Elias Pöyliö

Juniperin verkkolaitteiden automaatio

Junos-skriptauksen perusteet

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikka
Insinöörityö
17.4.2012

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Elias Pöyliö Juniperin verkkolaitteiden automatisointi 41 sivua + 2 liitettä 17.4.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja(t)	Yliopettaja Janne Salonen

Tämän insinöörityön tarkoitus oli tutkia Junos-skriptien toimintaa ja mahdollisuuksia. Työ tehtiin Metropolia ammattikorkeakoululle vuoden 2012 ensimmäisten kuukausien aikana.

Tietoa Junos skriptien ja niiden ympäristön, Junos-käyttöjärjestelmän, toiminnasta saatiin tarkkailemalla liikennettä Junos-laitteilla ja keräämällä tietoa useista eri dokumenteista. Junos-laitteilla myös testattiin erilaisia skriptejä.

Työn tuloksena kirjoitin tutkimuksen Junos-skriptauksen peruskäsitteistä ja niiden vuorovaikutuksesta käyttöjärjestelmän ja Junos-konfiguraation kanssa. Dokumentoin myös Junos-skriptien toimintaa laboratorioympäristössä.

Avainsanat	Juniper, Junos, skripti, SLAX, XML, FreeBSD

Metropolia

Abstract

Author(s) Title Number of Pages Date	Elias Pöyliö Automation of Juniper's network devices 41 pages + 2 appendices 17 April 2012
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Computer Networks
Instructor(s)	Janne Salonen, Principal Lecturer

Purpose of this thesis was to research the operation and possibilities of Junos scripts. The thesis was done for Helsinki Metropolia University of Applied Sciences during the first months of the year 2012.

Information about operation of Junos scripts and their environment, the Junos operating system, was gathered through monitoring traffic in Junos devices and by collecting data from several documents. Also, different types of Junos scripts were tested on Junos devices.

As result of this work I wrote a study about the basic concepts of Junos scripting and their interaction with the operating system and the Junos configuration. I also documented Junos script operation in a laboratory environment.

Keywords	Juniper, Junos, script, SLAX, XML, FreeBSD

Sisällys

Lyhenteet

API *Application Programming Interface*. Rajapinta, jonka mukaan eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään.

ASIC *Application Specific Integrated Circuit*. Mikropiiri, joka on suunniteltu yhden tuotevalmistajan tarpeisiin.

BGP *Border gateway protocol*, reititysprotokolla.

BSD *Berkeley Software Distribution*, nimitys toiselle Unix-päähaaralle ja siitä polveutuville järjestelmille.

CLI *Command Line Interface*, rajapinta käyttäjän ja ohjelman tai kahden ohjelman välillä

Daemon Akronyymi engl. sanoista *Disk And Execution MONitor*. Unixissa ja muissa moniajo-käyttöjärjestelmissä taustalla suoritettava ohjelma, jota käyttäjä ei suoraan hallitse.

DOM *Document Object Model*, ohjelmointirajapinta.

FTP *File Transfer Protocol*, tiedostonsiirtomenetelmä.

HTTP *Hypertext Transfer Protocol*, hypertekstin siirtoprotokolla.

RPC *Remote Procedure Call*, tapahtuma jossa ohjelma suorittaa toimenpiteen toisessa osoiteavaruudessa.

SCP *Secure Copy Protocol*, tiedostonsiirtomenetelmä.

SIM *SCSI interface module*, FreeBSD:n moduuli, jota käytetään apuna ulkoisten laitteiden liittämisessä.

SLAX *Stylesheet language alternative syntax*, Juniperin kehittämä Perlin kaltainen skriptikieli.

SNMP *Simple Network Management Protocol*, TCP/IP- verkkojen hallinnassa käytettävä tietoliikenneprotokolla.

XSLT *Extensible Stylesheet Language Transformations*, skriptikieli.

XML *Extensible Markup Language*, tiedonvälitykseen järjestelmien välillä ja dokumenttien tallentamiseen käytetty merkintäkieli.

1 Johdanto

Tietoverkon liikenteen katkeaminen on verkon ylläpidosta vastaaville ilmiö, jota mieluusti halutaan välttää. Myös verkonhallinnan toimenpiteisiin liittyviin pakollisiin katkoksiin kulunut aika halutaan pitää mahdollisimman pienenä tietoverkkoa edellyttävän toiminnan jatkumiseksi. Nämä hallitut huoltokatkokset ovat käytännössä todella pieni osa kaikista verkkokatkosten aiheuttajista. Muita aiheuttajia ovat erilaiset laitteisto- ja ohjelmisto-ongelmat sekä "inhimilliset tekijät". Näiden ihmisen aiheuttamien tapahtumien osuus on tutkimuksesta riippuen jopa 50 - 80% kaikista verkkokatkosten aiheuttajista. (1, s.3)

Juniper Networks on kehittänyt *Junos script automationina* tunnetun tavan luoda omia Juniperin laitteissa toimivia skriptejä verkonhallinnan toimenpiteiden helpottamiseksi ja vikatilanteiden korjaamisen nopeuttamiseksi. Skriptien avulla voidaan esimerkiksi varmistaa, että laitteille tehtävät konfiguraatiot eivät aiheuta ongelmia. Samoin konfigurointia voidaan nopeuttaa valmiiden sapluunoiden avulla, sillä usein verkonhallinta on saman tehtävän toistoa ja valmiiden tekstitiedostojen muokkausta ja kopiointia komentoriville. Silloinkin virheiden mahdollisuus kasvaa. Tämän työn tarkoitus on tutkia tekniikkaa, jolla skriptaus Juniperin laitteissa saadaan toimimaan sekä todentaa Junos-skriptien tuloksia eri dokumenttien ja testiverkon avulla.

2 Junos-automaatio

Junos-automaatio on sarja työkaluja, joilla voidaan automatisoida verkonhallinnan menetelmiä ja käytäntöjä. Sen tarkoitus on säästää aikaa, lisätä verkon suorituskykyä ja helpottaa laajan verkon hallintaa yksinkertaistamalla monimutkaisia tehtäviä. (2, s.8)

Työkalujen avulla voidaan ajaa suurin osa Junos-käyttöjärjestelmän komentorivin komennoista, puuttua *commit*-prosessiin ja automatisoida käytäntöjä määriteltyihin tapahtumiin. Junos-käyttöjärjestelmässä tunnetaan kolme erilaista automaation tapaa, jotka eroavat toisistaan automaation käynnistyksen mukaan:

- *Operation*-skriptit (*op*-skriptit) ajetaan käyttäjän antaman komentorivin syötteen kautta.

- *Event*-skriptit suoritetaan tilanteessa, jonka käyttäjä on määritellyt konfiguraatioon.

- *Commit*-skriptit suoritetaan *commit*-prosessin aikana, eli silloin kun konfiguraatioon tehdään muutoksia.

Junos-automaation avulla organisaatio voi sulauttaa henkilökunnan tai asiakkaiden kanssa sovittuja käytäntöjä suoraan Junos-käyttöjärjestelmään. Op-skriptien avulla monimutkaisista komentojen sarjoista voidaan tehdä yksinkertaisempia ja ohjattuja. Event-skriptit saavat Junos-laitteen seuraamaan omaa toimintaansa ja reagoimaan toivotulla tavalla sekä kirjaamaan ylös haluttua tietoa tapahtumista tulevan ongelmanratkaisun avuksi. Commit-skriptit tarkastavat konfiguraatiomuutokset ja voivat näin estää inhimillisiä virheitä. (2, s.70)

3 Juniperin käyttöjärjestelmä ja laitteet

3.1 Junos-käyttöjärjestelmä

Junos on käyttöjärjestelmä, joka toimii kaikissa Juniper Networksin laitteissa. Sen ensimmäinen versio julkaistiin vuonna 1998 Juniperin M40-reitittimen käyttöjärjestelmänä. Junosiin ilmestyy uusi päivitys neljä kertaa vuodessa.(3, s.14) Tämän insinöörityön tuloksena syntyneiden skriptien testaamiseen ja niitä varten tehtyyn tutkimustyöhön on käytetty versioita 10.0 ja 11.4.

Junos on käytännössä FreeBSD-käyttöjärjestelmä, jonka kernelin verkko-ominaisuuksia on muokattu runsaasti. Siihen on myös ohjelmoitu reitiysprotokollia ja daemoneja, jotka kuljettavat paketteja sekä pitävät yllä käyttöjärjestelmää ja sen alustaa. Kerneliin on lisätty laajennukset alustan hallintaa ja rajapintoja varten ja laitteisiin on kirjoitettu sulautettu mikrokerneli edelleenlähettämään paketteja.(3, s.15)

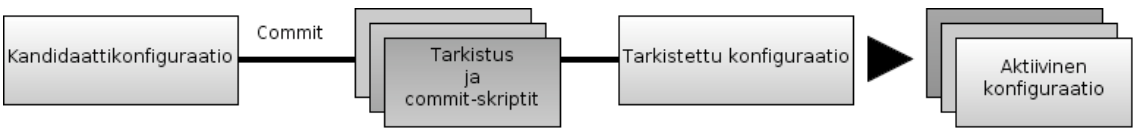
Junosissa toiminta tapahtuu usean eri prosessin ja daemonin toimesta, joten yhden tietyn prosessin voi lopettaa häritsemättä koko laitteen toimintaa. Pakettien edelleenlähetys ja reititys on jaettu siten, että edelleenlähetystä tekee useampi ASIC ja reitityksen tekee sisäänrakennettu PC. Junosin käyttöliittymä tukee komentojen täydennystä

ja tekstitiedostojen käyttöä. Edelliset konfiguraatiot säilyvät muistissa, joten niihin voi palata takaisin jälkeenpäin. Laitteissa on kovalevy myös päivityksiä, core dumpia ja dokumentaatiota varten.(3, s.18)

Junosin CLI:ssä on kaksi tasoa, toiminnallinen taso (*operational*) ja konfiguraatiotaso. Itse CLI:in päästään normaalin Unix-shellin kautta. Unix-shellissä voidaan suorittaa esimerkiksi tiedostojen hallintaan liittyvä toimenpiteitä. Toiminnallisella tasolla tapahtuu laitteen monitorointi. Sillä voidaan tarkistaa esimerkiksi rajapintojen tila, alustan hälytykset ja suorittaa käyttöjärjestelmän versionhallintaa. Laitteen ja sen rajapintojen konfigurointi tapahtuu konfiguraatiotasolla. Konfigurointi sisältää käyttäjänhallinnan, protokollien asetukset, turvallisuusasetukset ja järjestelmän laitteistoasetukset.(4, s.8)

3.1.1 Konfigurointi

Kun konfiguraatiota muutetaan Junosissa, se tapahtuu oman prosessinsa kautta, toisin kuin esim Cisco Systemsin IOS:ssä, jossa muutokset tulevat voimaan samalla kun komennot annetaan terminaalin kautta. Junosissa konfiguraatiomuutokset tehdään ensin kandidaattikonfiguraatioon, joka on kopio aktiivisesta konfiguraatiosta. Kandidaattikonfiguraatiota voi myös verrata aktiiviseen konfiguraatioon ennen kandidaattikonfiguraation aktivointia. Aktivointi tapahtuu *commit*- tai *commit confirmed* -komennoilla. Niiden jälkeen Junos tarkistaa kandidaattikonfiguraation ja virheiden löytyessä pysäyttää prosessin sekä antaa virheilmoituksen. Tässä vaiheessa myös mahdolliset commit-skriptit aktivoituvat. Mikäli virheitä ei löydy, niin kandidaattikonfiguraatio muutetaan aktiiviseksi konfiguraatioksi. Kandidaattikonfiguraatio tallentuu silloin tiedostoon */config/juniper.conf.gz* ja laite muutta edellisen aktiivisen konfiguraation tiedostonimen *juniper.conf.gz* tiedostonimeksi *juniper.conf.1.gz*.(4, s.39) Kuviolla 1 havainnollistetaan konfiguraatioprosessin tapahtumia.



Kuvio 1. Konfiguraatioprosessi.

3.1.2 Junos-käyttöjärjestelmän daemonit

Daemonit ovat käyttöjärestelmässä taustalla käyviä prosesseja, joita käyttäjä ei suoraan hallitse. FreeBSD:ssä ne suorittavat merkittävän osan tehtävistä. Näihin kuuluvat rutiininomaiset tehtävät, kuten esimerkiksi internet-yhteyksien käsittely (FreeBSD:ssä inetd-daemon) ja tehtävien aloittaminen tiettyyn aikaan (FreeBSD:ssä cron-daemon). Tavallisesti daemonit käynnistyvät kun järjestelmä käynnistetään ja toimivat siihen asti kunnes järjestelmä suljetaan.(5, s.151)

Daemonien toimintaa voi seurata esimerkiksi järjestelmän lokitiedostosta. Tiedostoon tallentuvissa viesteissä näkyy myös niiden lähteenä ollut daemon. Terminaaliesimerkissä 1 on viestin lähteenä ollut SNMP:tä hallinnoiva *snmpd*-daemon.

```
Jan  17  06:37:12     K6-JUN1  snmpd[1030]:  SNMPD_TRAP_COLD_START:
trap_generate_cold: SNMP trap: cold start
```

Terminaaliesimerkki 1. SNMP-daemonin järjestelmän lokitiedostoon jättämä viesti.

Junosissa on lukuisia daemoneja, joista kourallinen on sellaisia, joiden toiminta näkyy työn luonteesta riippumatta todennäköisesti myös käyttäjälle. Usein järjestelmän lokitiedostoa lukiessaan käyttäjä huomaa taulukossa 1 lueteltuja daemoneja, joiden lisäksi käyttöjärjestelmä käynnistää tarpeen vaatiessa useita muita.

Taulukko 1. Junosin toiminnan kannalta tärkeitä daemoneja.

Nimilyhenne	Nimi	Tehtävän kuvaus
rpd	Routing protocol daemon	Lähettää reititysprotokollien viestejä, päivittää reititystaulun ja ottaa käyttöön reitityskäytännöt.
dcd	Device control daemon	Kontrolloi rajapintojen fyysisiä ja loogisia ominaisuuksia.
mgd	Management daemon	Hallintadaemon, joka vastaa käyttäjän pääsystä laitteelle. CLI on hallintadaemonin pääte.
chassisd	Chassis daemon	Tarkkailee ja kontrolloi fyysisen laitteen ominaisuuksia.
pfed	Packet forwarding engine daemon	Vastaa pakettien uudelleenlähetysenginen ja reititysenginen välisestä kommunikaatiosta.

Esimerkiksi *event*-skriptit tarvitsevat konfiguraatioon määritellyn tapahtuman, joka käynnistää *event*-skriptin. Tärkein daemon *event*-skriptien kannalta on tapahtumia käsittelevä daemon, *eventd*. *Event*-skriptiin on kirjoitettu ohjeet siitä, miten tulee toimia ja konfiguraatioon on määritelty tapahtuma, joka käynnistää *event*-skriptin. Skriptin lopulta käynnistävä tieto tulee *eventd*-daemonilta.

3.2 Juniper SRX210 -yhdyskäytävä

Tässä insinöörityössä esiintyviä skriptejä kokeiltiin Juniper SRX210-laitteilla. SRX210 -yhdyskäytävä on tarkoitettu turvalliseen tiedonsiirtoon. Se tukee yleisiä verkon tekniikoita ja siihen on sisäänrakennettu paljon palomuuripalveluita. Lisäksi SRX210:ssä tietoturvaominaisuuksia, kuten UTM (*Unified Threat Management*), johon kuuluu roskapostinesto, virusturva ja verkon seulonta.(6, s.3)

SRX210 -yhdyskäytävässä on konsoliportti, kaksi gigabit-ethernet-porttia, kuusi fast-ethernet-porttia sekä Mini-PIM-korttipaikka lähi- ja laajaverkkokytkentöjä varten. Näiden lisäksi siinä on kaksi USB-porttia, joiden kautta voi käyttää ulkopuolista laitetta esimerkiksi käynnistykseen laitteen sisäisen flash-muisti -ongelman aikana.(6, s.12). Tässä insinöörityössä siirsin skriptejä muistitikulta laitteelle USB-portin kautta.

4 XML

4.1 XML:n perusteet

XML eli extensible markup language on tiedonvälitykseen järjestelmien välillä ja dokumenttien tallentamiseen käytetty standardi- tai merkintäkieli. Se on lisenssiltään vapaa, eli sitä ei ole sidottu minkään yksittäisen yrityksen liikeideaan tai ohjelmistoon.

XML:n ytimenä toimiva XML-spesifikaatio kertoo XML-dokumenttien elementtiperustaisen koodauksen kieliopin, jota tarvitaan hyvin muotoiltujen XML-dokumenttien kirjoittamiseen. Se myös määrittelee XML-dokumenttien tyyppikuvauskielen, jolla osoitetaan dokumenttien looginen rakenne.(7, s.1)

XML:ä voidaan kutsua metakieleksi, sillä se ei sisällä yhdenkään elementin määrittelyä ja merkkauksen semantiikka jää itse XML-sovellusten tehtäväksi. XML määrittelee XML-dokumenttien luokan kieliopin ja tarjoaa tyyppimäärityskielen. Toisin sanoen XML kuvaa syntaksia, jolla voidaan "luoda oma kieli".(7, s.7)

Otetaan esimerkki, jossa insinöörityö halutaan jakaa muiden ihmisten ja tietokoneohjelmien käyttöön. Sen sijaan että luotaisiin teksti- tai html-tiedosto, voidaan sisältö kirjoittaa XML-tiedostoon, joka itsessään sisältää kuvauksen itsestään ja rakenteestaan. Taulukossa 2 näkyvät eri tiedostomuotojen erot.

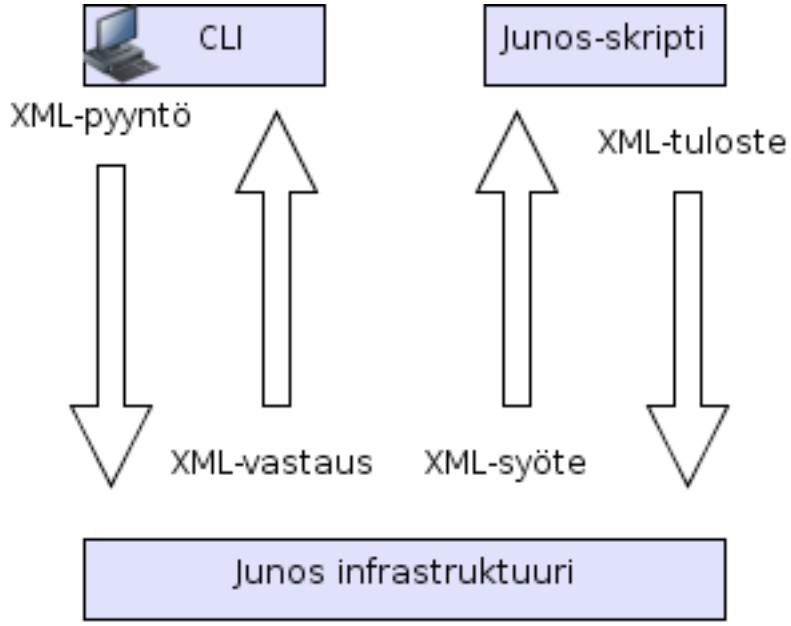
Taulukko 2. Esimerkit teksti- , HTML- ja XML-tiedostoista.

Tiedostonimi	Sisältö
tiedosto.txt	Insinöörityö Juniperin verkkolaitteiden automatisointi toimii.
tiedosto.html	<html> <head><title>Insinöörityö</title></head> <body>Juniperin verkkolaitteiden automatisointi toimii.</body> </html>
tiedosto.xml	<insinöörityö> <sisältö> Juniperin verkkolaitteiden automatisointi toimii.</sisältö> </insinöörityö>

4.2 XML-muotoisen tiedon kulku Juniperin laitteissa

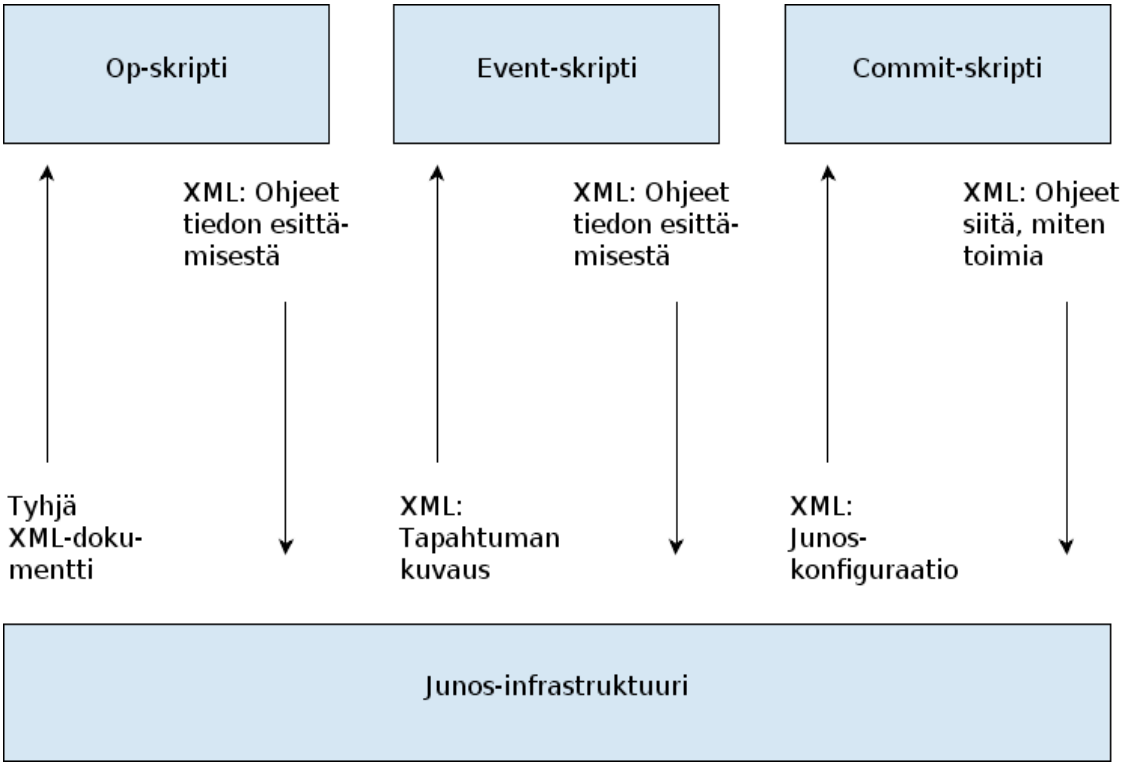
Juniperin laitteissa tieto kulkee XML-muodossa komentorivin ja Junosin infrastruktuurin välillä. Komentorivi lähettää esimerkiksi käyttäjän antamat konfiguraatiot XML-pyyntöinä käyttöjärjestelmälle ja antaa tiedon niiden onnistumisesta XML-vastauksena

komentoriville. Samalla tavalla viestivät keskenään myös Junos-skriptit ja Junos-infrastruktuuri. Skriptit vastanottavat tietoa XML-syötteenä ja antavat omat vastauksena XML-tulosteena. Esimerkiksi event-skripti vastaanottaa tiedon tapahtumasta käyttöjärjestelmän antamana XML-syötteenä ja antaa vastauksensa XML-tulosteena. Kuvio 2 havainnollistaa viestintää näiden toimijoiden välillä.



Kuvio 2. Tiedon kulku XML-muodossa Juniperin laitteessa.

Eri skriptien ja Junos-infrastruktuurin välillä kulkevassa liikenteessä on eniten eroja skriptien "sisääntulossa". *Op*-skriptit ottavat vastaan tyhjän XML-dokumentin ja lähettävät luomassaan XML-dokumentissa Junosille ohjeet siitä, että mitä tietoa sen tulisi esittää. Myös *event*-skriptit lähettävät samankaltaiset ohjeet, mutta ottavat Junosilta vastaan tapahtuman kuvauksen XML-dokumentissa. *Commit*-skriptit ottavat vastaan Junos-konfiguraation ja lähettävät Junosille ohjeet, siitä mitä tehdä seuraavaksi. XML-dokumentti voi ohjata esimerkiksi tekemään muutokset, lähettämään varoitusviestin tai lähettämään virheilmoitus. Kuvio 3 havainnollistaa skriptien "sisään" ja niistä "ulos" kulkevia viestejä.

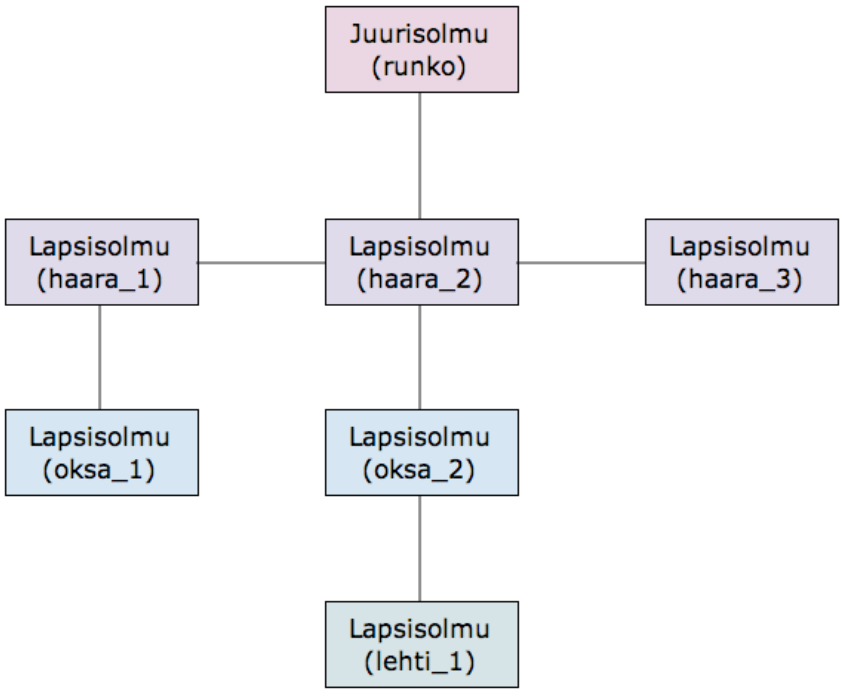


Kuvio 3. XML-dokumenttien vaihto Junos-infrastruktuurin ja eri skriptityyppien välillä.

4.2.1 Junos-konfiguraatio dokumenttipuuna

DOM (*document object model*) on esitysmalli, jonka mukaan XML-dokumentti esitetään hierarkkisena dokumenttipuuna. Puu koostuu solmuista (*nodes*), joista jokainen ilmentää yhtä elementtiä puun hierarkiassa.

Hierarkiassa ylimmällä tasolla on juurisolmu (*root node*). Juurisolmulla ja sen alapuolella olevilla lapsisolmuilla (*child nodes*) voi olla monia lapsisolmuja. Solmut voivat sisältää tietoa, joka toimii niiden niiden yläpuolella olevan äitisolmun (*parent node*) ominaisuutena tai lisää niiden omia lapsisolmuja. Jokaisella solmulla, paitsi juurisolmulla, on oma äitisolmu. Kuvio 4 havainnollistaa DOMin hierarkiaa.



Kuvio 4. Esimerkki DOM-dokumenttipuusta.

Kuvion dokumenttipuu koostuu "rungosta", joka on juurisolmu ja lapsisolmuista, jotka muodostavat "haaroja", "oksia" ja "lehden". XML-kielellä dokumenttipuu voitaisiin kuvata, kuten skriptiesimerkissä 1.

```
<?xml version="1.0"?>
<runko>
	<haara_1>
		<oksa_1>
		</oksa_1>
	</haara_1>

	<haara_2>
		<oksa_2>
			<lehti_1>
			</lehti_1>
		</oksa_2>
	</haara_2>

	<haara_3>
	</haara_3>
</runko>
```

Skriptiesimerkki 1. Dokumenttipuu XML-dokumenttina.

Junosin konfiguraation voi lukea myös XML-dokumenttipuuna. Terminaaliesimerkissä 2 näkyvät neljä tärkeää käsitettä, jotka ovat keskeisiä XML-dokumentin lukemisessa ja joiden avulla käyttäjä voi kommunikoida Junosin XML API:n kautta: elementit, solmut, attribuutit ja nimiavaruudet sekä XML-RPC-protokolla.

```
sandels@King> show configuration | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.4R2/junos">
    <configuration   junos:commit-seconds="1334322532"   junos:commit-
localtime="2012-04-13 13:08:52 UTC" junos:commit-user="root">
            <version>11.4R2.14</version>
            <system>
                <host-name>King</host-name>
                <root-authentication>
                    <encrypted-password>
                        $1$1kb/czay$4goUvpH9cqt67L3g1Bl/e.
                    </encrypted-password>
                </root-authentication>

                [...]

            </system>

           [...]

    </configuration>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
sandels@King>
```

Terminaaliesimerkki 2. Konfiguraatio XML-puuna.

Elementit ovat XML-dokumentin perusyksikkö. Elementit voivat sisältää tietoa, kuten merkkojonon, numeron tai muita elementtejä. Terminaaliesimerkin *<system>*-elementti on yksi elementti, jonka sisällä on muita elementtejä.

Junos-konfiguraation DOM-puurakenteessa *<system>*-solmu on sen *<host-name>*- ja *<root-authentication>*-lapsisolmujen äitisolmu. Dokumentin juurisolmu on *<configuration>*-solmu. Junosin *root*-käyttäjän salasana on salattuna merkkijonona *<encrypted-password>*-solmun sisällä.

Elementit voivat sisältää lisätietoja attribuutteina. Attribuuttien arvot kirjoitetaan lainausmerkkeihin yhtäläisyysmerkin jälkeen. Terminaaliesimerkissä *<configuration>*-elementillä on kolme attribuuttia: *junos:commit-seconds*, *junos:commit-localtime* ja *junos:commit-user*. Näiden arvoista voi lukea tietoja viimeisestä *commit*-prosessista.

Jokainen terminaaliesimerkin *<configuration>*-elementin attribuuteista alkaa *junos*-sanalla ja kaksoipisteellä. Nämä määrittelevät attribuutin nimiavaruuden. XML:n nimiavaruuksia käytetään tilanteissa, joissa useampi elementti tunnetaan samalla nimellä. Nimiavaruus määrittelemällä tietty nimi erotetaan muista nimistä. Esimerkiksi *com-*

mit-seconds-attribuutti voisi olla toisella taholla käytössä, mutta *junos:commit-seconds*-määrittely kertoo, että se kuuluu *junos*-nimiavaruuteen.

Terminaaliesimerkin tulosteen ensimmäinen ja viimeinen rivi tarkoittavat, että konfiguraatio kuuluu <*rpc-reply*>-elementtiin. Tämä johtuu siitä, että tuloste on Junosin vastaus RPC:nä pyydettyyn XML-tietoon.

4.2.2 XML-RPC

Jotta XML:ää voitaisiin käyttää eri loogisten objektien (esimerkiksi verkkopalvelun ja tietokannan) välillä, tarvitaan ratkaisu tiedon kuljettamiseen niiden välillä. Kun yksi objekti hakee tietoa toiselta objektilta, tätä ratkaisua kutsutaan *remote procdure callik-si* eli RPC:ksi. Tiedon kuljettamiseksi tarvitsee päättää esimerkiksi siitä, että missä haettava tieto sijaitsee, missä muodossa se halutaan noutaa, minkälaisen verkon yli se noudetaan ja miten se pakataan. Näitä kysymyksiä varten RPC:lle on kehitetty omat protokollansa.

XML-RPC on yksinkertaisimpia protokollia RPC:n toteuttamiseen. Se käyttää HTTP:tä kuljetusprotokollanaan ja XML:ää koodauskielenään. Yksinkertaisuudestaan huolimatta sillä pystytään käsittelemään monimutkaisiakin tietorakenteita.(8) RPC on keskeinen osa Junos-käyttöjärjestelmää. Jokainen komentoriville kirjoitettu käsky käännetään XML-RPC:ksi, jonka hallintadaemon *mgd* suorittaa.

Kaikki Junos-skriptit voivat käyttää RPC:tä tiedon hakemiseen. Jokaisella komennolla on oma RPC:nsä, jonka saa näkyviin kirjoittamalla komennon jälkeen pystyviivan ja *display xml rpc*, kuten terminaaliesimerkissä 3.

```
root@pawn1> show route | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.4R2/junos">
    <rpc>
        <get-route-information>
        </get-route-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>

root@pawn1> show system users | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.4R2/junos">
    <rpc>
        <get-system-users-information>
        </get-system-users-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>

root@pawn1>
```

Terminaaliesimerkki 3. RPC:t saa näkyviin komentoriviltä.

Skriptissä käytettävä osa on <*rpc*>-elementin sisältämä elementti. Terminaaliesimerkistä 3 selviää että komennot *show route* ja *show system users* voi suorittaa skriptissä skriptiesimerkin 2 osoittamalla tavalla.

```
var $get-route-info = <get-route-information> {
}

var $get-inventory = <get-system-users-information> {
}
```

Skriptiesimerkki 2. Terminaaliesimerkin RPC:t skriptissä.

4.3 XPath

XPath on kieli, jolla pystytään viittaamaan osoitteisiin XML-dokumentissa. Sillä on mahdollista muokata merkkijonoja, numeroita ja boolean-arvoja. (9)

XML-dokumentin osiin viitataan XPathin sijaintipolkujen (engl. 'location path') avulla. Terminaaliesimerkissä 2 nähdyn <*encrypted-password*>-elementin sijaintipolku nähdään skriptiesmerkissä 3.

```
system/root-authentication/encrypted-password
```

Skriptiesimerkki 3. Root-käyttäjän salatun salasanan sijaintipolku.

XPathissa viitataan solmun äitisolmuun samankaltaisella syntaksilla, kuin UNIXissa viitataan työskentelykansion äitikansioon. Skriptiesimerkissä 4 viitataan *root-authentication*-solmun äitisolmuun, *system*-solmuun.

```
../root-authentication/encrypted-password
```

Skriptiesimerkki 4. Äitisolmuun viitataan kahdella pisteellä.

Kun skriptissä haetaan Junosin konfiguraatiosta tietoa, niin todennäköisesti vastaan voi tulla tilanne, jossa sijaintipolku täytyy ohjata oikeaan solmuun. Skriptiesimerkissä 5 sijaintipolku viittaa useamman käyttäjän joukosta käyttäjän *sandels* käyttäjätunnukseen.

```
system/login/user[name='sandels']/uid
```

Skriptiesimerkki 5. Sijaintipolkua voi ohjata arvojen avulla.

Jos sijaintipolun halutaan viittaavan elementin attribuuttiin, kuten skriptiesimerkissä 6, sijaintipolussa käytetään @-merkkiä, kuten skriptiesimerkissä 7.

```
<wrapper>
    <element name="Elias"/>
</wrapper>
```

Skriptiesimerkki 6. Elementin nimi.

```
element/@name
```

Skriptiesimerkki 7. Elementin nimeen viittaava sijaintipolku.

4.4 XSLT

XSLT eli Extensible Stylesheet Language Transformations on XML:llä kirjoitettu merkintäkieli XML-tiedostojen muunnoksiin. Se käyttää XPathia löytääkseen XML-lähdedokumentista osat, joita käytetään prosessin tuloksena syntyvässä dokumentissa. Kaikki XPathin funktiot ovat XSLT-prosessissa käytettävissä ja näiden lisäksi XSLT 1.0 -versiossa on omiakin funktioita. (7, s.288)

XML-muunnoksen tuloksena syntyy kohdedokumentti, joka kuvaa lähdedokumenttia. Muunnos kuvaa lähdedokumentin merkkauksen puurakenteen eli lähdepuun (source tree) kohdedokumentin merkkauksen puurakenteeksi eli tulospuuksi (result tree). Itse lähdepuuta ei mitenkään muuteta muunnoksessa vaan muunnos kuvaa vain miten tulospuu kuvaa lähdepuun. Ne voivat erota toisistaan niin, että lähdepuusta on poistettu

(tai jätetty esittämättä) osia tai niin, että tulospuuhun on lisätty osia, joita ei ole lähdepuussa.(10, s.203)

4.5 SLAX

4.5.1 SLAXin perusteet

SLAX eli stylesheet language alternative syntax on Juniperin XSLT:n päälle kehittämä skriptikieli. XSLT:n syntaksi on yleisempiin ohjelmointi- ja skriptauskieliin tottuneelle usein monimutkaisen näköistä, joten Juniper Networks alkoi kehittää vuonna 2005 SLAXia skriptien kirjoittamisen helpottamiseksi.(11) SLAX muistuttaa syntaksiltaan yleisiä ohjelmointi- ja skriptauskieliä, kuten C:tä tai Perliä. Näin ollen muita tavanomaisimpia kieliä opetelleelle SLAX on todennäköisesti helpompi omaksua kuin XSLT. Asiaa tutkiessani en löytänyt kuitenkaan muita tahoja, jotka käyttäisivät SLAXia. SLAXin helpompilukuisen syntaksin takia kaikki tämän insinöörityön skriptit on kirjoitettu SLAXilla.

SLAX on avoimen lähdekoodin toteutus ja se on jaossa *libslax*-kirjastona. Kirjasto on kirjoitetettu C-kielellä ja se perustuu Daniel Veillardin *libxml2*- ja *libxslt*-kirjastoihin.(11)

Seuraavissa skriptiesimerkeissä näkyy joitain SLAXin ja XSLT:n eroja. Skriptiesimerkki 8 on kirjoitettu SLAXilla.

```
/* SSH-testi */
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
match configuration {
    /* Varmista onko SSH konfiguroitu */
    if( jcs:empty( system/services/ssh ) ) {
        <xnm:error> {
            <message> "SSH must be enabled.";
        }
    }
}
```

Skriptiesimerkki 8. SLAXilla kirjoitettu skripti, joka tarkistaa onko SSH käytössä.

SLAXilla kirjoitetun skriptin saa Junosissa käännettyä XSLT:ksi ja toisinpäin. Terminaaliesimerkissä 4 suoritetaan käännös ja tulostetaan tuloksena syntynyt XSL-tiedosto.

```
root@pawn1> request system scripts convert slax-to-xslt source
/var/db/scripts/commit/ssh-testi.slax                    destination
/var/db/scripts/commit/ssh-testi.xsl
conversion complete

root@pawn1> file list /var/db/scripts/commit

/var/db/scripts/commit:
ssh-testi.slax*
ssh-testi.xsl

root@pawn1> file show /var/db/scripts/commit/ssh-testi.xsl
<?xml version="1.0" standalone="yes"?>

<xsl:stylesheet        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:junos="http://xml.juniper.net/junos/*/junos"
xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:jcs="http://xml.juniper.net/junos/commit-scripts/1.0"      versi-
on="1.0">
  <!-- SSH-testi -->
  <xsl:import href="../import/junos.xsl"/>
  <xsl:template match="configuration">
    <!-- Varmista, että SSH on konfiguroitu -->
    <xsl:if test="jcs:empty(system/services/ssh)">
      <xnm:error>
        <message>SSH must be enabled.</message>
      </xnm:error>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
root@pawn1>
```

Terminaaliesimerkki 4. SLAX-skripti käännetynä XSLT:ksi.

4.5.2 Muuttujat

SLAX:n muuttujat määritellään vain kerran ja sen jälkeen niitä ei muuteta. Määriteltäessä muuttujan eteen laitetaan lause *var*, joka kertoo kääntäjälle, että seuraavaksi määritellään muuttuja. Muuttujan nimen eteen tulee dollarin symboli $, samoin kuin esimerkiksi Perlissä. Muuttujan nimi voi sisältää isoja ja pieniä kirjaimia, numeroita, viivoja ja alaviivoja.

SLAX tukee neljää XPathin tietotyyppiä: merkkijonoja, numeroita, boolean-arvoja ja solmujoukkoja. Näiden lisäksi SLAX tukee myös XSLT:n *result tree fragment* -tietotyyppiä (RTF).

Merkkijonot määritellään joko yksin- tai kaksinkertaisilla lainausmerkeillä. SLAXissa voi myös käyttää XPathin *string()*-funktiota, joka kääntää sille annetun muuttujan merkkijonoksi.

Numerot ovat IEEE 754:n määrittelemiä liukulukuja, joihin kuuluu myös erikoisarvo *NaN* ("Not a number"). Numeeristen muuttujien arvot voi määritellä desimaalien kanssa tai ilman desimaaleja. Muuttujat voivat määräytyä myös matemaattisten lausekkeiden tai XPathin *number()*-funktion avulla.

Boolean-muuttujat voivat saada joko arvon "tosi" tai "epätosi". SLAX ei tunnista kirjallisia boolean-arvoja, mutta siinä voidaan käyttää XPathin *true()*- ja *false()* -funktioita sekä *boolean()*-funktiota, joka kääntää sille annetun arvon joko "todeksi" tai "epätodeksi". Numeeriset arvot kääntyvät "todeksi" lukuunottamatta nollaa, joka kääntyy "epätodeksi". Merkkijonot kääntyvät "todeksi", ellei arvo ole tyhjä, jolloin se kääntyy "epätodeksi". Samalla logiikalla myös tyhjät solmujoukot kääntyvät "epätodeksi" ja arvoja sisältävät "todeksi". Huomionarvoista on RTF:stä aina löytyvä juurisolmu, jonka vuoksi "tyhjäkin" RTF kääntyy aina "todeksi".

Solmujoukkojen avulla voidaan käsitellä monimutkaisia tietorakenteita XML-muodossa. XSLT ei tunne taulukoita, mutta solmujoukkojen avulla voidaan luoda ja muokata semanttisesti samankaltaisia rakenteita.

RTF on XSLT:ssä esiintyvä konsepti, jolla voidaan siirtää tietoa tulospuuhun, käsitellä RTF:n sisältöä merkkijonona tai kääntää se solmujoukoksi.(12, s.26)

4.5.3 Lauseet

Monia XSLT:n elementtejä pystyy SLAXissa käyttämään valmiiden lauseiden kanssa. Valmiilla lauseilla on pyritty tekemään usein esiintyvien elementtien käyttö helpommaksi ohjelmoijalle.(13, s.9) Valmiita lauseita on yhteensä 20, joista hyvin yleisiä ovat esimerkiksi *import*, *match*, *ns*, *var* sekä ehto- ja toistolauseet.

Skriptin ulkopuolisista tiedostoista saa liitettyä koodia skriptiin *import*-lauseella. Skriptiesimerkissä 9 tuodaan tietoa *junos.xsl* -tiedostosta SLAX-vakioskriptiin. Kun tiedosto liitetään *import*-lauseella, sen sisältämä koodi on hierarkiassa alempana kuin koodia hakevan skriptin koodi. Tämä erottaa *import*-lauseen *include*-lauseesta, joka toimii muuten samoin, mutta sen sisältämää koodia luetaan kuin se olisi osa hakevaa skriptiä. Näin ollen vakioskriptissä tulee aina liittää *junos.xsl* -tiedosto *import*-lauseella, jotta

junos.xsl -tiedoston sisältämää *match*-lausetta ei lueta ennen vakioskriptiin kuuluvaa *match*-lausetta.

```
import "../import/junos.xsl";
```

Skriptiesimerkki 9. Import-lause.

Kun skripti ajetaan, ensimmäisenä luettu *match*-lause sisältää ensimmäiset varsinaiset toiminta-ohjeet. Lause kirjoitetaan skriptin ylimmälle tasolle luomaan *match*-sapluuna ja sen perään tulee sapluunaa vastaava kaava. Näiden jälkeen aaltosulkeiden sisään kirjoitetaan varsinainen sapluunan koodi. Skriptiesimerkissä 10 näkyy *match*-lauseen syntaksi.

```
match kaava {
        /* koodia */
}
```

Skriptiesimerkki 10. *Match*-lause.

Nimiavaruudet määritellään SLAXissa käyttäen *ns*-lausetta. Jokaisessa SLAX-skriptissä määritellään skriptiesimerkissä 11 näkyvät vakionimiavaruudet.

```
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
```

Skriptiesimerkki 11. SLAXin vakionimiavaruudet.

SLAX-skriptin muuttujat määritellään *var*-lauseella. Skriptiesimerkissä 12 määritellään käyttäjän syötteen mukainen merkkijono.

```
var $static-route = jcs:get-input("Enter the static route: ");
```

Skriptiesimerkki 12. Muuttujan määrittely *var*-lauseella

Ehtolauseet kirjoitetaan SLAXissa samalla tavalla kuin esimerkiksi Perlissä, eli niin, että boolean-arvo tulee sulkeiden sisään ja ehdolliset lauseet aaltosulkeiden sisään. Samalla tavalla kirjoitetaan myös toistolauseet, joissa käsiteltävä solmujoukko kirjoitetaan sulkeiden sisään ja suoritettava koodi aaltosulkeiden sisään. Skriptiesimerkeissä 13 ja 14 on käytetty *if*- ja *for-each*-lausetta.

```
var $viikonpaiva = date:day-name();
if( $viikonpaiva == "Saturday" || $viikonpaiva == "Sunday" ) {
    <output> "On viikon!oppu.";
}
else {
    <output> "On arkipaiva.";
}
```

Skriptiesimerkki 13. Yksinkertainen *if*-lause.

```
var $configuration = jcs:invoke( "get-configuration" );
for-each( $configuration/interfaces/interface ) {
    <output> name;
}
/*
* Tuloste:
* ge-0/0/0
* ge-0/0/1
* fe-0/0/2
* fe-0/0/3
* fe-0/0/4
* fe-0/0/5
* fe-0/0/6
* fe-0/0/7
*/
```

Skriptiesimerkki 14. Yksinkertainen *for*-lause.

4.5.4 Operaattorit

Operaattoreiden tarkoitus on tehostaa skriptien toimintaa. Operaattoreiden avulla voidaan suorittaa matemaattisia laskuja, vertailla ja kääntää arvoja sekä muodostaa monimutkaisia lausekkeita.(2, s.28) Liitteessä 1 esitetään SLAXin operaattorit.

SLAXissa käytettävien operaattoreiden voi katsoa olevan samoja, kuin XSLT:ssä ja useimmissa ohjelmointikielissä. Eräs yleinen operaattori, "not", kuitenkin puuttuu SLAXista. Sen sijaan SLAXissa voi kuitenkin käyttää *not()*-funktiota, joka palauttaa sille syötetyn arvon vastakkaisen boolean-arvon.(2, s.30).

4.5.5 Funktiot

SLAX 1.0:ssa ei ole tukea funktioiden käytön syntaksille, joten funktiot kirjoitetaan siinä XSLT:llä. Itse funktiot sisältyvät Junosin mukana tulevaan *libxlst*-kirjastoon. Jotta funktioita voidaan käyttää, täytyy *func*-nimiavaruus ilmoittaa skriptin alussa, kuten skriptiesimerkissä 15.

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";

ns func extension = "http://exslt.org/functions";

ns mycorp = "http://xml.mycorp.com/junos"
import "../import/junos.xsl";
```

Skriptiesimerkki 15. Funktioiden käyttöön tarvittava nimiavaruus on ilmoitettu rivillä 6.

5 Erilaiset skriptit

5.1 Kolme erilaista skripti-tyyppiä

Juniper jakaa skriptit kolmeen eri ryhmään: *op*- , *commit*- ja *event*-skriptit. Tämä malli perustuu siihen, miten skriptit suoritetaan. Ne on mahdollista suorittaa joko käyttöjärjestelmän pyynnöstä tai käyttäjän komennosta.

Jokaisen tyypin vakioskriptissä määritellään SLAX:n versio, vaadittavat kolme nimiavaruutta ja *import*-lause, jolla tuodaan *junos.xsl* -tiedostosta käytännölliset parametrit ja sapluunat. Näiden lisäksi varsinainen skripti kirjoitetaan *match*-komennon jälkeen.

5.2 *Op*-skriptit

Op-skriptit ajetaan käsin komentoriviltä tai sisään kirjautuessa eli ne on suunniteltu interaktiivisiksi eikä reagoimaan automaattisesti tapahtumiin. (2, s.70) Kaikille skripteille on oma tallennuspaikkansa Junos-laitteella ja *op*-skriptien tallennuspaikka on kansiossa */var/db/scripts/op/*.

Skriptiesimerkissä 16 on yksinkertainen *op*-skripti ja terminaaliesimerkissä 5 komento, jolla skripti konfiguroidaan. Komentoriviltä ajettaessa skripti tulostaa terminaaliin merkkijonon "Hello World!".

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
match / {
  <op-script-results> {
       <output> "Hello World!";
       }
}
```

Skriptiesimerkki 16. *Op*-skripti, joka komentoriviltä ajettaessa tulostaa ruudulle tekstiä.

```
[edit]
root@K6-JUN1# set system scripts op file hello-world.slax
```

Terminaaliesimerkki 5. *Op*-skripti täytyy myös konfiguroida.

5.3 *Event*-skriptit

5.3.1 *Event*-skriptien ominaisuuksia

Event-skriptit eroavat Op-skripteistä siten, että ne ajetaan järjestelmän käskystä reaktiona määriteltyyn tapahtumaan eikä käyttäjän toimesta komentoriviltä. Event-skriptien ja op-skriptien ajamista voidaan pitää niiden ainoana suurena erona. Molemmat kirjoitetaan käyttäen joko XSLTtä tai SLAXia ja noudattaen samoja ohjelmoinnin sääntöjä. (2, s.70)

Mahdollisia event-skriptin ajamisen käynnistäviä tapahtumia ovat systeemin lokitiedoston merkinnät, alustan hälytykset ja SNMP-ansojen sekä sisäisten laskurien viestit. Event-skriptit voivat reagoida näihin tapahtumiin keräämällä tietoa ongelmanratkaisua varten, käynnistää op-skriptin tai muuttaa itse konfiguraatiota. Näin laite saadaan reagoimaan itsestään suunnittelemattomiin muutoksiin tietoverkossa tai suunnitellusti toimimaan tietyllä tavalla tietyllä ajan hetkellä.

Skriptiesimerkissä 17 näkyy yksinkertainen event-skripti, jossa on Junos-vakioskriptin määrittelyt ja *event-script-results*-solmun jälkeen varsinainen skriptin toiminta. Skripti kirjoittaa järjestelmän lokitiedostoon "Hello world!". Skriptin lisäksi konfiguraatioon pitää määritellä tapahtumakäytäntö(engl. 'event policy'). Terminaaliesimerkissä 6 näkyy tekemäni tapahtumakäytäntö, joka käynnistää skriptiesimerkin, kun käyttäjä aloittaa *commit*-prosessin.

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
match / {
    <event-script-results> {
        /* Send Hello World! to syslog from facility external with se-
verity
info */
        expr jcs:syslog("external.info", "Hello World!");
    }
}
```

Skriptiesimerkki 17. *Event*-skripti, joka kirjoittaa lokitiedostoon tekstin "Hello world!".

```
[edit]
root@K6-JUN1#   set   event-options   event-script   file   log-hello-
world.slax;

[edit]
root@K6-JUN1# edit event-options

[edit event-options]
root@K6-JUN1# show
event-script {
    file log-hello-world.slax;
}

[edit event-options]
root@K6-JUN1# set policy hello-world events ui_commit

[edit event-options]
root@K6-JUN1# edit policy hello-world

[edit event-options policy hello-world]
root@K6-JUN1# set then event-script log-hello-world.slax
```

Terminaaliesimerkki 6. *Event*-skriptille täytyy tehdä konfiguraatioon tapahtumakäytäntö, joka suorittaa skriptin.

5.3.2 Tapahtumakäytännöt

Tapahtumakäytännöt luodaan antamaan Junosille ohjeet siitä, miten sen tulisi tietyissä tilanteissa toimia. Jokaisella tapahtumakäytännöllä on useimmissa ohjelmointikielissä esiintyvä *if-then*-rakenne. *If*-osio sisältää yhden tai useamman ehdon. Jos nämä ehdot täyttyvät, Junos suorittaa *then*-osiossa määritellyt toimentpiteet. Yksi tapahtuma voi laukaista yhden tai useamman tapahtumakäytännön. Junosin tapahtumia käsittelevä daemon käsittelee tapahtumakäytännöt sekvensseinä siinä järjestyksessä, jossa ne on konfiguroitu sekä suorittaa kaikki tapahtumaan sopivat käytännöt.(2, s.80)

Junos pystyy ajamaan korkeintaan 15:sta tapahtumakäytäntöä samaan aikaan. Tämä rajoitus on luotu säästämään laitteen resursseja ja ehkäisemään tapahtumasilmukoita. Jos ajettavien tapahtumakäytäntöjen määrä ylittyy, Junos ei aja enempää käytäntöjä vaan kirjoittaa aiheesta viestin järjestelmän lokitiedostoon.(2, s.80)

5.4 *Commit*-skriptit

5.4.1 *Commit*-skriptien ominaisuuksia

Junos ajaa *commit*-skriptit konfiguraatiomuutoksia tehdessä. Näin niillä voidaan hallita muutoksia joko varoitusviesteillä tai konfiguraatiomakrojen avulla. (2, s.128)

Commit-skriptit eroavat muista skripteistä näkyvimmin siten, että niissä ei ole samanlaista *match /* -lausetta kuin *op*-skripteissä tai *event*-skripteissä. Sen sijaan niissä on *match configuration* -lause, joten *junos.xsl*-tiedostossa oleva *match /* -sapluuna luetaan ensin.

Skriptiesimerkki 18 tarkistaa, että kandidaattikonfiguraatioon on konfiuroitu seuraavat elementit:

- SSH-palvelu
- käyttäjätunnus *jnpr*
- fxp0-rajapinnan IP-osoite

Mikäli joku elementeistä puuttuu, niin *commit*-prosessi keskeytyy ja skripti tulostaa virheilmoituksen. Terminaaliesimerkissä 7 konfiguroidaan skriptiesimerkki 18.

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
match configuration {
    if( jcs:empty( system/services/ssh ) ) {
        <xnm:error> {
            <message> "SSH must be enabled.";
        }
    }
if( jcs:empty( system/login/user[name == "jnpr"] ) ) {
        <xnm:error> {
            <message> "The jnpr user account must be created.";
        }
    }
var $fxp0-interface = interfaces/interface[name == "fxp0"];

if(jcs:empty($fxp0interface/unit[name=="0"]/family/inet/address/name))
{
        <xnm:error> {
            <message> "fxp0 must have an IP address.";
        }
    }
}
```

Skriptiesimerkki 18. *Commit*-skripti, joka tarkistaa kandidaattikonfiguraatiosta kolme elementtiä.

```
[edit system scripts commit]
root@K6-JUN1# set file basic-sanity-check.slax
```

Terminaaliesimerkki 7. *Commit*-skriptin konfiguraatio.

5.4.2 Junos-laitteen käynnistys ja *commit*-skriptit

Junos-laitteen käynnistäminen luo omanlaisensa tilanteen, joka on syytä ottaa huomioon *commit*-skriptejä kirjoittaessa. Kun Junos-laite käynnistetään, sen täytyy käydä läpi *commit*-prosessi alustaakseen daemonit käytettävää konfiguraatiota varten. Tämä prosessi on muutamaa poikkeusta lukuunottamatta samanlainen kuin normaali *commit*-prosessi. Kaikki tieto laitteesta, kuten alustan vielä alustamattomien komponenttien tila, ei ole tässä vaiheessa käytettävissä. Mikäli *commit*-skripti luo tämän tiedon puuttumisen takia *commit*-virheen, laite käynnistyy ilman konfiguraatiota. Laite pysyy tässä tilassa, kunnes käyttäjä korjaa ongelman.

5.5 Skriptien siirtäminen Juniper-laitteelle

Skriptit voi siirtää verkon kautta Juniper-laitteelle käyttäen joko FTP:tä (*file transfer protocol*) tai SCP:tä (*secure copy protocol*). Nämä tiedostonsiirtoprotokollat edellyttävät

palvelimen, jolta tiedostot haetaan. Juniper SRX210:ssä on kaksi USB-porttia, joiden kautta pystyy myös siirtämään tiedostoja laitteelle.

Tämän työn kannalta yksinkertaisin ratkaisu on käyttää USB-muistitikkua, jolla siirtää USB-portin kautta skriptitiedostot työasemalta SRX210:een. Tämä on huomattavasti hitaampaa kuin esimerkiksi tiedostojen siirtäminen yleisimmille Windows-työasemille. Toisaalta FTP- tai SCP-palvelimen asennus ja käyttöönotto laboratorioluokan koneelle tai omalle kannettavalle vie myös aikaa. Varsinkin yleisessä käytössä olevat laboratorioluokan koneet ovat hankalia, sillä työskennellessään ei voi olla varma pääseekö samalle koneelle, jolle edellisellä kerralla asensi palvelimen.

USB-muistitikkua käyttäessä tarvitaan Junos-laitteella jonkin verran FreeBSD:ssä ja Unixissa käytettäviä komentoja. Tässä yhteydessä hyödyllisiä ovat:

- *ls*
- *mount*
- *cp*

Tiedostojen siirto USB-muistitikulta tapahtuu shell-tilassa. Muistitikku kytketään SRX210:n etupaneelissa olevaan USB-porttiin. Laite ilmoittaa terminaaliin, että USB-porttiin on kiinnitetty USB-muistitikku. Terminaaliin tulee ilmoitus, jossa kerrotaan muistitikun nimi, mihin porttiin se on kytketty, portin SIM, SIMin väylä sekä muuta tietoa.

```
root@King% umass1: Kingston DataTraveler 2.0, rev 2.00/1.10, addr 4

da1 at umass-sim1 bus 1 target 0 lun 0
da1: <Kingston DataTraveler 2.0 PMAP> Removable Direct Access SCSI-0
device
da1: 40.000MB/s transfers
da1: 954MB (1953792 512 byte sectors: 64H 32S/T 954C)
root@King%
```

Terminaaliesimerkki 8. USB-muistitikku kytketään laitteen USB-porttiin.

Olennaista on tietää SIM ja väylä. Tämän jälkeen muistitikku pitää liittää *mount*-komennolla haluttuun väliaikaiseen kansioon. Kansion sisällön saa näkyviin listana *ls*-komennolla.

```
root@King% ls /dev/da*
/dev/da0        /dev/da0s2      /dev/da0s3c     /dev/da0s4a
/dev/da0s1      /dev/da0s2a     /dev/da0s3e     /dev/da0s4c
/dev/da0s1a     /dev/da0s2c     /dev/da0s3f     /dev/da1
/dev/da0s1c     /dev/da0s3      /dev/da0s4      /dev/da1s1
root@King% mount_msdosfs /dev/da1s1 /mnt
root@King% ls /mnt
.Spotlight-V100         Insin??rity?            k5-jun1.txt
.Trashes                hello-world.slax        k5-jun2.txt
._.Trashes              k1-jun1.txt             lammi1
._Insin??rity?          k1-jun2.txt             lopputy?
._mpls.ppt              k1-jun3.txt             mpls.ppt
root@King%
```

Terminaaliesimerkki 9. Komennolla *ls/dev/da** saa selville tiedostopolun oikeaan väylään jonka jälkeen USB-muistitikku voidaan liittää.

Kun muistitikku on liitetty, voidaan siltä kopioida tiedostoja haluttuun kansioon SRX210:llä *cp*-komennolla. Terminaaliesimerkissä 10 kopioidaan *op*-skripti *hello-world.slax* väliaikaisesta kansiosta *op*-skriptien tallennuspaikkaan.

```
root@King% cp /mnt/hello-world.slax /var/db/scripts/op
root@King% ls /var/db/scripts/op
hello-world.slax
root@King%
```

Terminaaliesimerkki 10. Tiedoston kopiointi.

Muistitikun liittämisen purkaminen ennen muistitikun irroittamista tapahtuu *umount*-komennolla. Terminaaliesimerkissä 11 näkyy tämä toimenpide.

```
root@King% umount /mnt
root@pawn1% umass1: at uhub1 port 2 (addr 4) disconnected
(da1:umass-sim1:1:0:0): lost device
(da1:umass-sim1:1:0:0): removing device entry
umass1: detached

root@King%
```

Terminaaliesimerkki 11. Liittämisen purkaminen, jonka jälkeen USB-muistitikku irroitetaan laitteesta.

Huomasin USB-muistitikun käytössä yhden eron kahden eri Junos version välillä. Tammi- ja helmikuussa 2012 kokeilin skriptejä Junosin versiolla 10.0. Myöhemmin keväällä, kun Junos oli päivitetty versioon 11.4, saman USB-muistitikun käyttö oli huomattavasti sujuvampaa. Aiemmalla versiolla muistitikun liittämisen terminaaliesimerkissä 8 näkyvän listauksen syntyyn saattoi mennä 2-4 minuuttiia. Uudemmalla versiolla tämä tapahtui välittömästi.

6 Laboratorioverkko ja skriptien testaaminen

6.1 Valmistelut

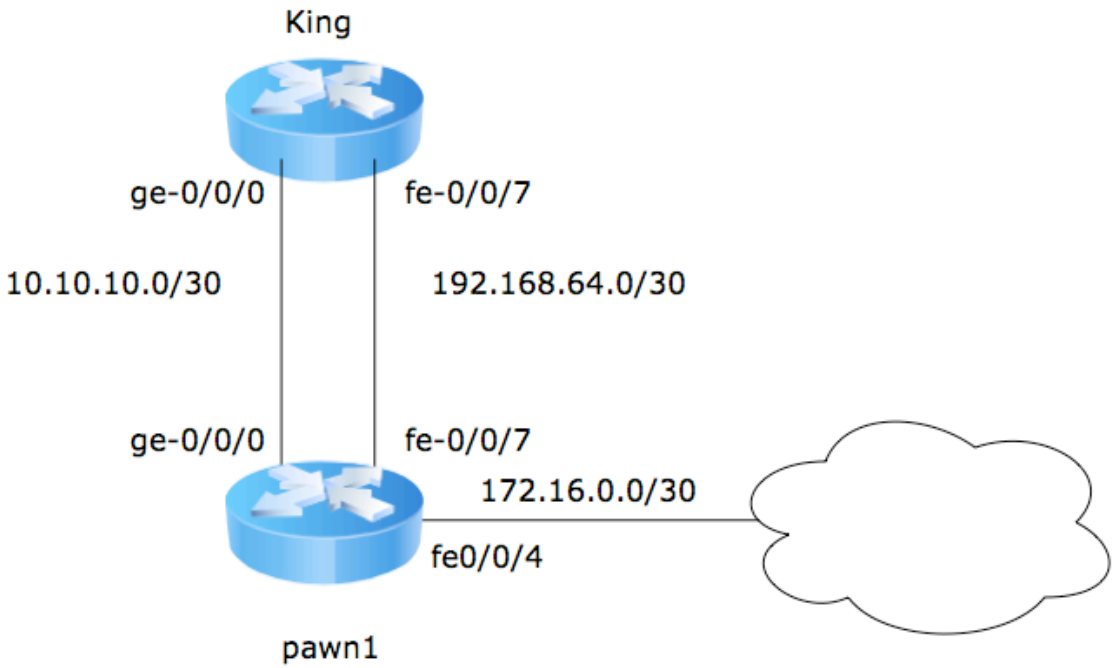
Kokosin laboratorioverkon Metropolian Bulevardin toimipisteen U206-luokkaan. Kiinnitin kahden kaapin 1 räkissä olevan Juniper SRX210-yhdyskäytävän etupaneeliin tarrat, joissa ilmoitin laitteiden olevan insinöörityökäytössä sekä omat yhteystietoni. Syyslukukauden 2011 jälkeen U206 luokan kaikista kaapeista oli irroitettu yksi Juniper SRX210, joten kolmen laitteen sijaan kaapeissa oli vain kaksi Juniperin laitetta. Näin ollen eri kaapeissa olevat laitteet täytyi yhdistää, mikäli tarkoitus oli rakentaa kahta laitetta suurempi verkko pelkillä Juniperin laitteilla.

Resetoin laitteet ensin tehdasasetuksille. Otin tehdasasetuksista joitain turvallisuusasetuksia pois, sillä niillä ei skriptien testaamisen osalta ollut merkitystä ja osa jopa haittaisi laitteiden toimintaa keskenään, ellei asetuksiin olisi tehnyt erikseen poikkeuksia. Tämän jälkeen asetin niille nimet *King* ja *pawn1*, loin kaksi kappaletta samoja käyttäjätunnuksia molempiin ja asetin samassa verkossa olevat IPV4-osoitteet Gigabit-Ethernet-portteihin. Osoitteet on kirjattu taulukkoon 3.

Taulukko 3. Laboratorioverkon IPV4-osoitteet.

Rajapinta	Hostname King	Hostname pawn1
ge-0/0/0	10.10.10.1/30	10.10.10.2/30
fe-0/0/4	N/A	172.16.0.2/30
fe-0/0/7	192.168.64.1/30	192.168.64.2/30

Molempien reitittimien gigabit-ethernet-0/0/0-porttien välissä olevaa verkkoa käytin testaustilanteessa telnet- ja SSH-yhteyden saamiseksi laitteelta toiselle. Laitteiden välillä oli IBGP-naapuruus fast-ethernet-0/0/7-porttien kautta. Lisäksi *pawn1*-laitteelta oli simuloitu yhteys "ulkopuolelle" fast-ethernet-0/0/4-portin kautta. Kuviolla 5 havainnollistetaan laboratorioverkon topologiaa.



Kuvio 5. Laboratorioverkon topologia.

En konfiguroinut kolmatta laitetta kahden Juniper SRX210:n lisäksi, vaikka olin konfiguroinut toiselle laitteelle IP-osoitteen 172.16.0.2, joka topologiassa olikin samassa verkossa kolmannen laitteen kanssa. Testausta varten rajapinta, jolle tämä osoite oli konfiguroitu, piti saada kuitenkin vastaamaan ICMP-paketteihin. Sain tämän aikaan kytkemällä rajapintaan ethernet-kaapelin, jonka toisen pään laitoin kiinni samassa kaapissa olevan HewlettPackardin kytkimen satunnaiseen porttiin. Kuvassa näkyvät laboratorioverkon kytkennät.

Kuva 1. Laboratorioverkon kytkennät. Kaksi alinta laitetta ovat laboratorioverkon Juniper SRX210 -laitteet.

Vuoden 2012 alussa Juniperin laitteissa oli vielä käytössä Junosin versio 10.0. Testasin uudelleen joitain skriptejä vielä huhtikuussa 2012 ja tällöin huomasin, että käyttöjärjestelmäksi oli päivitetty Junosin versio 11.4.

6.2 Skriptien testaaminen

Kokeilin ensin esimerkeissä esiteltyjä skriptejä. Luontevin ensimmäiseksi esimerkiksi oli *op*-skripti *hello-world.slax*, joka näkyy skriptiesimerkistä 16. Skripti toimi ensimmäisellä yrittämällä.

```
[edit]
root@King# set system scripts op file hello-world.slax

[edit]
root@King# commit
commit complete

[edit]
root@King# exit
Exiting configuration mode

root@King> op hello-world.slax
Hello World!
```

Terminaaliesimerkki 12. *Op*-skripti tulostaa tekstin ruudulle.

Seuraavaksi kokeilin skriptiesimerkin 17 *event*-skriptiä. Terminaaliesimerkissä 13 näkyy järjestelmän lokitiedoston merkintä jonka skripti on *commit*-prosessin aikana tehnyt.

```
[edit]
root@King# run show log messages | match cscript
Jan 31 14:19:19  K6-JUN1 cscript: Hello World!
```

Terminaaliesimerkki 13. Lokitiedoston merkintä.

Ensimmäisenä *commit*-skriptinä kokeilin skriptiesimerkkiä 18. SSH-palvelu oli konfiguroituna, mutta käyttätunnus *jnpr* ja fxp0-rajapinnan IP-osoite puuttuivat konfiguraatiosta. Skripti toimi, eikä antanut viedä *commit*-prosessia loppuun, kuten terminaaliesimerkin 14 tuloste osoittaa.

```
[edit system scripts commit]
root@K6-JUN1# commit
error: The jnpr user account must be created.
error: fxp0 must have an IP address.
error: 2 errors reported by commit scripts
error: commit script failure

[edit system scripts commit]
root@K6-JUN1#
```

Terminaaliesimerkki 14. Skriptin antamat virheilmoitukset.

Lisäsin käyttäjätunnuksen ja rajapinnan IP-osoitteen konfiguraatioon. Terminaaliesimerkissä 15 näkyy, kuinka konfiguraatio tehtiin sekä tuloste onnistuneesta *commit*-prosessista.

```
[edit]
root@King# edit system login

[edit system login]
root@King# edit user jnpr

[edit system login user jnpr]
root@King# set class read-only

[edit system login user jnpr]
root@King# set authentication plain-text-password
New password:
Retype new password:

[edit system login user jnpr]
root@King# exit

[edit]
root@King# set interfaces fxp0 unit 0 family inet address 172.0.0.1

[edit]
root@King# commit
commit complete

[edit]
root@King#
```

Terminaaliesimerkki 15. Konfigurointi ja onnistunut *commit*-prosessi.

Kokeilin RPC:tä skriptiesimerkin 19 avulla. *Op*-kriptin tarkoitus on suorittaa toisella laitteella RPC, jolla saadaan selville toisen laitteen Junosin versio. Skripti kysyy ensin käyttäjältä käyttäjätunnuksen ja salasanan, joilla kirjaudutaan toiselle laitteelle. Sen jälkeen avataan SSH-yhteys *jcs:open()*-funktion avulla. Varsinainen tieto haetaan *$remote-results*-muuttujaan *jcs:execute*-funktiolla, joka suorittaa RPC:n.

```
/* haku.slax */
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match / {
    <op-script-results> {
        var $user-name = jcs:get-input( "Enter user-name: " );
        var $password = jcs:get-secret( "Enter password: " );

/* Avaa etäyhteys */

    var $connection = jcs:open( '10.10.10.1', $user-name, $password );

    var $remote-results = jcs:execute( $connection, "get-software in-
formation" );
    <output>'Tulos: ' _  $remote-results;
              }
}
```

Skriptiesimerkki 19. Skripti, joka hakee tietoa toiselta laitteelta.

Skripti edellytti, että käyttäjälle oli luotu *pawn1*-laitteelle SSH-avain. Lisäksi autentikointi piti suorittaa avaamalla SSH-yhteys kerran *King*-laitteelle. Näiden toimenpiteiden jälkeen skripti onnistui suorittamaan RPC:n. Terminaaliesimerkissä 16 skripti on hakenut toiselta laitteelta tiedon, jonka mukaan laitteella on käytössä Junosin versio 10.0.

```
root@pawn1> op haku
Enter user-name: sandels
Enter password:
Tulos:
King
srx210-poe
srx210-poe


junos
JUNOS Software Release [10.0R1.8]

root@pawn1>
```

Terminaaliesimerkki 16. Skriptin tulostama vastaus.

Koska SSH:ta tarvittiin RPC:n suorittamiseen toisella laitteella, niin laitoin *pawn1*-laitteelle skriptiesimerkkinä 8 olleen *commit*-skriptin. Kokeilin skriptiä ensin, kun olin poistanut SSH-palvelun konfiguraatiosta. Skripti ei antanut suorittaa *commit*-prosessia loppuun. Kun SSH-palvelun lisäsi takaisin konfiguraatioon, niin *commit*-prosessin pystyi taas suorittamaan.

```
[edit]
root@pawn1# set system scripts commit file ssh-testi.slax

[edit]
root@pawn1# commit
error: SSH must be enabled.
error: 1 error reported by commit scripts
error: commit script failure

[edit]
root@pawn1# set system services ssh

[edit]
root@pawn1# commit
commit complete

[edit]
root@pawn1#
```

Terminaaliesimerkki 17. SSH:n konfigurointi ja onnistunut *commit*-prosessi.

Konfiguraatioon muutoksia tekeviä skriptejä kirjoittamalla ollaan hieman lähempänä verkonhallinnan käytännöllisiä työkaluja. Kokeilin staattisten reittien luomista skriptiesimerkillä 20. Skripti pyytää staattisen reitin ja *next-hopin* käyttäjältä syötteenä. Se

sijoittaa annetut merkkijonot solmujoukkoon, joka on konfiguraatioon tehtävä muutos. Sen jälkeen skripti avaa *jcs:open()*-funktiolla yhteyden sisäistä RPC:tä varten, jolla konfiguraatiomuutos tehdään. Lopussa on ehtolauseet onnistuneen ja epäonnistuneen *commit*-prosessin varalle. Näiden jälkeen sisäistä RPC:tä varten avattu yhteys suljetaan.

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
match / {
  <op-script-results> {

    var $static-route = jcs:get-input("Enter the static route: ");

    var $next-hop = jcs:get-input("Enter the next-hop: ");

    var $configuration = <configuration> {
                <routing-options> {
                  <static> {
                    <route> {
                      <name> $static-route;
                      <next-hop> $next-hop;
                    }
                  }
                }
              }

    var $connection = jcs:open();

    var $results := { call jcs:load-configuration( $connection,
$configuration ); }

    if( $results//xnm:error ) {
      for-each( $results//xnm:error ) {
        <output> message;
      }
    }

    if( jcs:empty( $results//xnm:error ) ) {
      <output> "Committed without errors.";
    }

    var $close-results = jcs:close($connection);
  }
}
```

Skriptiesimerkki 20. SLAX-skripti *add-route.slax*.

Lisäsin reitin *King*-laitteelta *pawn1*-laitteen ja "pilven" väliseen verkkoon. Skripti suoriutui ilman virheilmoituksia ja tämän jälkeen *pawn1*-laitteen rajapinnan osoite vastasi *ping*-testiin.

```
[edit]
Konow@King# set system scripts op file add-route.slax

[edit]
Konow@King# commit
commit complete

[edit]
Konow@King# run op add-route
Enter the static route: 172.16.0.0/30
Enter the next-hop: 10.10.10.2
Committed without errors.

[edit]
Konow@King# run ping 172.16.0.2
PING 172.16.0.2 (172.16.0.2): 56 data bytes
64 bytes from 172.16.0.2: icmp_seq=0 ttl=64 time=7.907 ms
64 bytes from 172.16.0.2: icmp_seq=1 ttl=64 time=2.158 ms
6^C
--- 172.16.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.158/4.403/7.907/2.510 ms

[edit]
Konow@King#
```

Terminaaliesimerkki 18. Reitin lisäys skriptillä ja onnistunut testaus.

Kokeilin tapahtumakäytäntöjen toimintaa luomalla terminaaliesimerkissä 19 näkyvän tapahtumakäytännön, joka suorittaa aiemmin testatun *hello-world.slax*-skriptin. Käynnistäväksi tapahtumaksi konfiguroin käyttäjän sisäänkirjautumisen. Kun käyttäjä kirjautuu sisään, niin kirjautumistapahtuman tiedoista tarkistetaan onko käyttäjätunnus "Konow". Toimenpiteenä suoritetaan *hello-world.slax*-skripti, jolle on määritelty XML-kohdetiedosto. Tiedostoon kirjoitetaan skriptin tuloste. Kohdetiedosto luodaan *local*-kohteeseen, joka määritellään erikseen tapahtuma-asetusten *destinations*-elementin sisällä konfiguraatiossa. Kohde on *var/tmp/*-kansio.

```
[edit]
Konow@King#run show configuration

[...]

    policy hello-login {
        events ui_login_event;
        attributes-match {
            ui_login_event.username matches "^Konow$";
        }
        then {
            event-script hello-world.slax {
                output-filename hello-world-output;
                destination local;
                output-format xml;
            }
        }
    }

[...]

    event-script {
        file user-logout.slax;
        file hello-world.slax;
        file log-hello-world.slax;
    }
    destinations {
        local {
            archive-sites {
                var/tmp;
            }
        }
    }
}

[...]

[edit]
Konow@King#
```

Terminaaliesimerkki 19. Tapahtumakäytäntö, jolle määritelty *op*-skripti.

Terminaalisesimerkistä 20 nähdään, että skripti loi *var/tmp/*-kansioon tiedoston jokaisen uudelleenkirjautumisen jälkeen. Terminaalisesimerkin 20 neljänneltä riviltä alkavat viisi listattua tiedostoa ovat tapahtumakäytännön luomia.

```
Konow@King> file list /var/tmp/

/var/tmp/:
King_hello-world-output_20120416_092711
King_hello-world-output_20120416_093249
King_hello-world-output_20120416_095442
King_hello-world-output_20120416_095839
King_hello-world-output_20120416_100629
cleanup-pkgs.log
dfwc-anaidx-BSDkXM
dfwc-anaidx-zQ0Bbs
dfwc-iflidx-EMaA3Y
dfwc-vlanidx-0jBEgc
dfwc-vlanidx-ftl7qJ
eedebug_bin_file
event_tags.php
flowd_octeon_hm.core-tarball.0.tgz
flowd_octeon_hm.core-tarball.1.tgz
flowd_octeon_hm.core-tarball.2.tgz
flowd_octeon_hm.core-tarball.3.tgz
flowd_octeon_hm.core-tarball.4.tgz
flowd_octeon_hm.core.0.gz

---(more)---[abort]

Konow@King>
```

Terminaaliesimerkki 20. Kansion */var/tmp/* sisältö.

Edellinen tapahtumakäytäntö käynnisti *op*-skriptin, mutta myös pelkillä tapahtumakäytännön asetuksilla pystytään suorittamaan komentoja. Tapahtumia voi myös luoda itse. Terminaaliesimerkissä 21 näkyy tapahtumkäytäntö, joka suorittaa toimenpiteen tiettyyn kellonaikaan. Kellonaika määritellään tapahtumaksi *generate-event*-elementin alle. Luotua tapahtumaa voidaan tämän jälkeen käyttää samalla tavoin, kuin Junosin vakiotapahtumia. Terminaaliesimerkissä 21 tapahtumaa käytetään samankaltaisesti, kuin terminaaliesimerkissä 19, eli luomaan XML-tiedosto.

```
[edit event-options policy tallenna-aika then execute-commands]
Konow@King#run show configuration

[...]

    generate-event {
        aika time-of-day "10:44:00 +0000";
    }

[...]

    policy tallenna-aika {
        events aika;
        then {
            execute-commands {
                commands {
                    "show system uptime";
                }
                output-filename aika;
                destination local;
                output-format xml;
            }
        }
    }

[...]

    destinations {
        local {
            archive-sites {
                var/tmp;
            }
        }
    }
}

[...]

[edit event-options policy tallenna-aika then execute-commands]
Konow@King#
```

Terminaaliesimerkki 21. Tapahtumakäytäntö, jolle ei ole määritelty suoritettavaa skriptiä.

Terminaaliesimerkkissä 21 näkyvä tapahtumakäytäntö luo tiedoston, johon on tallennettu terminaaliesimerkissä 22 näkyvän, tietoja käyttöjärjestelmän kellonajoista ja viimeisestä käynnistyksestä antavan komennon tuloste XML-muodossa. Terminaaliesimerkistä 23 nähdään, että viisi sekuntia sen jälkeen kun kello ylitti ajan 10:44, skripti loi *var/tmp/*-kansioon XML-tiedoston.

```
Konow@King# run show system uptime
Current time: 2012-04-16 10:34:56 UTC
System booted: 2012-04-16 08:55:51 UTC (01:39:05 ago)
Protocols started: 2012-04-16 08:59:46 UTC (01:35:10 ago)
Last configured: 2012-04-16 10:23:26 UTC (00:11:30 ago) by Konow
10:34AM  up 1:39, 1 user, load averages: 0.01, 0.02, 0.03
```

Terminaaliesimerkki 22. Komennon *show system uptime* tuloste ennen tapahtumakäytännön toteutumista.

```
[edit event-options policy tallenna-aika then execute-commands]
Konow@King# run file show /var/tmp/King_aika_20120416_104405
<?xml version="1.0" encoding="us-ascii"?>
<junoscript                  xmlns="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:junos="http://xml.juniper.net/junos/11.4R2/junos"   schemaLocati-
on="http://xml.juniper.net/junos/11.4R2/junos   junos/11.4R2/junos.xsd"
os="JUNOS" release="11.4R2.14" hostname="King" version="1.0">
<!-- session start at 2012-04-16 10:44:05 UTC -->
<!-- No zombies were killed during the creation of this user interface
-->
<!-- user root, class super-user -->
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.4R2/junos">
<system-uptime-information
xmlns="http://xml.juniper.net/junos/11.4R2/junos">
<current-time>
<date-time  junos:seconds="1334573045">2012-04-16  10:44:05  UTC</date-
time>
</current-time>
<system-booted-time>
<date-time  junos:seconds="1334566551">2012-04-16  08:55:51  UTC</date-
time>
<time-length junos:seconds="6494">01:48:14</time-length>
</system-booted-time>
<protocols-started-time>
<date-time  junos:seconds="1334566786">2012-04-16  08:59:46  UTC</date-
time>
<time-length junos:seconds="6259">01:44:19</time-length>
</protocols-started-time>
<last-configured-time>
<date-time  junos:seconds="1334573005">2012-04-16  10:43:25  UTC</date-
time>
<time-length junos:seconds="40">00:00:40</time-length>
<user>Konow</user>
</last-configured-time>
<uptime-information>
<date-time junos:seconds="1334573045">10:44AM</date-time>
<up-time junos:seconds="6524">1:48</up-time>
<active-user-count junos:format="1 user">1</active-user-count>
<load-average-1>0.30</load-average-1>
<load-average-5>0.16</load-average-5>
<load-average-15>0.08</load-average-15>
</uptime-information>
</system-uptime-information>
</rpc-reply>
[edit event-options policy tallenna-aika then execute-commands]
Konow@King#
```

Terminaaliesimerkki 23. XSL-tiedosto.

Kokeilin myös reaaliajassa käyttäjän komentoja lukevaa skriptiesimerkkiä 21. Terminaaliesimerkin 24 mukaan konfiguroituna sen pitäisi lopettaa käyttäjän sessio, mikäli käyttäjä antaa komennon *clear bgp neighbor*. Tämä komennon jälkeen laite alkaa muodostamaan uudestaan BGP-naapuruuksiaan ja voisi tuotantokäytössä aiheuttaa suurenkin katkoksen liikenteessä. Yrityksistä huolimatta en saanut skriptiä toimimaan enkä löytänyt tälle syytä.

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";
import "../import/junos.xsl";
/* Username argument */
param $username;
match / {
    /* Logout the user */
    var $command = <command> "request system logout user " _ $userna-
me;

    var $results = jcs:invoke( $command );

    /* If any errors occurred then report them to the syslog */
    if( $results/..//xnm:error ) {
        for-each( $results/..//xnm:error ) {
            expr jcs:syslog( "external.error", "Error logging out ",
$username,
": ", message);
        }
    }
    else {
        var $message = "Logged out " _ $username _ " for clearing all
BGP
neighbors.";
        expr jcs:syslog( "external.notice", $message );
    }
}
```

Skriptiesimerkki 21. Skripti, joka lopettaa käyttäjän session.

```
Konow@King> show configuration

[...]

policy logout-user {
        events ui_cmdline_read_line;
        attributes-match {
            ui_cmdline_read_line.command matches "^(run )?clear bgp
neighbor$";
            ui_cmdline_read_line.command matches "^clear bgp neigh-
bor$";
        }
        then {
            event-script user-logout.slax {
                arguments {
                    username "{$$.username}";
                }
            }
        }
    }

[...]

Konow@King> clear bgp neighbor
Cleared 1 connections
```

Terminaaliesimerkki 24. Tapahtumakäytäntö, jonka laukaisee tietty komento *clear bgp neighbor* -komento.

7 Yhteenveto

Dokumentteja ja Juniperin laitteita tutkimalla tekniikka Junos-skriptien taustalla tuli työn aikana hyvin tutuksi. XML:n ja siihen liittyvien yksityiskohtien sekä oikeilla laitteilla kulkevien viestien tutkiminen auttoi ymmärtämään, että miten skriptit toimivat OSI-mallin ylimmillä tasoilla. XSLT jäi vähälle huomiolle, sillä tutkin ja kirjoitin skriptejä SLAXilla.

Skriptien kirjoittaminen ja testaus oli paikoitellen todella hankalaa. Testaustilanteessa skriptin tallentaminen ulkoiselle USB-muistille, muistin liittäminen hakemistopuuhun ja vanhan skriptin uudella korvaaminen tuntui erittäin työläältä. Hankaluus korostuu varsinkin, jos vertaa työtapaa nykyaikaisiin ohjelmistokehitysympäristöihin. SLAX-projektin kotisivuilta löytyi mielenkiintoinen *slaxproc*-kirjasto, jolla pystyy esimerkiksi testaamaan SLAX-skriptejä syntaksivirheiden varalta ja muuntamaan SLAX-skriptejä XSLT:ksi ja päinvastoin. Yritin asentaa kirjastoa kahdelle eri käyttöjärjestelmälle: Mac OS X:lle ja Ubuntulle. Mac OS X:llä, joka on Junosin tavoin FreeBSD-pohjainen, minulla oli ongelmia kääntäjän kanssa. Ubuntulla tarvittavien kirjastojen lataaminen olisi teettänyt lisätyötä epävarman tuloksen eteen, joten en käyttänyt enempää työtunteja siihen. Sain kuitenkin laboratoriossa kokeiltua yksinkertaisia skriptejä, joissa onnistuin todentamaan käytännössä työn teoriaosuuden käsittelemiä asioita. Lähdin tutkimaan aihetta teorian kautta ja käytännön tasolla testaukset jäivät "pintaraapaisuiksi" laajasta aiheesta.

Työn aikana muodostui selvempi käsitys Junos-skriptauksen mahdollisuuksista. Lyhyt työkokemus verkonhallinnasta on luonut tietynlaisen käsityksen siitä, että minkälaisia työkaluja "kentällä" tarvitaan. Verkkolaitteiden konfigurointi suuressa verkossa voi olla helpompaa esimerkiksi erillisen hallintapalvelimen kautta, kuin jokaiselta laitteelta erikseen ajettavien skriptien kautta. Erillisellä hallintapalvelimella turvallisuus ja työkalujen hallinta toteutettaisiin keskitetysti ilman että resursseja kulutettaisiin laitteiden ohjelmistopäivitysten lisäksi skriptien päivittämiseen. Junos-skriptit voivat kuitenkin lukea laitteen konfiguraation läpi nopeasti ja ne kommunikoivat suoraan Junosin daemonien kanssa. Tämän onkin Junos-skriptien vahvuus, sillä näiden toimintojen mahdollistaminen ulkopuolisella palvelimella olisi hankalampaa.

Työtä olisi voinut jatkaa tutkimalla tietokantojen käytön mahdollisuuksia skriptauksessa. Yritin työn alkuvaiheessa pitkään kirjoittaa skriptiä joka olisi lukenut toiselta laitteel-

ta valikoidusti tietoja ja käyttänyt niitä konfiguraatiossa. SLAXin avulla tietoja pystyi lukemaan tiedostosta, mutta tietyn arvon valinta ja tulosten käsittely halutulla tavalla ei onnistunut. Yksi monimutkainen mahdollisuus olisi ollut kirjoittaa Perl-skripti tiedoston käsittelyyn. Ongelmaan löytyi kuitenkin Juniper Booksin Day One -opassarjasta mahdollinen ratkaisu, joka oli konfiguraatioryhmien käyttö tietokantana. Niiden kanssa yhden Junos-laitteen konfiguraatioon voisi rakentaa "tietokannan", jolta rajoittamaton määrä Junos-laitteita voisi RPC:n avulla hakea tietoja omaan konfiguraatioonsa. Yksi laite toimisi näin tietokantapalvelimena ja voisi samalla toimia tavanomaisessa käyttötarkoituksessaan. En ehtinyt kuitenkaan saada skriptiä testausvaiheeseen työn aikana.

Toteuttamattomista ideoista huolimatta työn teoriaosuus käsittelee tiiviisti suuren osan Junos-skriptuksen keskeisistä aiheista. Junosia tarkemmin käsittelevää aineistoa ei löytynyt suomeksi ollenkaan, lukuunottamatta Theseus-tietokannasta löytyviä insinööritöitä. Junos-käyttöjärjestelmästä ja sen automatisoinnista työ tarjoaa todennäköisesti paljon uutta suomenkielistä tietoa.

Lähteet

1 Increasing Network Availability with Automated Scripting White Paper. 2008. PDF-tiedosto. Juniper Networks.

2 Call, Curtis. 2008. This Week: Applying Junos Automation. PDF-tiedosto. Juniper Networks Books.

3 Garret, Aviva. 2006. JUNOS Cookbook. O'Reilly.

4 Gadecki, Cathy. Scruggs, Michael. 2011. Day One: Exploring the Junos CLI. PDF-tiedosto. Juniper Networks Books.

5 Lehey, Greg. 2003. The Complete FreeBSD, 4th Edition. Sebastopol, CA: O'Reilly.

6 SRX210 Services Gateway Hardware Guide. 2011. PDF-tiedosto. Juniper Networks.

7 Hunter, David ...[et al.]. 2007. Beginning XML. Indianapolis, IN : Wrox.

8 What is XML-RPC?. Verkkosivu. <http://xmlrpc.scripting.com/>. Luettu 15.4.2012.

9 XML Path Language (XPath) Version 1.0. Verkkodokumentti. <http://www.w3.org/TR/xpath/>. Luettu 15.4.2012.

10 Nykänen, Ossi. 2001. XML. Jyväskylä: Docendo

11 libslax. Verkkosivu. <http://code.google.com/p/libslax/> 15.4.2012

12 Schulman, Jeremy. Call, Curtis. 2011. This Week: Mastering Junos Automation Programming. PDF-tiedosto. Juniper Networks Books.

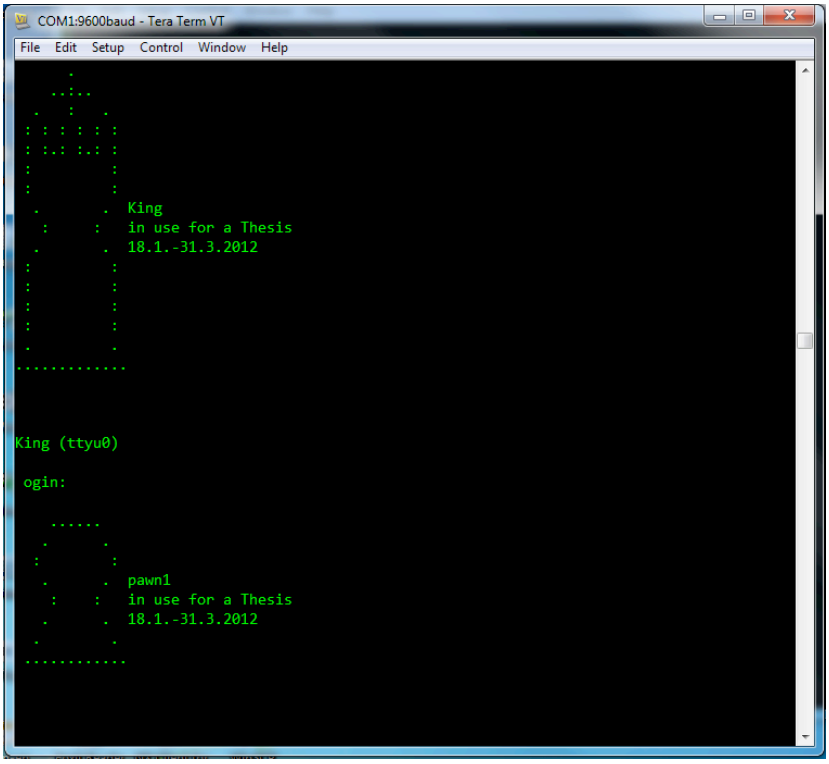
13 Call, Curtis. 2011. This Week: Junos Automation Reference for SLAX 1.0. PDF-tiedosto. Juniper Networks Books.

SLAXin operaattorit

Operaattori	Toiminta	Syntaksi
+	Yhteenlaskuoperaattori laskee kaksi numeroa yhteen.	`$nro1 + $nro2;`
-	Vähennyslaskuoperaattori vähentää numeron toisesta.	`$nro1 - $nro2;`
*	Kertolasku kertoo kaksi numeroa.	`$nro1 * $nro2;`
div	Jakolasku jakaa numeron toisella.	`$nro1 div $nro2;`
mod	Jakojäännös palauttaa kahden numeron jakojäännöksen.	`$nro1 mod $nro2;`
==	Vertailuoperaattori, "on yhtä kuin".	`$nro1 == $nro2;`
!=	Yhtäläisyyden vertailuoperaattorin vastakohta, "ei ole yhtä kuin".	`$nro1 != $nro2;`
<	Vertailuoperaattori, "on pienempi kuin".	`$nro1 < $nro2;`
<=	Vertailuoperaattori, "on pienempi tai yhtä suuri kuin".	`$nro1 <= $nro2;`
>	Vertailuoperaattori, "on suurempi kuin".	`$nro1 > $nro2;`
>=	Vertailuoperaattori, "on suurempi tai yhtä suuri kuin".	`$nro1 >= $nro2;`
()	Sulut toimivat samoin kuin sulut matemaattisissa laskuissa.	`$nro1 + ($nro2 + nro3);`
&&	Looginen JA-operaattori.	`$nro1 && $nro2;`
||	Looginen TAI-operaattori.	`$nro1 || $nro2;`
_	Alaviivalla yhdistetään merkkijonot keskenään.	`$mjono1 _ $mjono2;`
|	Solmujoukkojen liitto.	`$sjoukko1 | $sjoukko2;`

:=	RTF:stä solmjoukoksi kääntäminen.	var $sjoukko := {
		...sisältö...
		}

Bannerit


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help
King
in use for a Thesis
18.1.-31.3.2012
King (ttyu0)
ogin:
pawn1
in use for a Thesis
18.1.-31.3.2012