokkaan testauksen avainkohtana on tietysti tehdä siitä niin tehokas kuin mahdollista. Tarkka suunnittelu ja projektin hallinta ovat osa tehokkuutta, mutta testauksen automatisointia ja sen tuomia etuisuuksia kannattaa myös pohtia. [1]

Testaus koostuu useasta työvaiheesta:

- testauksen suunnittelu
- testiympäristön luonti
- testin suorittaminen
- tulosten tarkastelu.

Näihin työvaiheisiin ja niihin läheisesti liittyvään virheiden jäljitykseen ja korjaukseen kuluu tyypillisesti yli puolet ohjelmistoprojektin resursseista, joten testauksen läpivientiin parhaalla mahdollisella tavalla kannattaa kiinnittää huomiota. Testauksen määrä on aina kompromissi käytettävissä olevien resurssien ja luotettavuudesta saavutetun varmuuden välillä. [2]

3.1.1 Testauksen tavoitteet

Kuvassa 3.1 mainitaan kuusi asiaa, jotka toimivat tavoitteena testausprosessissa. Listan ensimmäiset viisi kohtaa voidaan suoraan lokeroida osana joko positiivista tai negatiivista testaustyyliä. Alimpana kohtana mainitaan loppukäyttäjälle aiheutuvat riskit. Tämä on ollut nopeasti yleistyvä tapa (vuonna 2007) määritellä testausprosessi. Määritellään riskien perusteella testattavat asiat. [1]

1. Tuote vastaa sille asetettuihin vaatimuksiin
2. Tuoda esille ohjelmiston ongelmakohdat
3. Pitää huolta ettei ohjelmisto tee mitään mitä sen ei pitäisikään tehdä
4. Kerätä luottamusta laitteen toiminta varmuudesta
5. Ymmärtää rajat, jotka ylitettäessä järjestelmän rajat ylitetään
6. Ymmärtää mahdolliset riskit, joille käyttäjät voivat altistua saadessaan järjestelmän haltuunsa

Kuva 3.1 Testauksen tavoitteet

Testaamisen tärkeys tuotekehityksessä on merkittävä, koska se on ainoa tapa varmistaa, että asiakkaalle saadaan toimiva tuote. Täydellistä tuotetta ei kuitenkaan voida saavuttaa, eli niin kattavaa testausjärjestelmää on mahdoton rakentaa, että se huomaisi kerralla kaikki mahdolliset tuotteen ongelmat ja viat. Tes-

taamisen tarkoitus on kuitenkin pienentää sitä riskiä, että laite voisi esimerkiksi aiheuttaa vaaratilanteita käyttäjälle.[1]

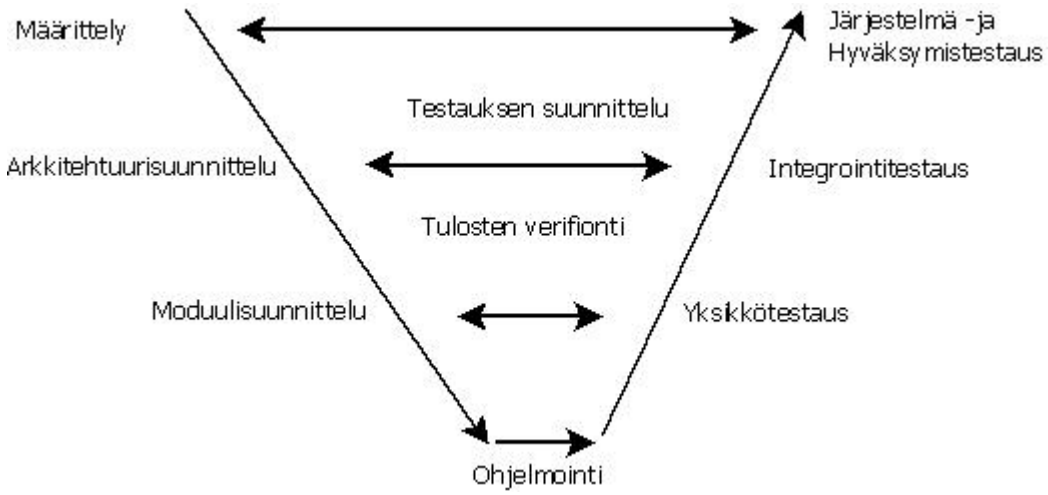
Tässä työssä käytettävä Arctic 3G Gateway ei viallisena laitteena aiheuta suoraan käyttäjälle hengenvaaraa, mutta kustannuksia siitä asiakkaalle ja Viola Systems Oy:lle voi aiheutua. Laite on tarkoitettu etäohjattavaksi, eli laitteen sijainti voi monestikin olla kaukana ja vaikeapääsyisessä maastossa. Tämän vuoksi laitteen luotettavuuden kannalta on tärkeää, että laite osaa mahdollisen vikatilanteen sattuessa palautua normaaliin toimintakuntoon. Oli erittäin tärkeää rakentaa Arctic 3G Gatewaylle järjestelmä, joka huomaa ennalta mahdolliset ongelmatapaukset, jotka laitteelle voivat tapahtua sen ollessa kentällä useita kuukausia tai jopa vuosia. Lisäksi nopeatempoisessa tuotekehityksessä on tärkeää havaita mahdolliset virheet mahdollisimman tehokkaasti.

3.2 Ohjelmistokehitysmallit testauksessa

Ohjelmistokehitysmallin päämääränä on tuottaa hyvälaatuista ohjelmistoa nopeasti. Kyseinen päämäärä toistuu kaikissa ohjelmistonkehitysmalleissa [3]. Tässä luvussa käydään läpi testauksen osuutta V-mallissa sekä vesiputousmallissa. V-mallissa testaus suoritetaan tasoittain yhdessä suunnittelun kanssa, kun taas vesiputousmallissa suoritetaan testaus yhtenä kokonaisuutena.

3.2.1 Ohjelmistokehityksen V-malli

V-mallissa testauksen suunnittelu tapahtuu testaustasoa vastaavalla suunnittelutasolla (kuva 3.2). Ohjelmiston kehitysvaiheessa korostuu yksikkötestaus, jossa testataan ohjelmistoa ohjelmarivitasolla. Testaus voi käsittää esimerkiksi yhden ohjelmaluokan testaamisen. Seuraavana testausvaiheena tulee integraationtestaus, joka käsittää yksikkötestauksessa testattujen ohjelmaosien yhdistämisen jälkeisen testauksen eli testataan, kuinka yksiköt toimivat yhdessä. [4]

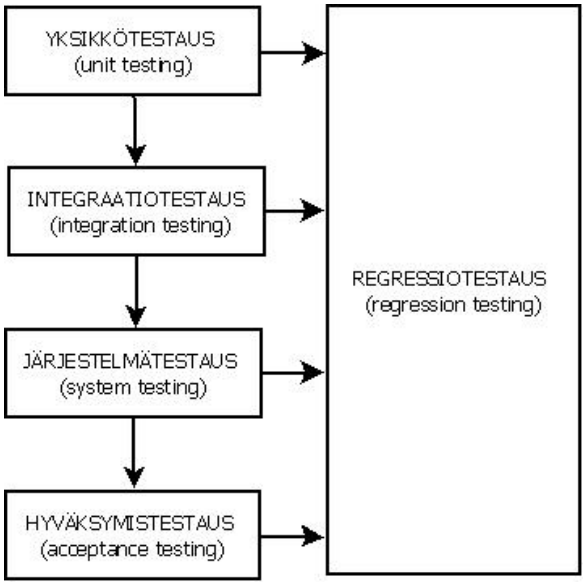


Kuva 3.2 Ohjelmistokehityksen V-malli

Järjestelmätestaus toimii suurimpana kokonaisuutena ja vastaa järjestelmän toiminnallisuuden testaamisesta. Järjestelmätestaus keskittyy ongelmiin, jotka ilmaantuvat integraation korkeimmalla tasolla. Tyypillisesti järjestelmätestaus käsittää useita testaustyyppejä, kuten toiminnallisuuden, käytettävyyden, turvallisuuden, luetettavuuden, palautumisen ja tehokkuuden testaamisen. [4]

Järjestelmätestauksen jälkeen suoritetaan hyväksymistestaus. Hyväksymistestauksella varmistetaan, että järjestelmä vastaa asiakkaan vaatimuksia sekä an-

taa luottoa sen toimivuudesta, ennen kuin järjestelmä luovutetaan asiakkaan käyttöön. Asiakkaan näkökulmasta haluttaisiin mahdollisimman perusteellinen hyväksymistestaus, joka lähentelee järjestelmätestausta, mutta myyjän näkökulmasta testaus suoritetaan siihen pisteeseen saakka, kunnes tuotteesta saadaan tuottoa. [4]



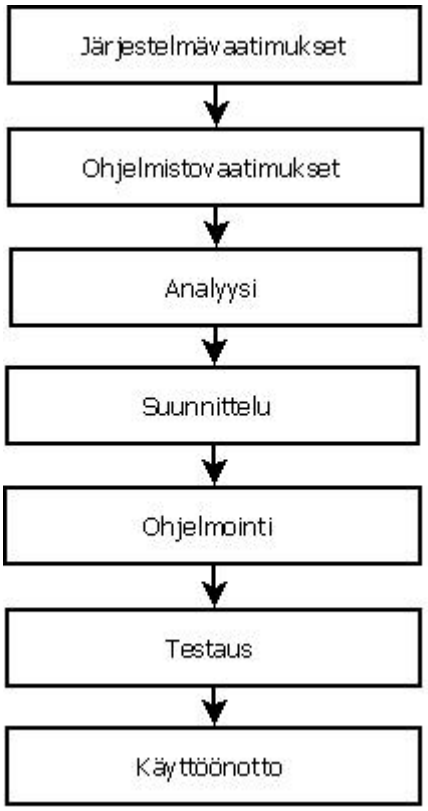
Kuva 3.3 Testaustasot [1]

Regressiotestaus ei varsinaisesti ole testaustaso, kuten aikaisemmin mainitut, mutta se on testaustekniikka, jota hyödynnetään muissa testaustasoissa (kuva 3.3). Asiakkaalta saatujen lisävaatimusten tai järjestelmässä havaittujen vikojen korjaamisen jälkeen tulee järjestelmälle ajaa regressiotestausta. Regressiontestauksella varmistetaan tuotteen toimivuus myös sen jälkeen, kun uusia ominaisuuksia on lisätty tai vanhoja ominaisuuksia on korjattu. Regressiotestauksen tulee olla mahdollisimman tehokas, joten testimoduulien täytyy olla uudelleen käytettäviä. [1]

3.2.2 Ohjelmistokehityksen vesiputousmalli

Vesiputousmalli on vaiheellinen ohjelmistotuotantoprosessi, jossa suunnittelu- ja toteutusprosessi etenee vaihe vaiheelta alaspäin kuin vesiputouksessa (Kuva 3.4). Vesiputousmalli vaatii jokaisen vaiheen suoritusta loppuun, ennen kuin seuraava voidaan aloittaa. Huolellinen suunnittelu ohjelmiston elinkaaren alussa, todennäköisesti johtaa merkittäviin säästöihin projektin myöhemmissä vaiheissa. Virhe, joka löydetään varhaisessa vaiheessa, on rahallisesti, työmäärällisesti ja ajallisesti halvempi korjata, kuin jos sama virhe olisi havaittu myöhemmässä vaiheessa. Toisaalta heikkoutena on sen riippuvaisuus muista vaiheista eli todennäköisesti jokainen vaihe ei mene läpi niin kuin suunniteltiin, jolloin tuote viivästyy, eikä saavuta viimeistä vaihetta aikataulussa. [5][6]

Vesiputousmallin testausvaiheessa suoritetaan vastaavanlaisessa järjestyksessä ohjelmiston testaus kuin V-mallissa: yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. [6]



Kuva 3.4 Vesiputousmalli

3.3 Testauksen suorittaminen

Testaus tapahtuu usein kokeilemalla ohjelmaa umpimähkäisesti jollain syöttöaineistolla. Varsinkin jos testaajana on ohjelman tekijä, tavoitteena on ennemminkin osoittaa ohjelman toimivuus kuin virheiden etsintä. Testauksen määrä ei välttämättä ole sama kuin testauksen tehokkuus eli muutamien tuntien huolellisesti suunniteltujen testitapausjoukkojen suorittaminen testattavalle ohjelmistolle voi olla tehokkaampaa, kuin päiväkausien umpimähkäinen kokeilu. [2]

Usein testattavien syötteiden määrä tekee testauksesta mahdottoman, jos ajatellaan kaiken kattavaa testausta. Lisäksi, varsinkin reaaliaikajärjestelmissä, laitteen sen hetkinen sisäinen tila voi vaikuttaa testitulokseen eriävästi aikai-

sempiin testauskertoihin verrattuna. Reaaliaika järjestelmien testauksen suurin ongelma onkin se, että virhetilanteiden toistaminen niiden syiden tutkimiseksi on joskus tavattoman vaikeaa. [2]

Ohjelman testauksessa pystytään kattamaan vain häviävän pieni murto-osa kaikista mahdollisista tilanteista, ja reaaliaikajärjestelmien tapauksessa testauksen ”todistusvoima” on vieläkin vähäisempi. Tämän takia ohjelman toimivuuteen ei kannata liian paljon luottaa hyvistä testaustuloksista huolimatta. Reaaliaikajärjestelmien testauksen ongelmallisuudesta johtuen formaalien verifiointimenetelmien merkitys tulee tulevaisuudessa lisääntymään. [2]

3.4 Virheiden etsintä

Testauksen yhteydessä virhe (englanniksi error, mistake, bug) on poikkeama spesifikaatiosta. Spesifikaation puuttuessa ei voida testausta suorittaa, koska ei voida todeta lopputuloksen oikeellisuutta. Tavallisimmin testauksessa käytettäviä spesifikaatioita ovat toiminnallinen määrittely ja tekninen määrittely. Käytännössä suurin ongelma on spesifikaation puutteellisuus, jolloin ristiriitatilannetta ei voida ratkaista spesifikaatiota tutkimalla. [2]

Virheen vakavuus voi vaihdella järjestelmän käytön kokonaan estävästä virheestä käyttäjää ärsyttävään yksityiskohtaan. Ohjelmissa arvioidaan yleensä olevan ohjelmoinnin jälkeen yksi virhe muutamaa kymmentä riviä kohden. Lisäksi noin viisi prosenttia ohjelmavirheistä jää kokonaan huomaamatta. [2]

3.5 Testauksen riittävyyden määrittäminen

Testauksen määrää on vaikea arvioida. Varsinkin järjestelmätestauksessa testausta voidaan jatkaa ”kunnes aika ja rahat loppuvat”. Tuotekehitystyössä testauksen lopettaminen on usein kompromissi tuotteessa olevien vikojen aiheuttamien kustannusten ja markkinoilta myöhästymisen aiheuttaman tuoton menetyksen välillä. [2]

Testauksen lopettamiselle tulisi asettaa hyväksymiskriteerit, jotka määritellään testaussuunnitelmassa. Hyväksymispiste voi olla esimerkiksi tilanne, jossa korjattujen ja löydettyjen virheiden suhde tasaantuu, eli ei löydetä enää uusia virheitä samaan tahtiin kuin vanhoja virheitä korjataan. Toisaalta tällä tavalla ei voida määrittää projektin lopetusajankohtaa, joka taas on yleensä lyöty lukkoon. [2]

3.6 Automaattitestaus

Ohjelmiston testaus ja ylläpito on tullut maksamaan 40–60 % ohjelmistoprojektien budjeteista, viime vuosikymmenten aikana. Kovan kilpailun takia yritykset yrittävät lyhentää mahdollisimman paljon ohjelmistoprojektin aikatauluja ja näin ollen myös lyhentää testaukseen laitettavaa aikaa ja rahaa. Automaattitestaus on yksi tehokas ratkaisu ongelmaan. [7]

Ennen kuin automaattitesteihin kannattaa laittaa rahaa ja vaivaa, yrityksen insinööreillä ja johtajilla, täytyy olla hyvä käsitys automaattitestausjärjestelmistä. Automaattitestausjärjestelmien ylläpitoon kannattaa varata erillinen työryhmä, joka vastaa suunnittelusta, implementoinnista ja järjestelmän kehityksestä, yh-

dessä ohjelmistokehittäjien ja testaus insinöörien kanssa. Lisäksi ryhmä vastaa automatisoinnin kouluttamisesta. Ilman erillistä testausryhmää, automaattijärjestelmien ylläpito ja kehitys voi tulla kalliiksi. [7]

Johtajien ja insinöörien sitoutuminen testausautomatisointiin on tärkeää. Jotta heillä olisi tietoisuus automatisoinnin menoista, heillä täytyy olla hyvä tietoisuus automaattijärjestelmien monimutkaisuudesta. Lisäksi heidän täytyy osata tehdä hyviä projektisuunnitelmia, jossa otetaan myös automatisointi huomioon. Automaattijärjestelmien ylläpito- ja kehityskuluista tulee olla tietoinen. Automatisointi maksaa, mutta se voi lyhentää projektin aikataulua, parantaa testauslaatua ja vähentää miestyötunteja. Tämän vuoksi kannattaa suunnitella automatisoinnille budjetti, aikataulu ja yleisestikin suunnitelma projektin läpi viemiseksi, ennen kuin alkaa järjestelmää rakentaa. [7]

Automatisointiin ei kannata siirtyä, kun sille ei ole tarpeeksi aikaa. Jos tuotteen kehitys on niin pitkällä jo, ettei kunnon suunnitelmaa ehditä tekemään, on automatisointi myöhäistä ja ei-kannattavaa. Epärealistiset tavoitteet ja odotukset kannatta unohtaa, koska silloin tulee tehtyä mahdottomia testisuunnitelmia, joiden läpi vieminen tulee olemaan mahdotonta. [7]

Automaattitestit mahdollistavat testien uudelleenkäytettävyyden ja näin ollen testien ajaminen voidaan suorittaa useaan otteeseen tuotekehityksen aikana. Toisin sanoen automatisointia voidaan hyödyntää erityisen paljon regressiontestauksessa. Tällä tavalla säästetään miestyötunteja sekä voidaan verrata automaattitestien suorituksia aikaisempiin suorituksiin. Testien automatisointia kannattaa harkita, kun

- testataan sovelluksia, joilla on nopea käännös ja julkaisu sykli
- testataan sovelluksia, jotka toimitetaan useille alustoille
- testataan sovelluksia, joissa on monimutkaisia käyttöliittymiä

- testataan sovelluksia, jotka vaativat perinpohjaisen, täsmällisen ja toistettavan testauksen
- on tarve pienentää testausaikataulua sekä vähentää taustauksen määrää.

Automaattitestien hyöty korostuu varsinkin suoritus, kuormitus, volyymi- ja stressitesteissä. Lisäksi testien ajaminen ei ole riippuvainen suorituksen testaajasta, vaan testien suoritus voi tapahtua kellon ympäri seitsemänä päivänä viikossa. Useiden testilaitteiden käyttäminen samaan aikaan tekee testauksesta erittäin tehokkaan. [1]

4 KÄYTETYT OHJELMISTOT

Seuraavissa luvuissa kerrotaan ohjelmista, joita asennettiin testausympäristöä rakennettaessa. Ensisijaisena ohjelmistoina olivat virtuaalisointiohjelmat sekä testien ajamiseen tarvittava Robot Framework. Lisäksi työssä kerrotaan hieman myös Ubuntu käyttöjärjestelmästä, jota käytetään pääsääntöisesti virtuaalikoneina, mutta myös palvelimen käyttöjärjestelmänä.

4.1 Xen-virtualisointiohjelmisto

Virtuaalisointiohjelmistoja löytyy markkinoilta kaupallisina ja ei-kaupallisina versioina. Suurimpana ja tunnetuimpana virtuaalisointiohjelmistojen kehittäjänä tunnetaan VMware, joka vuonna 1999 julkaisi ensimmäisen virtuaalisointiohjelmistoversionsa ”VMware Workstation”. Kyseinen tuote saavutti suuren suosion, jonka vuoksi vuonna 2001 julkaistiin ”VMware GSX Server” ja ”VMware ESX Server” sekä vuonna 2003 ”VirtualCenter” ja ”VMotion”. VMware on säilyttänyt asemansa markkinajohtajana, vaikka haastajia on tullut muun muassa Microsoftilta ja vapaaseen lähdekoodiin perustuvan Xenin osalta. [10]

Mextin vuonna 2009 teettämän tutkimuksen mukaan suomalaisissa yrityksissä suurimman suosion virtuaalisointiohjelmistoista on saanut VMware ESX. Tutkimuksessa kysyttiin käytettyjä palvelinvirtuaalituotteita VMware ESX:n saadessa noin 80 %:n osuuden. Tässä työssä käytettiin ilmaista Xen-virtuaalisointiohjelmistoa, jonka osuus käytetyistä virtuaalisointiohjelmistoista oli kolme prosenttia. [12]

Xen-virtuaalisointiohjelmisto on Cambridgen yliopistossa kehitetty ilmainen ja vapaa (GPL) virtuaalikone IA-32, IA-64 ja PowerPC-arkkitehtuureille, joka mahdollistaa useiden käyttöjärjestelmien ajamisen samalla tietokoneella [9]. Xenillä voidaan ajaa useita tunnettuja käyttöjärjestelmiä kuten Linuxia, NetBSD:tä, FreeBSD:tä, Solarista ja Windowsia. [13]

Xenin toiminta perustuu kolmeen pääkomponenttiin: Xen Hypervisoriin, Domain 0:aan (Dom0) ja Multiple DomainU:n (DomU). Xen Hypervisor toimii rajapintana raudan (CPU, I/O, kovalevy) ja virtuaalisoidun käyttöjärjestelmän välillä, välittäen erityyppisiä pyyntöjä näiden välillä. Tämä mahdollistaa useiden käyttöjärjestelmien ajamisen turvallisesti ja itsenäisesti. Dom0 käynnistetään Xen Hypervisorin toimesta, ja sitä voi hyödyntää kaikki muut käyttöjärjestelmät paitsi Windows. Dom0:lla on oikeudet Xen Hypervisoriin, jonka kautta erilaisten toimintojen, kuten virtuaalikoneiden käynnistäminen ja sammuttaminen, on mahdollista. DomU:lla tarkoitetaan virtuaalikonetta, jota Dom0:lla ohjaa. Virtuaalikoneet voivat olla joko täysvirtuaalisia tai paravirtuaalisia. [13]

Tässä työssä asennettiin Xen 3.3.0 -virtuaalisointiohjelmisto fyysiselle palvelimelle. Sen käytöstä oli kokemuksia ennestään ja Viola Systems Oy:ssä sitä käytettiin myös yleisesti virtuaalisointiohjelmistona.

4.2 Robot Framework

Robot Framework mahdollistaa helpon ja yhtenäisen tabular-syntaksiin perustuvan testitapauksien rakentamisen. Se mahdollistaa uudelleenkäytettävien korkean tason avainsanojen luomisen olemassa olevista avainsanoista. Yhdistelemällä saadaan koottua laajoja avainsanasarjoja, joista testit rakentuvat. Raportointi perustuu helppolukuiseen HTML – formaattiin. Omien testikirjastojen luomiseen on olemassa API-kirjasto, johon voidaan toteuttaa testikirjastoja java- tai python-ohjelmointikielillä. Robot Framework muun muassa sisältää selainoh-

jaukseen tarkoitetun Selenium-kirjaston, SSH- ja Telnet-kirjastot sekä monia muita kirjastoja, jotka mainitaan työn myöhemmässä vaiheessa. [14]

Testausautomatisointiin tarvittiin sovellus, joka mahdollistaisi erilaisten testien toteuttamisen. Testien toteuttamisella tarkoitetaan laitteen tai järjestelmässä olevan virtuaalikoneen ohjaamista ja tutkimista telnet tai ssh -yhteyden avulla. Esimerkiksi laitteen lokitiedostoa voidaan käydä lukemassa tai avata palomuurista reittejä. Robot Framework soveltui hyvin tähän tarkoitukseen sen kirjastolaajuuden vuoksi. Lisäksi Robot Framework mahdollistaa testitulosten lukemisen web-pohjaiselta käyttöliittymältä.

Työn kannalta tärkeää oli saada ympäristö mahdollisimman nopeasti toimintakuntoon, jotta testien rakentaminen pystyttiin aloittamaan ilman, että kulutettiin aikaa oman komento- ja raportointiohjelmiston rakentamiseen. Testien toteuttamisesta ja suunnittelemisesta Robot Frameworkilla tullaan kertomaan työn myöhemmässä vaiheessa. Seuraavassa käydään yleisesti läpi Robot Frameworkin toimintaperiaatetta ja kirjastoja.

4.2.1 Robot Frameworkin kirjastot

Robot Frameworkin toiminta perustuu sen kirjastojen käyttöön. Kirjastot (Taulukko 4.1) sisältävät erilaisia komentoja eri osa-alueille, joita yhdistelemällä saadaan aikaiseksi erityyppisiä testikomponentteja. Kirjastot jaetaan standardi- ja ulkoiseen kirjastoon. Standardikirjastolla tarkoitettaan kirjastoja, jotka ovat automaattisesti asennettu Robot Frameworkin mukana. Kyseiset kirjastot pitää esitellä jokaisessa testitiedostossa, joissa niitä käytetään, poikkeuksena Builtln -kirjasto. Ulkoisilla kirjastoilla tarkoitetaan kirjastoja, jotka pitää asentaa erikseen. Kyseiset kirjastot voivat olla ulkoisen henkilön tai ryhmän tekemiä. [15]

Standardit kirjastot	Ulkoiset kirjastot
BuiltIn	SeleniumLibrary
OperatingSystem	SwingLibrary
Telnet	AutoItLibrary
Collections	DatabaseLibrary
String	SSHLibrary
Dialogs	
Screenshot	
Remote	

Taulukko 4.1 Robot Frameworkin kirjastot

Työssä pääasiallisesti käytettyjä kirjastoja ovat BuiltIn, OperatingSystem, Telnet, String, SeleniumLibrary ja SSHLibrary. BuiltIn-kirjastoa hyödynnetään hyvin paljon, koska sillä voidaan esimerkiksi todeta, onko haluttu tulos tosi tai epätosi, ja sen mukaan ajaa sille haluttu operaatio. Lisäksi BuiltIn-kirjastolla voidaan suorittaa tyyppimuunnoksia ja luoda muun muassa testin aikana tarvittavia muuttujia. [15]

OperatingSystem-kirjasto sisältää erilaisia toimintoja, joita ajetaan paikallisella koneella, eli koneella, johon Robot Framework on asennettu. Kirjastolla voitiin suorittaa unix-komentoja paikallisesti. [15]

Telnet-kirjastolla voidaan muodostaa yhteys telnet-palvelimeen, joka mahdollistaa kyseisen telnet-palvelimen omaavan laitteen käskyttämisen. Työssä hyödynnettiin telnet-kirjastoa muun muassa Arctic 3G Gatewayn sarjaporttitesteissä, jossa otettiin telnet-yhteys kyseisen laitteen telnet-palvelimeen. [15]

SSHLibary-kirjaston toimintaperiaate on hyvin samanlainen kuin telnet-kirjaston. SSHLibararylla voidaan lisäksi siirtää tiedostoja testattavaan laitteeseen ja laitteesta pois. [15]

String–kirjastolla voidaan manipuloida merkkijonoja sekä vertailla testistä saatua merkkijonoa odotettuun merkkijonoon. Lisäksi String-kirjastolla voidaan käydä läpi tekstitiedostoja, sekä manipuloida tai tarkastella niitä. [15]

SeleniumLibrary-kirjasto perustuu suosittuun Selenium-web-testaustyökaluun. Kyseisen työkalun kehitti vuonna 2004 chicagolainen Jason Huggins ja sillä voidaan suorittaa käskyjonoja, joita paikallinen selain suorittaa halutun laitteen web-käyttöliittymään [16]. Tässä työssä jokaiseen laitteeseen, jonka www-käyttöliittymää haluttiin ohjattavan, asennettiin virtuaalikone, johon asennettiin selenium-palvelin. Robot Frameworkin ja SeleniumLibraryn avulla lähetettiin käskyjä selenium-palvelimelle, joka suoritti komennot selaimessa ja näin ollen ohjasi testattavaa laitetta. Näin muutettiin testattavan laitteen asetuksia, testattiin sen web-käyttöliittymää sekä haettiin tietoa laitteen tilasta. Yhdeltä selenium-palvelimelta voitaisiin myös ohjata useamman laitteen web-käyttöliittymää, mutta tässä laiteympäristössä jouduttiin asentamaan jokaiselle testattavalle laitteelle oma lähiverkossa oleva virtuaalikone, jonka vuoksi samalla asennettiin myös kyseisille testilaitteille oma selenium-palvelin. [15]

4.2.2 Toimintaperiaate

Robot Frameworkia voidaan käyttää joko UNIX- tai Windows-käyttöjärjestelmissä. Robot Frameworkin toiminta vaatii, että käyttöjärjestelmään on asennettu joko Python-kirjasto tai Javaan perustuva Jython- (myös tunnettu nimeltä JPython) kirjasto. Tässä työssä käytettiin Unix-käyttöjärjestelmää, koska sen ympärille oli helpompi rakentaa lisätoiminnallisuutta. Lisäksi valittiin python, koska se oli valmiiksi asennettu Linux-käyttöjärjestelmään.

Testin ajaminen käynnistyy komentoriviltä antamalla käskyn ”pybot” sekä antamalla testitiedoston nimen tai hakemiston, jossa testitiedostot ovat.

$ pybot testi.tsv
$ pybot hakemisto/

Pybot käynnistää varsinaisen testiohjelman ”runner.py”, jolle voidaan antaa erilaisia komentorivioptioita, kuten raportointihakemisto, halutut raportit, muuttujatiedostot ja monia muita optiota, jotka saadaan esille komennolla ”pybot –help”.

$ pybot --outputdir raportit/ --report raportti.html –log loki.html –variablefile muuttujapy testi.tsv
$ pybot --outputdir raportit/ –variable MUUTTUJA:ARVO testi.tsv

Jos komentorivillä määritelty hakemisto sisältää useita testitiedostoja, suoritetaan testit aakkosjärjestyksessä. Tämän vuoksi on hyvä nimetä testitiedostot liittämällä tiedostonimen alkuun järjestysnumero.

01__testi.tsv 02__testi2.tsv 03__testi.tsv

4.2.3 Esimerkkitestin rakenne

Testin rakenne voi perustua HTML- (lyhennetty sanoista hypertext markup language), TSV- (lyhennetty sanoista Tab Separated Values) tai reST -formaattiin (lyhennetty sanasta reStructuredText). Tässä työssä käytettiin pääsääntöisesti TSV-formaattia.

Tässä luvussa käydään läpi automaattitestin rakenne demonstroimalla esimerkki yksinkertaisesta testistä, joka on rakennettu Robot Frameworkin ymmärtämään muotoon. Esimerkkitestissä (uptime.tsv) haetaan testattavan laitteen aika ja verrataan sitä aikaisemmin haettuun aikaan, jolloin huomataan, onko laite käynnistynyt uudestaan (Kuva 4.1). Esimerkki on hyvin pelkistetty, eikä kaikkia testivaiheita käydä läpi.

Automaattitestit rakennetaan avainsanoista (keyword). Avainsanat voivat olla valmiiden kirjastojen avainsanoja tai käyttäjän omia. Kuvan 4.1 esimerkissä, testin nimenä on ”Hae Aika”. Testissä tarkistetaan aluksi laitteen alusta (asetettu laitteen muuttujatiedostossa). Laitteessa yksi (Laite 1) on käytössä ssh-palvelin ja laitteessa kaksi (Laite 2) telnet-palvelin, joten molemmilla laitteilla on omat avainsanansa ”Hae Laite 1 Aika” sekä ”Hae Laite 2 Aika”. Haettuaan uuden ajan laitteelta palataan ”Hae Aika”-avainsanaan ja käynnistetään uusi avainsana ”Vertaa Aikoja”, joka tarkistaa, onko laite käynnistynyt uudestaan.

Testitiedostossa on lisäksi määritelty asetus- (setting) ja muuttuja- (variable) kentät. Muuttujakentässä voidaan alustaa muuttujia, joita voidaan käyttää hyödyksi testitiedoston avainsanoissa. Asetuskentässä voidaan määritellä muun muassa kirjastot, joista avainsanat muodostuvat. Toimivampi käytäntö oli kuitenkin esitellä resource.tsv-tiedosto, joka sisältää kyseiset määritykset. Muuttamalla resource-tiedoston asetuksia voidaan keskitetysti vaikuttaa kaikkiin testitiedostoihin samalla kertaa. Resource.tsv -tiedostoon voidaan lisäksi laittaa avainsanoja, joita voidaan hyödyntää muissakin testeissä.

uptime.tsv

Setting

Resource resource.tsv

Variable

${IP} 192.168.11.1

Test Case

Testin nimi	Testi	Argumentti	Argumentti
Hae Aika	Run Keyword If	'${ALUSTA}' == 'LAITE1'	Hae Laite 1 Aika
	Run Keyword If	'${ALUSTA}' == 'LAITE2'	Hae Laite 2 aika
	Vertaa Aikoja		

Keyword

Testin nimi	Testi	Argumentti	Argumentti	Argumentti
Hae Laite 1 Aika	SSHLibrary.Open Connection	${IP}		
	SSHLibrary.Login	kayttaja	salasana	
	…			

Testin nimi	Testi	Argumentti	Argumentti	Argumentti
Hae Laite 2 Aika	Telnet.Open Connection	${IP}		
	Telnet.Login	kayttaja	salasana	
	…			

Testin nimi	Testi	Argumentti	Argumentti	Argumentti
Vertaa Aikoja	…			

Kuva 4.1 Esimerkkitestin rakenne

4.2.4 Raportointi

Testien raportoinnin tarkoituksena on auttaa käyttäjää selvittämään testeissä ilmenneet ongelmat. Tämän vuoksi raportin tulee olla mahdollisimman laaja ja sen tulee ottaa huomioon useita näkökulmia, jolloin ongelman ytimeen päästään mahdollisimman nopeasti. Robot Framework tarjoaa xml-pohjaisen raportointityökalun, josta voidaan helposti web-selaimen avulla nähdä testien tulokset sekä suoritusajat. Robot Frameworkin raportointi perustuu loki-, raportti- ja yhteenvetotiedostoihin. [14]

Yhteenvetotiedostossa voidaan tarkastella pelkistetysti testien tuloksia, joten se on hyvin kätevä silloin, kun suuria määriä testitapauksia suoritetaan yhdellä kerralla. Robot Frameworkissa kyseinen raportoinnin osa on oletuksena poissa käytöstä. Tässä työssä yhteenvetotiedostolle ei ollut tarvetta, koska suoritettavat testit ovat kohtuullisen lyhyitä. [14]

Raporttitiedostolla voidaan tiivistää testisarjan osat yhdeksi kootuksi listaksi, jolloin nähdään kunkin osan testitulokset. Erona yhteenvetotiedostoon on se, että raportointitiedostossa nähdään myös testikohtaiset tulokset. Raporttitiedosto sisältää linkityksen lokitiedostoon, josta voidaan tarkemmin seurata testin kulkua ja nähdä testin suoritus muuttujatasolla. Tämä helpottaa testien rakennusvaiheessa ilmenevien ongelmien havaitsemista sekä korjaamista. [14]

Robot Frameworkin tarjoaman raportoinnin lisäksi tekstitiedostojen käyttötiedon tallennuksessa on hyvin toimiva tapa silloin, kun halutaan koostaa useiden testisarjojen tietoja. Esimerkiksi tekstitiedostoihin voidaan tallentaa. Kyseisiä arvoja voidaan verrata aikaisempiin arvoihin tai piirtää niistä erilaisia tilastoja kuvaajilla.

4.3 Ubuntu-käyttöjärjestelmä

Ubuntu on vapaista ohjelmistoista (avoimesta lähdekoodista) koostuva Linux-käyttöjärjestelmä, joka rakentuu Debian-projektin tekemälle työlle. Ubuntusta julkaistaan uusi versio säännöllisesti 6 kuukauden välein. Tämä takaa että käytössäsi ovat aina viimeisimmät ja parhaat avoimen lähdekoodin ohjelmat. Jokaista julkaistua versiota tuetaan vähintään 18 kuukautta. Päivitykset uusiin versioihin ovat ja tulevat aina olemaan maksuttomia. [20]

5 FYYSISET LAITTEET

Fyysisillä laitteilla tarkoitetaan järjestelmän rakenteellista kokonaisuutta. Järjestelmän perustana on palvelin, jossa virtuaalisointiohjelmistolla pääasiallisesti rakennettu testausympäristö toimii. Palvelimen valintaperusteena oli sen tehokkuus, koska sen on jaksettava ylläpitää kymmeniä virtuaalikoneita. Testilaitteiden liittämiseksi osaksi testausympäristöä tarvittiin verkkokytkin. Verkkokytkimen valintaperusteena oli porttien lukumäärä ja virtuaalilähiverkkojen (VLAN) tuki, josta kerrotaan lisää työn myöhemmässä vaiheessa. Järjestelmää rakennettaessa piti myös ottaa huomioon mahdolliset sähkökatkokset. Mahdollisten sähkökatkoksien vuoksi palvelin ja kytkin kytkettiin UPS:ään, josta myös kerrotaan lisää myöhemmin tässä luvussa.

5.1 Palvelin

Tietoliikenteen yhteydessä palvelimelle tarkoitetaan tietokoneessa suoritettavaa palvelinohjelmistoa sekä tällaista ohjelmistoa suorittavaa tietokonetta. Palve-

linohjelmistojen tehtävänä on tarjota erilaisia palveluja muille ohjelmille, joko tietokoneverkon välityksellä tai paikallisesti samassa tietokoneessa. [8]

Palvelimeksi valittiin Viola Systems Oy:n tuoteperheeseen kuuluva M2M Gateway Enterprise Edition, joka sopeutui hyvin kyseiseen tehtävään jo pelkästään sen takia, että se oli yrityksen oma tuote (Kuva 5.1).

	M2M Gateway Enterprise Edition
Processor Environment	2 GHz Quad-Core Xeon
Memor y	16 Gb RAM 2x72 Gbytes SAS hot swap
Power	Dual power supply (100-240 VAC)
Casing	Metal 19” Rack Mountable (1U)
Operating Environment	Operating temperature 0 to +45 °C Storage temperature –20 to +45 °C Humidity 10% to 90% RH non-cond.
Network Connection	2 x Ethernet RJ-45 (10/100/1000 Base-T)

Kuva 5.1 Palvelinkone

Palvelinkoneen varmatoimisuutta edesauttaa kaksoisvirtalähde, jonka vuoksi palvelinkone ei sammu, vaikka toinen virtalähteistä hajoaisi. Koneessa on rautapohjainen RAID-ohjain, joka kirjoittaa kahdelle kiintolevylle samanaikaisesti. Toisen kiintolevyn mahdollisesti hajotessa, järjestelmä jatkaa toimintaa toisella kiintolevyllä ja näin ollen turvaa tietojen säilymisen ja järjestelmän toiminnan jatkumisen. Palvelimen käyttöjärjestelmäksi valittiin Ubuntu 9.04 64-bittinen palvelinversio. Käyttöjärjestelmänä Ubuntu oli ennestään tuttu ja 64-bittinen versio vaadittiin, jotta yhteensopivuusongelmilta fyysisen muistin, käyttöjärjestelmän ja virtuaalisointiohjelmiston kannalta vältyttiin.

5.2 Verkkokytkin

Verkkokytkimellä tarkoitetaan laitetta, joka yhdistää saman verkon laitteita keskenään mahdollistaen laitteiden välisen keskusteluyhteyden. Tärkeää verkkokytkimen valinnassa oli porttien lukumäärä ja virtuaaliverkkotuki. Laitteiden vaatiessa useita verkkoliitäntöjä valittiin kaksi 24-porttista HP:n ”ProCurve 2510-24” verkkokytkintä, joiden avulla ympäristöä voidaan laajentaa myös jatkossa useammille laitealustoille.

5.3 UPS

UPS (lyhennetty sanoista Uninterruptible Power Supply) on järjestelmä tai laite, jonka tehtävä on taata tasainen virransyöttö lyhyissä katkoksissa ja syöttöjännitteen epätasaisuuksissa. UPS:ään kytkettiin virtajohto palvelinten ja kytkimien virtalähteistä. [9]

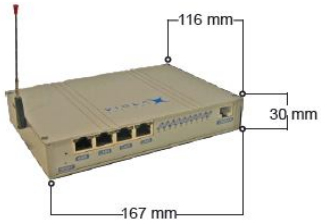
UPS:n asentaminen järjestelmään oli välttämätön mahdollisten sähkökatkoksien vuoksi. Ilman UPS:ää järjestelmän palvelin sammuisi sähkökatkoksessa ja näin ollen aiheuttaisi testien keskeytymisen. Pitkänajantestien kannalta oli tärkeää, että virran saanti on jatkuvaa, jotta voidaan seurata laitteen toimintaa ilman ulkoisia keskeytyksiä. UPS:n valintaan vaikutti sen tehokkuus: 1 kW:N saanti sähkökatkoksen sattuessa laskettiin riittävän palvelimelle, kytkimille sekä testilaitteille.

Kuva 5.2 Kuvassa ylempänä UPS ja palvelin alempana

5.4 Arctic 3G Gateway

Arctic 3G Gateway (kuva 5.3) eli tuotekehityksessä lyhennettynä A2, toimi työssä testattavana laitteena. A2:n on Linux-reititinlaite, jossa langattomien ja kiinteiden yhteyksien lisäksi on sarjaporttiliitännät eri sarjaliikenneprotokollia varten. Langattomina yhteyksinä toimii 3G- ja GPRS-yhteydet.

Arctic 3G Gatewayn hallinnointi tapahtuu SSH:n, Telnetin tai selaimen välityksellä. Pääasiallisena tehtänä laitteella on muodostaa salattu yhteys palvelimeen, jolloin asikkaan valvomo saa yhteyden A2:n sarjaportissa tai Lan-portissa olevaan laitteeseen.



Kuva 5.3 Arctic 3G Gateway

6 VERKOT

Verkkojen suunnittelu on syytä suorittaa huolellisesti ennen niiden rakentamista testausympäristöön. Aluksi tuli valita erillinen sisäverkon IP-avaruus, jotta testausympäristö olisi erillään yrityksen omasta verkosta. Kyseiseen ip-avaruuteen tulisi muun muassa fyysisten palvelimien osoitteet.

Seuraavaksi piti suunnitella ympäristössä tarvittavien aliverkkojen lukumäärä, jotta voitaisiin kattavasti testata Arctic 3G Gatewayn eri käyttötarkoituksia eli asiakastapauksia. Verkot toimisivat virtuaalisina verkkoina fyysisten testilaitteiden sekä palvelimille asennettavin virtuaalikoneiden välillä. Virtuaalisista verkoista kerrotaan seuraavassa luvussa lisää. Ulkoisia ip-osoitteita tarvittiin, jotta testattavat laitteet, jotka hyödyntävät GPRS- ja 3G-verkkoa, voisivat ottaa yhteyden testiympäristöjen palvelinkoneisiin.

6.1 Virtuaalilähiverkot

Virtuaalilähiverkko (englanniksi Virtual LAN) eli VLAN on tekniikka, jolla fyysinen tietoliikenneverkko voidaan jakaa loogisiin osiin. Virtuaaliverkot helpottavat verkkojen hallinnointia, koska ne poistavat fyysiset esteet, joita verkkojen hallinnoinnissa perinteisesti on jouduttu huomioimaan. [17]

Verkkojen hallinnointi voidaan hoitaa nopeasti etäyhteyden avulla riippumatta henkilön sijainnista. Lisäksi virtuaaliverkkojen hyötynä on verkkojen parempi turvallisuus sekä verkkoliikenteen ohjattavuus ja seuranta. Virtuaalilähiverkkoja tukevat laitteet liittävät lähettämiinsä paketteihin tunnuksia, joiden avulla vastaanottava laite tietää, mihin verkkoon vastaanotettu paketti kuuluu. Kun paketti lähetetään eteenpäin laitteelle, joka ei tue virtuaalilähiverkkoja, poistetaan siitä tunnus. [17]

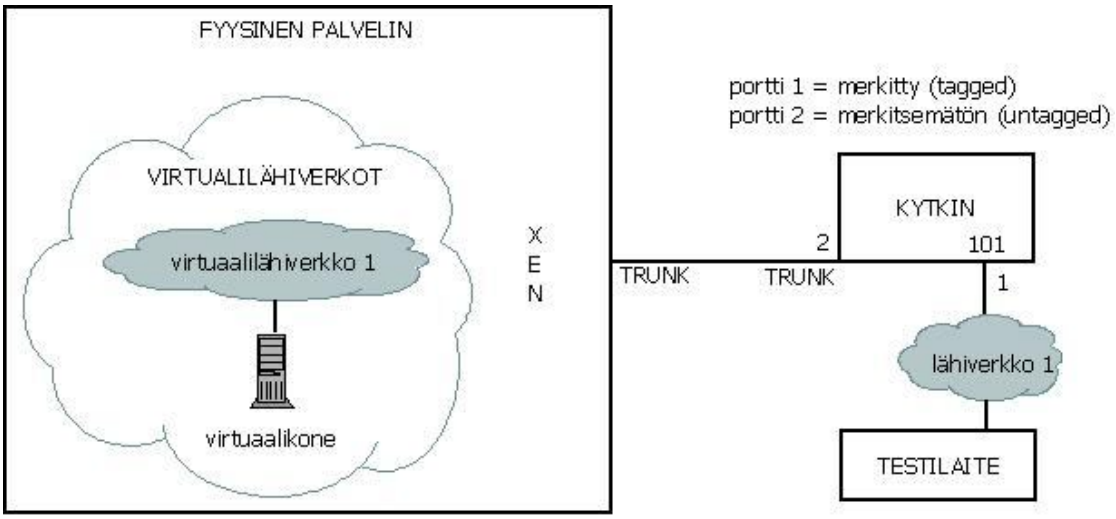
Laitteiden porttien määrityksissä käytetään termejä merkitty (tagged) ja merkitsemätön (untagged) sen mukaan, tukeeko vastaanottava laite virtuaalilähiverkkoja vai ei. Merkittyyn porttiin lähetettävään pakettiin liitetään virtuaalilähiverkkotunnus ja merkitsemättömään porttiin lähetettävästä paketista poistetaan mahdolliset tunnukset. Laitteen portti voi olla vain yhden virtuaalilähiverkon merkitsemätön jäsen, mutta useamman virtuaalilähiverkon merkitty jäsen. [17][18]

Virtuaalilähiverkkotuki asennettiin fyysiseen testauspalvelimeen, jota virtuaalisointiohjelmistossa käytettiin verkkoympäristön rakentamiseen. Näin pystyttiin määrittelemään kunkin virtuaalikoneen aliverkot. Kuvassa 6.1 havainnollistetaan termejä ja käytäntöä, jonka avulla yhteys fyysisen ja virtuaalisen verkon välille saatiin muodostettua. Tarkoituksena esimerkissä on yhdistää lähiverkko 1 ja virtuaalilähiverkko 1 toimimaan samana verkkona, jolloin kommunikointi virtuaalikoneelta testilaitteelle onnistuisi. Testilaite on kytketty porttiin 1, joka on asetettu untagged portiksi ja sille on annettu tunniste 101.

Portti 2 on merkitty tagged-portiksi. Kyseinen portti toimii trunk-porttina. Vastaavanlaisesti palvelimessa on trunk-portti. Trunk-portti tarkoittaa porttia, joka mahdollistaa usean virtuaalilähiverkon kuljettaa tietoa yhdestä fyysisestä liittimestä [17].

Palvelimeen asennetun virtuaalilähiverkkotuen ansiosta palvelimelle on luotu virtuaalinen Ethernet-rajapinta, joka sisältää tunnisteen 101. Virtuaalisointiohjelmisto XEN:n avulla luotu virtuaalikoneelle annetaan verkkorajapinnaksi tunnistetta 101 vastaava virtuaalinen Ethernet-rajapinta. Lopputuloksena testilaite pystyy kommunikoimaan virtuaalikoneen kanssa aivan kuin ne olisivat kaksi fyysisesti verkkokaapelilla yhdistettyä laitetta

.

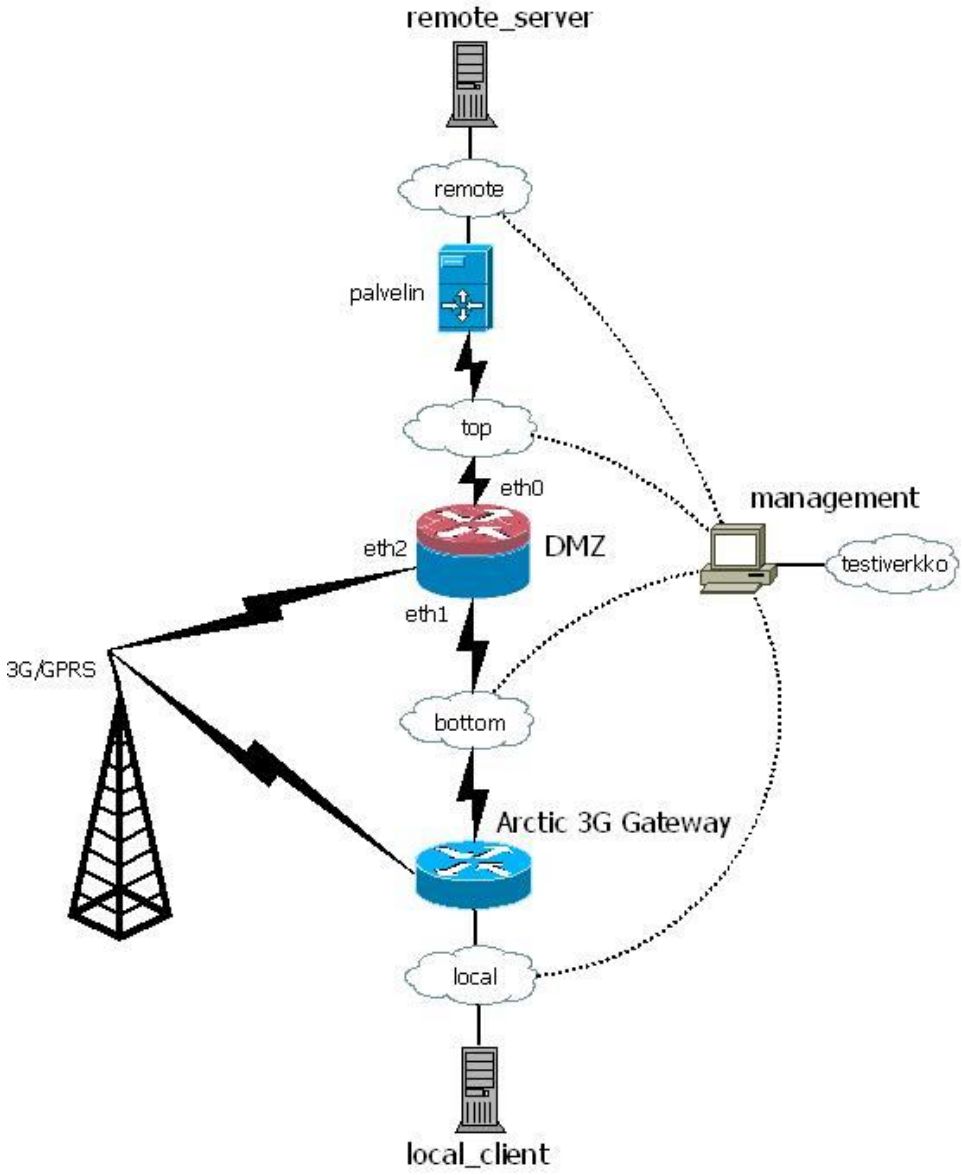


Kuva 6.1 Fyysisen- ja virtuaalisenverkon yhdistäminen

6.2 Testiverkkokaavio

Verkkokaavioiden pääasiallisena tarkoituksena on selventää testausympäristössä olevien laitteiden sijoittelua sekä näyttää niiden IP-osoitteet niiden omassa aliverkossaan. Tämä selventää kaikille ympäristön käyttäjille laitteiden olemassaolosta ja näin ollen myös ehkäisee päällekkäisten IP-osoitteiden antamisen.

Kuvan 6.2 verkkokaaviossa nähdään testausympäristö yhden laitteen kannalta. Ympäristö rakentuu viidestä erilaisesta virtuaalisesta sisäverkosta sekä ulkoverkosta. Näillä verkoilla pystytään toteuttamaan asiakaslähtöinen ympäristö testattaville laitteille.



Kuva 6.2 Verkkokaavio

Testattavan laitteen takana oleva local-virtuaalilähiverkko toimi VPN-tunnelin toisena päänä, josta yhteys oli tarkoitus saada muodostettua tunnelin toiseen päähän eli palvelimen takana olevaan remote-virtuaalilähiverkkoon. Molempiin verkkoihin asennettiin virtuaalikoneet (local_client, remote_server), joita hyväksi käyttäen pystyttiin testaamaan tunnelin toimivuus. Taulukossa 6.1 listataan neljän local_client-virtuaalikoneen

verkkotiedot. Jokainen verkko toimii erillään toisistaan, eli kyseiset laitteet eivät ”näe” toisiaan.

Local_client (IP)	Aliverkko	Verkkomaski	Oletuskäytävä (A2 Gateway IP-osoite)
10.10.12.2	10.10.12.0	255.255.255.0	10.10.12.1
10.10.13.2	10.10.13.0	255.255.255.0	10.10.13.1
10.10.14.2	10.10.14.0	255.255.255.0	10.10.14.1
10.10.15.2	10.10.15.0	255.255.255.0	10.10.15.1

Taulukko 6.1 Local-verkon esimerkkilaitteet

VPN-yhteyden muodostamiseksi palvelimeen testilaite hyödynsi 3G tai laajaverkkoa eli WAN-verkkoa. Yhteys 3G-verkkoon muodostettiin laitteen omalla 3G-moduulilla. Laajaverkkoa varten asennettiin bottom-virtuaaliverkko. Jotta testilaitteet voisivat ottaa yhteyden ulkoverkosta sekä bottom-verkosta palvelimeen, asennettiin DMZ-virtuaalikone, joka toimii verkossa oletuskäytävänä, palomuurina sekä liikenteen ohjaajana haluttuun palvelimeen käyttäen NAT-osoitteenmuunnosta.

DMZ:lla annettiin ulkoinen IP-osoite. Jotta voitaisiin muodostaa yhteys DMZ:sta palvelimeen, asennettiin top-virtuaaliverkko. Top-verkosta tuleva liikenne ohjattiin palvelimeen sekä palvelimelta tuleva liikenne ulkoverkkoon tai bottom-verkkoon. Ulkoverkkona toimi aikaisemmin mainittu top-virtuaaliverkko. Palvelimen sisäverkon osoite saatiin luomalla remote-virtuaaliverkko, jonka pääasiallisena tarkoituksena oli toimia HTTP-palvelimena sekä ”pingaus”-kohteena.

Robot Frameworkilla ajettavat testit tarvitsivat pääsyn testiverkkoihin, jotta jokaiseen laitteeseen ja virtuaalikoneeseen olisi yhteys. Tämän vuoksi ma-

nagement-virtuaalikoneelle, jolle Robot Framework oli asennettu, luotiin verkkorajapinnat jokaiseen verkkoon.

6.3 Reititys

Reitityksen tarkoituksena on mahdollistaa verkossa tapahtuva liikenne halutulla lailla mahdollisimman tehokkaasti ja nopeasti. Jotta tämä olisi mahdollista, tulee laitteelle kertoa kyseinen reitti. Tätä kutsutaan reititykseksi. Reititys koostuu osoitteistuksesta ja kommunikointiverkosta, joka toimii vastaavanlaisesti kuin postijärjestelmä. Aluksi postinkantaja toimittaa kirjeen tiettyyn postinumeroon välittämättä osoitteesta tai henkilöistä. Tämän jälkeen kirje toimitetaan osoitteeseen, jossa henkilö asuu. Lopuksi se henkilö avaa kirjeen, jolle kirje on tarkoitettu. [19]

Seuraavassa käydään läpi kuvan 6.2 verkkokaaviossa olevien laitteiden reititystä alhaalta ylöspäin. Kuvan alimpana laitteena oleva virtuaalikone (local_client) käyttää ainoastaan yhtä aliverkkoa, jolloin sille on asetettu vain IP-osoite sekä oletusreitti. Oletusreititin osoitteena toimii testilaitteen eli Arctic 3G Gatewayn local-verkon osoite. Testilaitteen oletusreittinä on DMZ-virtuaalikoneen bottom-virtuaaliverkon IP-osoite tai 3G:tä käytettäessä oletusreitti saadaan automaattisesti operaattorilta. DMZ vaati enemmän reitityssääntöjä. Aluksi estettiin sisääntuleva (input) sekä jatkettu (forward) liikenne sekä sallitiin ulosmenevä (output) liikenne.

iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

Tämän jälkeen määriteltiin poikkeustapaukset. Jos yhteyden tila oli jo muodostettu tai yhteys otetaan virtuaalikoneen sisältä (localhost), sallitaan liikenne. Li-

säksi sallitaan ping-paketit sekä SSH-yhteys bottom-virtuaaliverkosta konfigurointia varten.

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -s 127.0.0.0/8 -d 127.0.0.0/8 -i lo -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -s 172.30.29.0/24 -i eth1 -j ACCEPT
```

Ulkoverkosta tuleva liikenne ohjataan palvelimeen (x.x.x.x = ulkoinen IP-osoite, y.y.y.y = palvelimen top-virtuaaliverkon osoite).

```
iptables -A FORWARD -i eth0 -o eth2 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth0 -j ACCEPT
iptables -t nat -A PREROUTING -d x.x.x.x  -j DNAT --to-destination y.y.y.y
iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
```

Tämän jälkeen sallitaan liikenne bottom – ja top -virtuaaliverkkojen välillä.

```
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
```

Lopuksi estetään muu liikenne ulkoverkosta sisäverkkoon, sallitaan bottom-virtuaaliverkosta pääsy ulkoverkkoon sekä ulkoverkosta pääsy takaisin, jos yhteys on muodostettu aluksi sisäverkosta käsin.

```
iptables -A FORWARD -d 172.16.0.0/12 -o eth2 -j DROP
iptables -A FORWARD -i eth1 -o eth2 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Palvelimella olevia reitityksiä ei käydä läpi, koska palvelin oli valmis Viola Systems Oy:n palvelintuote (M2M Gateway), jonka vuoksi palomuuri ja reitit ykset olivat valmiiksi tehty jo asennusvaiheessa.

Management-virtuaalikoneelle lisättiin verkkorajapintoja sen mukaan, kun virtuaaliverkkoja asennettiin testausympäristöön. Jotta management-koneelta olisi pääsy local-virtuaaliverkkoihin, piti local-virtuaaliverkon rajapintaan asettaa virtuaalisia IP-osoitteita. Säännöt pystyttiin laittamaan ”/etc/network/interfaces/”-tiedostoon, josta koneen käynnistyessä säännöt asetettiin automaattisesti.

ip a a 10.10.12.65/24 dev eth10
ip a a 10.10.13.65/24 dev eth10
ip a a 10.10.14.65/24 dev eth10
ip a a 10.10.15.65/24 dev eth10

7 LAITETESTIT

Testien automatisointi oli työn tavoitteena, jonka vuoksi ympäristöä automatisoinnin mahdollistamiseksi alettiin alun perin rakentaa. Koska aikaisemmin kyseisiä testejä ajettiin manuaalisesti ihmisen toimesta, oli aika automatisoida testit. Automaattisilla testeillä voitaisiin toistuvasti ajaa useammalle laitteelle samaan aikaan testejä, jotka muutoin veisivät huomattavan määrän miestyötunteja. Lisäksi testien ajaminen automaattisesti varmistaisi sen, että testi ajetaan aina samalla lailla, jolloin voitaisiin mahdollisten ongelmien ilmetessä ajaa testit uudestaan ja löytää ongelma kohdat nopeasti.

Testien automatisoinnilla haetaan nopeita tuloksia, joista voidaan päätellä tuotteen sen hetkinen toiminnallisuus. Laitteen ohjelmistoa kehitettäessä tuotteelle tehdään useita ohjelmistoversiota, niin sanottuja ”dev-buildeja”, joita testataan automaattisesti, kun uusi versio käännetään käännöspalvelimella. Näin huomataan välittömästi, kannattaako versiota ottaa laajempaan testaukseen. Tätä testausta kutsutaan savutestiksi (englanniksi smoke test). Koska dev-buildeja voi tuotekehityksen aikana tulla useita kymmeniä, säästetään automatisoinnilla paljon miestyötunteja.

Automaattitestauksessa voidaan muun muassa suorittaa lyhytkestoisia rasitustestejä, jotka tuovat esille laitteen maksimaalisen sietokyvyn. Rasitustesti voi Arctic 3G Gatewayn tapauksessa olla esimerkiksi suurien tiedostojen lataaminen VPN–tunnelin läpi tai kuormittamalla sarjaporttia.

7.1 Automaattitestien suunnittelu

Automaattitestien suunnittelu pohjautuu olemassa oleviin manuaalisesti ajettaviin testeihin. Ensimmäisten testien rakentaminen aloitettiin tärkeysjärjestyksessä. Tärkeimpänä oli laitteen pääasialliseen toimintaan perustava VPN-yhteyksien testaaminen kiinteän yhteyden ja langattoman 3G-liikenteen ylitse. Testin rakentaminen koostui testausympäristön vaatimuksista, laiteasetuksista sekä testinsuoritustavasta.

Testausympäristön vaatimuksilla tarkoitetaan niitä edellytyksiä, joita testin suorittaminen edellyttää. Tämä tarkoitti virtuaalikoneiden hyödyntämistä ja lisäämistä, reitityksen suunnittelua sekä mahdollisten lisälaitteiden hankintaa.

Laiteasetuksilla tarkoitetaan asiakaslähtöisiä, testattavan laitteen sekä palvelinpään asetuksia, jotka mahdollistaisivat yhteyksien muodostamisen kyseisten laitteiden välillä. Erilaisten asetuskombinaatioiden pystyttiin testaamaan laitteen niitä ominaisuuksia joita asiakas käyttää ottaessaan sen käyttöönsä.

Testiensuoritustavalla pohdittiin sitä, millä tavalla voitiin todentaa, että kyseinen laite toimii oikein kyseisessä testaustilanteessa. Esimerkiksi VPN-yhteyksien testaamisessa korostui TCP/IP-pakettien liikkuvuus palvelimelta testattavalle

laitteelle ja takaisin. Testin piti huomata tilanne, jossa esimerkiksi TCP/IP-paketteja ”tippuu matkanvarrella”.

Testimoduulien rakentamisessa tuli pohtia sitä, miten voitaisiin hyödyntää samaa moduulia erilaisiin testitapauksiin antamalla eri komentoriviargumentteja. Tämä helpottaa moduulien jälleenkäyttöä tulevaisuudessa. Testien suunnittelun haasteena oli saada testimoduulit niin varmoiksi, että moduulit löytäisivät mahdollisimman varmasti ongelmakohdat ja ettei testi mene läpi siksi, että jokin näkökulma on testin kirjoittajalta jäänyt huomaamatta. Ihminen kirjoittaa testistä etsittävät virhetilanteet, jonka vuoksi testin ajaminen eri virhetilanteissa korostuu ennen testin käyttöönottoa.

7.1.1 VPN-tunnelin testausesimerkki

Tämän luvun tarkoituksena on käydä läpi VPN-tunnelin testaamiseen tarvittavat vaiheet, jotka Robot Frameworkilla toteutettiin.

Testin rakenne koostuu testilaitteen ja palvelimen konfiguroinnista sekä itse tunnelin testaustiedostosta. Testissä ei puututa VPN-tunnelin tyyppiin, vaan tarkoituksena on näyttää yksinkertainen esimerkki perustestistä, jota esimerkiksi Arctic 3G Gatewaylle ajettiin. Testitiedostot rakennettiin TSV -formaattiin.

01__vpn_palvelin_konfigurointi.tsv

02__vpn_testilaite_konfigurointi.tsv

03__vpn_testaa_tunneli.tsv

Ensimmäisenä testitiedostossa konfiguroitiin palvelin. Palvelimen konfigurointi tehtiin selaimen välityksellä. Tähän tarkoitukseen käytettiin Robot Frameworkin Selenium –kirjastoa sekä palvelimen konfigurointia varten asennettua virtuaalikonetta, jossa seleniumpalvelin sijaitsee. Taulukossa 7.1 on avainsana, jonka avulla kirjaudutaan palvelimelle. Seuraavana avainsanana voisi olla VPN tunnelin konfigurointi, joka tapahtuu vastaavanlaisilla komennoilla kuin taulukon 7.1 esimerkissä.

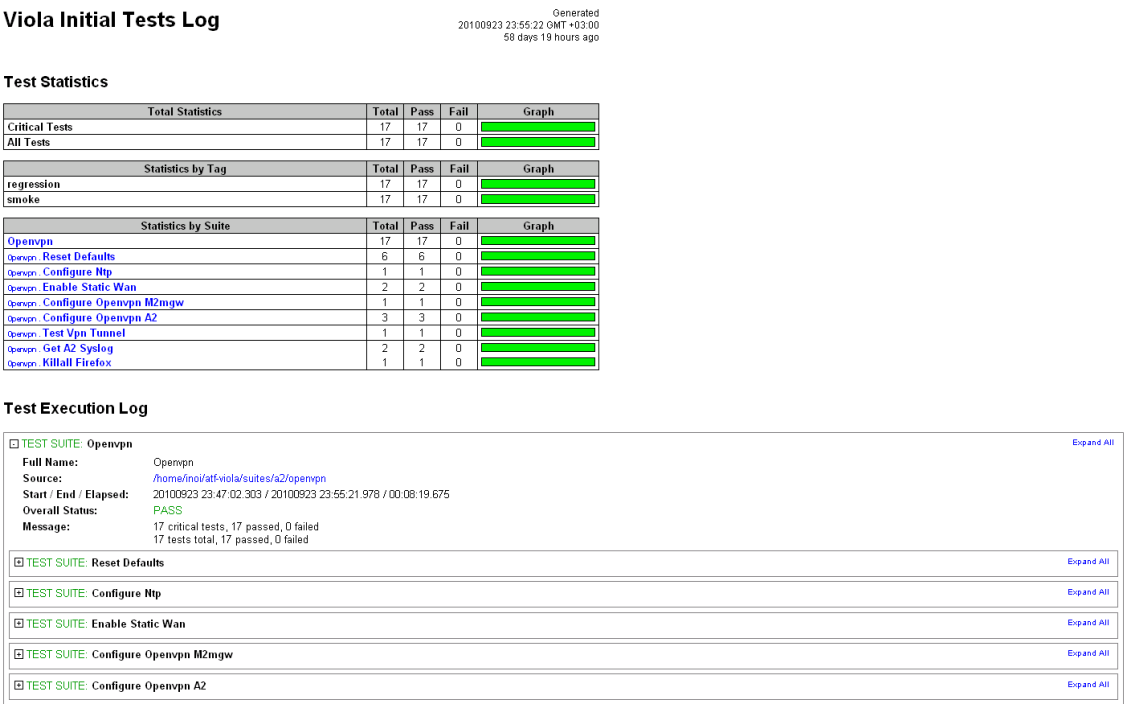
Kirjaudu palvelimelle			
	SeleniumLibrary.Open Browser	firefox	’palvelimen ip osoite’
	SeleniumLibrary.Input Text	’käyttäjäkentän id tai nimi’	’käyttäjänimi’
	SeleniumLibrary.Input Password	’salasanakentän id tai nimi’	’palvelimen salasana’
	SeleniumLibrary.Click Button	’painikkeen id tai nimi’	
	…		

Taulukko 7.1 Esimerkki avainsana

Palvelimen jälkeen konfiguroidaan testilaite vastaavalla tavalla kuin palvelin. Toisena vaihtoehtona olisi siirtää laitteen asetustiedosto suoraan ssh-yhteyden avulla laiteelle, jolloin säästetään aikaa. Testin viimeisenä vaiheena suoritetaan itse testi. Testissä ladataan HTTP-palvelimelta (remote_server virtuaalikone) tiedosto, joka on esimerkiksi 10 Mt:n kokoinen. Tiedoston lataus toteutetaan testilaiteen takana olevalta ”local_client”-virtuaalikoneelta. Jotta voitaisiin olla varmoja, että ladattu tiedosto vastaa alkuperäistä, otetaan md5-summa alkuperäisestä tiedostosta sekä ladatusta tiedostosta. Tämän jälkeen verrataan tiedostoja keskenään (Taulukko 7.2). Kuvassa 7.1 nähdään raportti onnistuneesta VPN-tunnelin testauksesta.

Testi			
	SSHLibrary.Open Connection	’local_client’	
	SSHLibrary.Login	’kayttaja nimi’	’salasana’
	SSHLibrary.Execute Command	wget http://’palvelimen ip osoite’/’tiedoston nimi'	
	${md5 summa}=	SSHLibrary.Execute Command	md5sum ’tiedoston nimi'
	Should Be Equal As Strings	${alkuperäisen tiedoston md5 summa}	${md5 summa}

Taulukko 7.2 Esimerkki avainsanasta, jossa ladataan tiedosto palvelimelta



Viola Initial Tests Log

Generated
20100923 23:55:22 GMT +03:00
58 days 19 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	17	17	0	
All Tests	17	17	0	

Statistics by Tag	Total	Pass	Fail	Graph
regression	17	17	0	
smoke	17	17	0	

Statistics by Suite	Total	Pass	Fail	Graph
Openvpn	17	17	0	
Openvpn. Reset Defaults	6	6	0	
Openvpn. Configure Ntp	1	1	0	
Openvpn. Enable Static Wan	2	2	0	
Openvpn. Configure Openvpn M2mgw	1	1	0	
Openvpn. Configure Openvpn A2	3	3	0	
Openvpn. Test Vpn Tunnel	1	1	0	
Openvpn. Get A2 Syslog	2	2	0	
Openvpn. Killall Firefox	1	1	0	

Test Execution Log

TEST SUITE: **Openvpn** Expand All

Full Name:	Openvpn
Source:	/home/inoi/atf-viola/suites/a2/openvpn
Start / End / Elapsed:	20100923 23:47:02.303 / 20100923 23:55:21.978 / 00:08:19.675
Overall Status:	PASS
Message:	17 critical tests, 17 passed, 0 failed 17 tests total, 17 passed, 0 failed

TEST SUITE: **Reset Defaults** Expand All

TEST SUITE: **Configure Ntp** Expand All

TEST SUITE: **Enable Static Wan** Expand All

TEST SUITE: **Configure Openvpn M2mgw** Expand All

TEST SUITE: **Configure Openvpn A2** Expand All

Kuva 7.1 Esimerkki raportti VPN-tunnelin testauksesta

7.2 Testien suorittaminen testausympäristössä

Aikaisemmassa työnvaiheessa kerrottiin Robot Frameworkin perustoiminnasta, joten tässä luvussa keskitytään Robot Frameworkin ympärille rakennettuun järjestelmään ja käydään ohjelmaa läpi tilakaaviomuodossa.

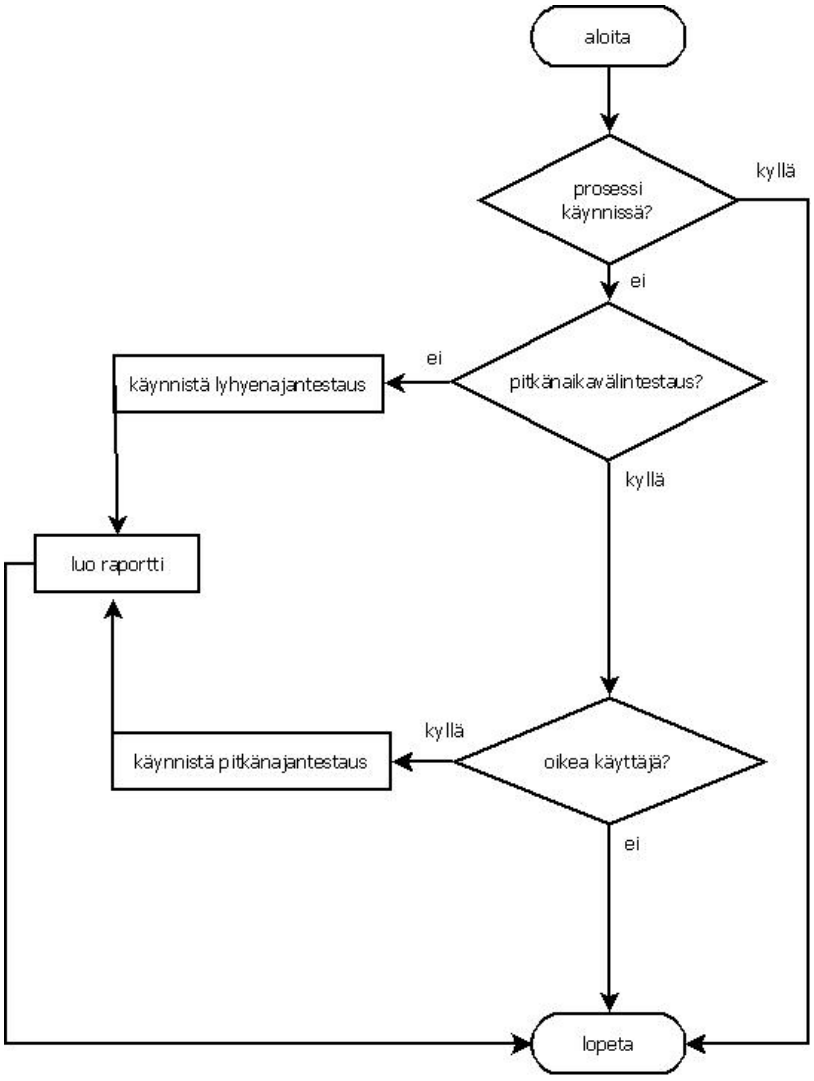
Robot Frameworkin ympärille rakennettiin python-ohjelmointikielellä lisäosia, joita hyväksikäyttämällä pystyttiin tekemään muun muassa testienajoympäristöön ja raportointiin liittyviä määrityksiä ja toimintoja. Testien suorittaminen pystyttiin aloittamaan komentoriviltä antamalla komento, joka sisältää laitteen ja testin.

run71.py hakemisto/testitiedosto.tsv

Poiketen Robot Frameworkin komennosta ”pybot.py”, käytetään komentona ”run71.py”, jolloin käynnistetään sarja erilaisia komentoja, jotka määrittävät laitteelle ominaiset komentorivioption, jotka mainittiin luvussa 4.2.2. Jokaisella laitteella on oma tunnistenumeronsa (esimerkki komennossa tunnistenumero on 71), jota komentorivioptiot sekä testauksen aikana käytettävät laitespesifisetmuuttujat hyödyntävät.

Kuvassa 7.2 käydään tilakaaviomuodossa ohjelmansuoritusvaiheita läpi yleisellä tasolla. Ohjelmassa on otettu huomioon tästä työstä pois rajattu pitkänajantestaus. Pitkänajantestaukselle on määritelty omat ympäristömuuttujat, joiden avulla pystyttiin erottelemaan kyseiset ympäristöt ja laitteet toisistaan.

Ensimmäisessä vaiheessa tarkistetaan, onko kyseisellä laitteella testin suoritus kesken. Jos testin suoritus on jo käynnissä, lopetetaan uuden prosessin suorittaminen. Muussa tapauksessa tarkistetaan, onko kyseinen laite tarkoitettu pitkänajantestatusta varten.



Kuva 7.2 Robot Frameworkin ympärille rakennettu ohjelman toiminnallisuus

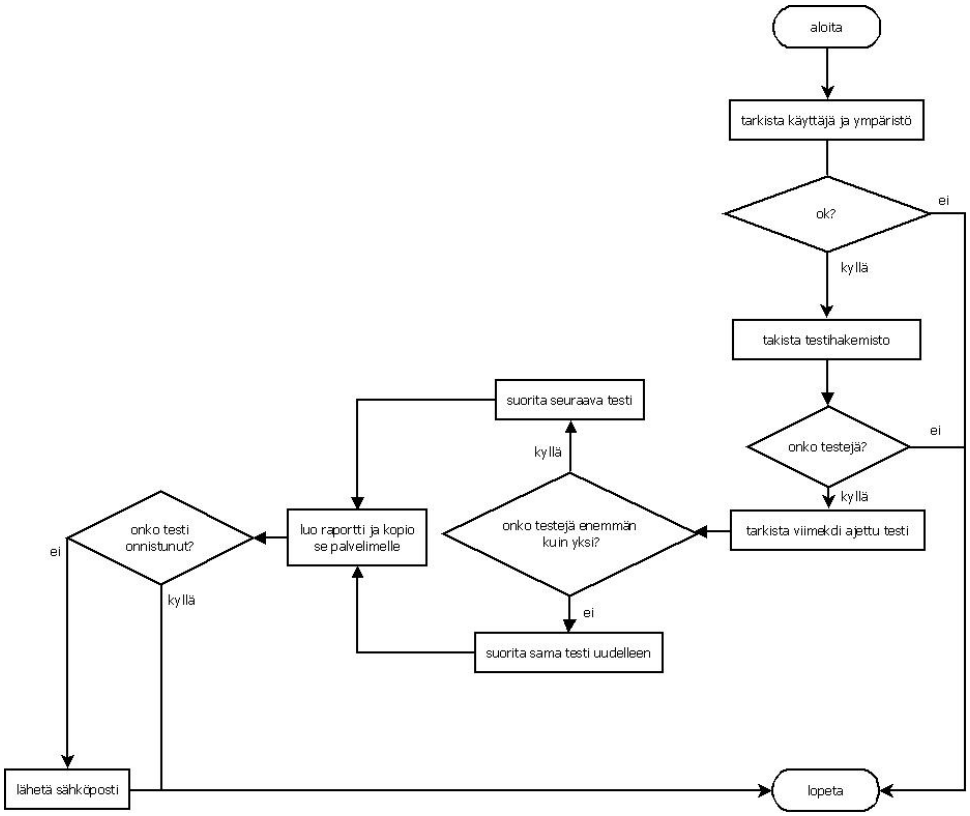
Laitteen ollessa pitkänajantestilaite tarkistetaan käyttäjä sekä ympäristö, jossa testiä suoritetaan. Tilanteessa, jossa väärällä käyttäjällä yritetään suorittaa testiä pitkänajantestiympäristössä, lopetetaan testiprosessin suoritus välittömästi. Tällä tavalla kyetään varmistamaan, että pitkänajantestejä ei keskeytetä tai häiritä.

Ympäristön ja käyttäjän ollessa oikeita, aloitetaan testin suoritus. Jos laite ei ole pitkänajantestilaite, käynnistetään lyhyenajantestit. Testin suorituksen jälkeen luodaan raportti ja tallennetaan se testauspalvelimelle.

Edellä mainitulla tavalla pystyttiin nopeasti ajamaan testimoduuleja ilman, että siitä jäi minkäänlaisia pysyviä raportteja laitteen testihistoriaan. Jotta laitteen testihistoria jäisi talteen, rakennettiin vastaavanlainen testinajotapa, kuin edellä mainittu, testinajotapa (kuva 7.2) mutta laajennetuilla toiminnoilla. Tämä tarkoitti ohjelman rakentamista, joka mahdollistaisi raportoinnin tietokantaan, hälytykset sähköpostin välityksellä, raporttien tallentamisen erilliselle palvelimelle sekä mahdollisuuden ajaa useita testisarjoja. Jokaiselle ajettavalle testikokonaisuudelle oli testitietokannassa testitapaus, johon automaattitestit viitattiin. Testien ajo suoritettiin erillisellä komennolla, joka lisättiin Cron-ohjelmiston ajastukseen:

*0 * * * * hakemisto/autorun.cron 71*

Esimerkissä ajetaan laitteelle numero 71 automaattitesti tasatunnein. Automaattitestit laitettiin erilliseen ”queue71” kansioon, josta ohjelma osaa suorittaa jokaisen testin vuorollaan kuvan 7.3 mukaisesti.



Kuva 7.3 Robot Frameworkin ympärille rakennettu laajempi ohjelmakokonaisuus

8 TESTAUSYMPÄRISTÖN KÄYTTÖÖNOTTO

Testausympäristön käyttöönotto tapahtu suunnitellulla aikataululla. Järjestelmä on toiminut osana Viola Systems Oy:n tuotekehitystä. Ympäristössä on ajettu automaattitestejä myös muille laitealustoille Arctic 3G Gatewayn lisäksi. Testit kattavat muun muassa VPN-yhteyksien testauksen.

9 YHTEENVETO

Testausympäristön rakentaminen projektina oli laaja. Työssä jouduttiin tekemään useaan otteeseen erilaisia muutoksia, niin laitteistojen kuin myös ohjelmistojen suhteenkin, koska vastaan tuli erilaisia yhteensopivuusongelmia. Reititys työssä oli vaivanloinen ja haasteellinen osa, johtuen useista laitteista sekä verkoista. Testauksen automatisointi on auttanut huomaamaan, kuinka paljon aikaa säästyy, kun testit automatisoidaan. Automaatin ajaessa testejä voidaan keskittyä uusiin testauskohteisiin, ja näin ollen testauksesta tulee nopeaa ja varmaa.

Testauksen automatisointi Robot Frameworkilla tuli nopeasti tutuksi ja laadukkaiden sekä uudelleenkäytettävien testien rakentaminen kehittyi ajan myötä. Varsinkin uusien testausalustojen liittäminen järjestelmään olisi kannattanut ottaa huomioon heti alkuvaiheessa, jolloin samojen testien ja testisarjojen käyttäminen olisi ollut alusta asti järkevää.

Tärkeimpänä oli kuitenkin saada käyttöön ympäristö, jolla pystytään ajamaan perustestejä testattaville laitteille ja sitä myöten kehittämään ympäristöä. Testausympäristön kehittäminen tulee jatkumaan uusien testikohteiden sekä uusien ympäristön ominaisuuksia ja käytettävyyttä parantavien ideoiden myötä.

LÄHTEET

[1] Watkins, John, ”Testing IT : An Off-the-Shelf Software Testing Handbook”, Cambridge University Press, 2001

[2] Hakala Ilkka – Jukka Märijärvi , ”Ohjelmistotuotanto”, Talentum, 2004

[3] Hass, Anne Mette Jonassen, ” Guide to Advanced Software Testing”, Artech House, 2008

[4] Copeland Lee, ” A Practitioner's Guide to Software Test Design”, Artech House, 2003

[5] Wikipedia, ”Vesiputousmalli”, [www-dokumentti]. Saatavilla: http://fi.wikipedia.org/wiki/Vesiputousmalli. (Luettu:11.8.2010)

[6] Loveland, Scot, ” Software Testing Techniques : Finding the Defects That Matter”, Charles River Media, 2005.

[7] Gao Jerry, ” Testing and Quality Assurance for Component-Based Software”, Artech House, 2003

[8] Wikipedia, ”Palvelin”, [www-dokumentti]. Saatavilla: http://fi.wikipedia.org/wiki/Palvelin. (Luettu:11.8.2010)

[9] Wikipedia, ”UPS”, [www-dokumentti]. Saatavilla: http://fi.wikipedia.org/wiki/UPS. (Luettu:11.11.2010)

[10] Warren, Steven S. , ”VMware Workstation 5 Handbook”, Boston, MA, USA: Course Technolgy, 2005

[11] Muller Al. ”Virtualization with VMware ESX Server”, Syngress Publishing, 2005

[12] Mext, ” virtualisointi-suomalaisissa-organisaatioissa”, [www-dokumentti].Saatavilla: http://www.mext.fi/fi/tutkimukset/virtualisointi-suomalaisissa-organisaatioissa-2009.html. (Luettu:6.6.2010)

[13] Xen hypervisor, ”WhatisXen”, [www-dokumentti] Saatavilla:http://www.xen.org/files/Marketing/WhatisXen.pdf. (Luettu:11.10.2010)

[14] Robot Framework, ”RobotFrameworkUserGuide”, [www-dokumentti] Saatavilla: http://robotframework.googlecode.com/svn/tags/ robotframework-2.1.3/doc/userguide/RobotFrameworkUserGuide.html#log-file. (Luettu:1.11.2010)

[15] Robot Framework, ”TestLibraries”, [www-dokumentti].Saatavilla: http://code.google.com/p/robotframework/wiki/ TestLibraries. (Luettu:1.11.2010)

[16] Selenium, ”Contributors”, [www-dokumentti].Saatavilla: http://seleniumhq.org/about/contributors.html. (Luettu: 7.12.2010)

[17] Rossi, Louis D, ” Cisco Catalyst LAN Switching”, McGraw-Hill Professional Publishing, 2000

[18] Wikipedia, ” VLAN”, [www-dokumentti]. http://fi.wikipedia.org/wiki/VLAN. (Luettu:11.8.2010)

[19] Medhi, Deepankar, "Network Routing : Algorithms, Protocols, and Architectu-res", Morgan Kaufmann, 2007

[20] Wikipedia, ”Ubuntu”, [www-dokumentti].http://wiki.ubuntu-fi.org/Esittely. (Luettu:11.13.2010)