TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Sami Oksanen

XEN-VIRTUAALIKONEYMPÄRISTÖN HALLINTAKOMPONENTTI

Työn ohjaaja Lehtori Jari Mikkolainen
Työn teettäjä Axxion Oy, valvojana Ville Karjalainen
Jyväskylä 2008

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Oksanen, Sami
Xen-virtuaalikoneympäristön hallintakomponentti
Tutkintotyö 44 sivua + 2 liitesivua
Työn ohjaaja Lehtori Jari Mikkolainen
Työn teettäjä Axxion Oy, valvojana Ville Karjalainen
Toukokuu 2008
Hakusanat ohjelmointi, Perl, Xen, XML-RPC, keskitetty hallinta

TIIVISTELMÄ

Insinöörityön tarkoituksena oli suunnitella ja toteuttaa Axxion Oy:lle palvelinkomponentti, joka tulisi olemaan osa laajempaa Xen-virtuaalikoneympäristön hallintaohjelmistoa. Työssä keskityttiin komponentin suunnitteluun ja toteutukseen, ja se vastaa hyvin ohjelmistoinsinöörin tyypillisiä työtehtäviä.

Työn tavoitteena oli luoda komponentti, jonka avulla voidaan ohjata ja seurata yhdestä tai useammasta fyysisestä palvelimesta koostuvaa Xen-virtuaalikoneympäristöä. Toteutettava komponentti päivittäisi jatkuvasti virtuaalikoneiden tilan tietokantaan sekä suorittaisi tietokannassa olevaan komentojonoon lisätyt komennot.

Työn toteutus tuli ajankohtaiseksi Axxion Oy:n siirtyessä käyttämään useampaa fyysistä palvelinta, joissa jokaisessa käytettiin Xen-virtuaalikonemoottoria. Koska useamman fyysisen palvelimen ylläpitäminen käsin olisi ollut hankalaa, eikä sopivaa valmista ratkaisua keskitettyyn hallintaan löytynyt, päätettiin vaadittava ohjelmisto toteuttaa itse.

Komponentin suunnitteluvaiheessa tutkittiin erilaisia toteutustapoja sekä tutustuttiin Xen-virtuaalikonemoottorin ohjaustekniikoihin. Valittavien tekniikoiden tuli olla pitkäikäisiä ja hyvin tuettuja, jotta säästyttäisiin komponentin käyttämien kirjastojen vaihtamiselta mahdollisimman pitkään. Komponentti toteutettiin Axxion Oy:n yleisesti käyttämällä Perl-ohjelmointikielellä.

Työn tulos on yksinkertainen ja käyttökelpoinen Xen-virtuaalikoneympäristön valvontaan ja hallintaan tarkoitettu komponentti, jolla on laajat jatkokehitysmahdollisuudet. Komponenttia käytetään osana Axxion Oy:n VServer Manager -ohjelmistoa.

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software engineering
Oksanen, Sami
Component of centralized management for Xen-based virtual system
Engineering Thesis 44 pages + 2 appendices
Thesis Supervisor Senior Lecturer Jari Mikkolainen
Commissioning Company Axxion Oy. Supervisor: Ville Karjalainen
May 2008
Keywords programming, Perl, Xen, XML-RPC, centralized management

ABSTRACT

The goal of this bachelor's thesis was to design and implement a server component for Axxion Oy for use in a larger software project for managing Xen-based virtual machine networks. The thesis focuses mainly on designing and implementing the component, which it is a quite typical task for a software engineer.

The purpose of the server component is to supervise and control a Xen-based virtual machine network that consists of one or more physical servers. The component continuously updates the status of each virtual machine to the database and performs tasks assigned to the command queue in the database.

The need for the server component arose when Axxion Oy added two physical servers to their Xen-based virtual system. Because managing the system manually would have been difficult and suitable existing solutions for centralized management were not found, the company decided to build custom software to handle the task.

During the component's design process, various methods of controlling the Xen-system and possible implementation methods were evaluated. The chosen methods were required to be robust and properly supported to ensure the stability of the design of the component. The component was implemented using Perl language, which is the primary programming language used by Axxion Oy.

The result of this thesis is a simple, useful and extensible component for managing the Xen-based virtual machine network. The component is used as a part of Axxion Oy's VServer Manager application.

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

ALKUSANAT

Tämä tutkintotyö on toteutettu Axxion Oy:lle vuoden 2008 alkupuoliskolla. Halusin saada tutkintotyön avulla uudenlaisia kokemuksia ohjelmistoprojektien toteutuksesta sekä perehtyä syvällisemmin Xen-virtualisointitekniikkaan.

Tahtoisin kiittää Axxion Oy:n toimitusjohtajaa Jukka Raimovaaraa mahdollisuudesta tähän tutkintotyöhön, sillä työn ansiosta pääsin tutustumaan syvemmin minua kiinnostaviin aiheisiin. Haluaisin kiittää myös Axxion Oy:n työntekijöitä Ville Karjalaista sekä Pekka Vesalaa asiantuntija-avusta.

Jyväskylässä 30. toukokuuta 2008

Sami Oksanen

TERMIT JA LYHENTEET

CLVM	Klusteroitu loogisten levyjen hallintajärjestelmä, jonka avulla useampi lohkolaite voidaan yhdistää yhdeksi isoksi lohkolaitteeksi ja jakaa sitä dynaamisesti pienempiin osiin. Klusteroinnin ansiosta useampi järjestelmä kykenee hallitsemaan lohkolaitteita yhtäaikaisesti.
Domain 0	Etuoikeutettu virtuaalikone, jonka virtuaalikonemoottori käynnistää. Saa vakiona suoran pääsyn koko laitteistoon. Käytetään muiden samalla fyysisellä palvelimella olevien virtuaalikoneiden (Domain U) konfigurointiin ja hallintaan /1/.
Domain U	Fyysisellä palvelimella oleva virtuaalikone, joka konfiguroidaan ja käynnistetään etuoikeutetusta virtuaalikoneesta (Domain 0) käsin. Virtuaalikoneelle voidaan sallia yksinoikeutettu pääsy tietylle laitteistolle.
Fyysinen palvelin	Vähintään emolevystä, suorittimesta ja keskusmuistista koostuva kokonaisuus.
Guest	Domain U -virtuaalikoneessa ajettava käyttöjärjestelmä.
Host	Domain 0 -virtuaalikoneessa ajettava käyttöjärjestelmä.
iSCSI	Protokolla, jonka avulla lohkolaitteita voidaan jakaa IP-verkossa.
Lohkolaite	Laite, jota käytetään datan tallentamiseen, yleisimmin kiintolevy.
Paravirtualisointi	Emuloivaa laitteistovirtualisointia parempi virtualisointitekniikka, jossa virtualisoivan käyttöjärjestelmän emuloinnin sijaan muokataan virtualisoitavan käyttöjärjestelmän rakennetta toimimaan virtuaalikonemoottorin käskykannalla /1/.

Resurssit	Levytilan, keskusmuistin, verkon sekä prosessoriajan käyttöaste.
UUID	*(engl. Universally Unique Identifier)*. Standardi yksilöllisten 128-bittisten tunnisteiden luomiseen, jossa kahden samanlaisen tunnisteen esiintyminen on erittäin epätodennäköistä. /7/
Virtuaalikone	Yhdellä fyysisellä palvelimella voi olla useampia toisistaan eristettyjä virtuaalikoneita. Virtuaalikone näyttää sisältä käsin normaalilta fyysiseltä palvelimelta.
Virtuaalikonemoottori	*(engl. Hypervisor)*. Ylimmällä suojaustasolla ajettava järjestelmä, joka hallitsee virtuaalikoneiden laitteistonkäyttöä /1/. Virtuaalikonemoottori toimii virtualisointikerroksena fyysisen laitteiston ja virtuaalikoneiden välissä.
Virtuaalikoneympäristö	Yhdestä tai useammasta fyysisestä palvelimesta ja niissä ajettavista virtuaalipalvelimista koostuva yhtenäisyys.
Virtuaalipalvelin	Virtuaalikoneesta ja siinä ajettavasta käyttöjärjestelmästä muodostuva kokonaisuus.
Xen	Paravirtualisoiva virtuaalikonemoottori, jonka päällä Domain 0 ja Domain U:t ajetaan. Tukee nykyään myös (käytettävästä prosessorista riippuen) täyttä virtualisointia, jossa Domain U:ssa ajettavaa käyttöjärjestelmää ei tarvitse muokata. /1/.

SISÄLLYSLUETTELO

1 JOHDANTO

Insinöörityön tarkoituksena on suunnitella ja toteuttaa Axxion Oy:lle Xen-virtuaalikoneympäristön valvontaan ja hallintaan tarkoitettu palvelinkomponentti. Axxion Oy on vuonna 2005 perustettu kotimaisessa yksityisomistuksessa oleva yritys, joka tarjoaa asiakkailleen luotettavat ja joustavat hosting-palvelut. Hosting-palveluihin kuuluvien verkkotunnus-, kotisivu- ja sähköpostipalveluiden lisäksi Axxion Oy tarjoaa asiakkailleen myös virtuaalipalvelimia sekä toteuttaa yksilöllisiä web-pohjaisia ohjelmistoprojekteja.

Työn toteutus tuli ajankohtaiseksi Axxion Oy:n laajentaessa yhdestä fyysisestä palvelimesta koostuvaa virtuaalikoneympäristöä kahdella uudella palvelimella. Ympäristön hallinta haluttiin keskittää yhteen työkaluun, eikä tehtävään löytynyt sopivaa valmista ratkaisua.

Työssä toteutettava palvelinkomponentti tulee olemaan osa laajempaa Axxion VServer Manager -sovellusta, jonka avulla voidaan ohjata ja seurata yhdestä tai useammasta fyysisestä palvelimesta koostuvaa Xen-virtualisointitekniikkaan perustuvaa virtuaalikoneympäristöä.

Työ käsittää esitutkimuksen, suunnittelun ja toteutusvaiheen. Työssä toteutettavan komponentin integrointitestaus toteutetaan myöhemmin, koska laajemman sovellusprojektin muiden osien toteutus on vielä kesken. Tämän vuoksi integrointitestausta ei käsitellä tässä dokumentissa.

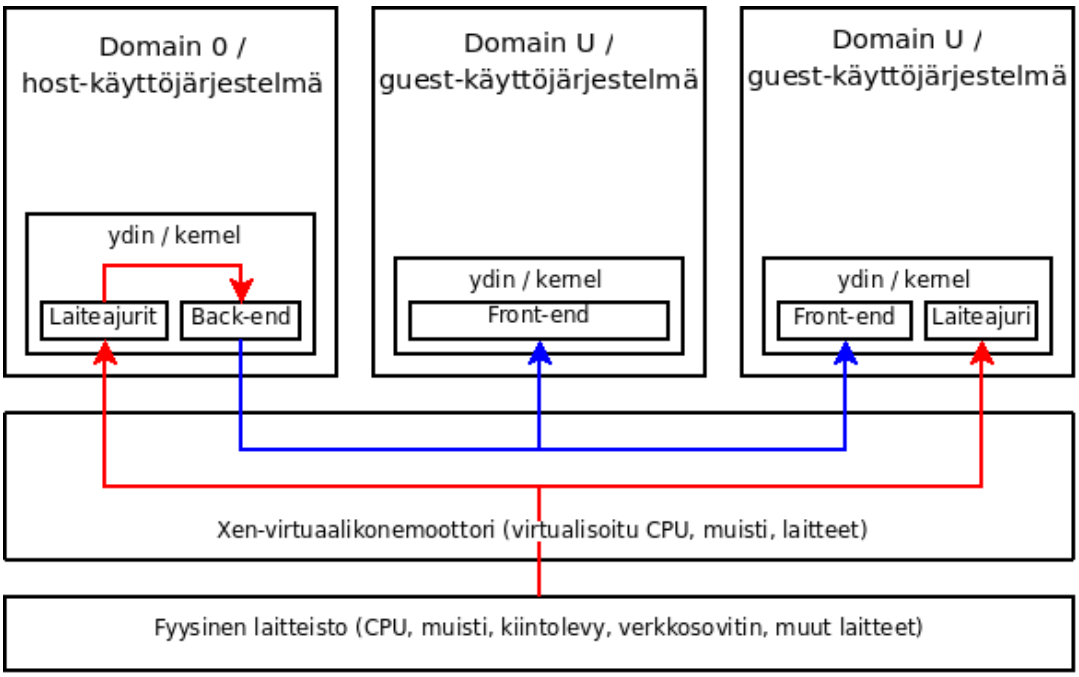
Palvelinkomponentin ohjelmointi on toteutettu Perl-kielellä ja toteutuksessa on käytetty hyväksi valmiita XML-RPC- sekä tietokantakirjastoja. Dokumentissa esitetyt ohjelmakoodiesimerkit on erotettu muusta sisällöstä `Courier`-fontilla ja niiden tarkoitus ja toiminta käydään läpi ennen listausta.

2 XEN

Xen on ilmainen Cambridgen yliopistossa kehitetty virtuaalikonemoottori, joka tukee tämän raportin kirjoittamishetkellä ainakin IA-32-, x86-, x86-64-, IA-64- ja PowerPC 970 -arkkitehtuureja. Sen avulla voidaan ajaa yhdellä fyysisellä palvelimella useampia toisistaan eristettyjä virtuaalikoneita, joissa jokaisessa on oma käyttöjärjestelmänsä. Virtualisoinnin ansiosta virtuaalikoneiden toiminta on toisistaan riippumatonta ja jokaisen virtuaalikoneen ajoa pystytään ohjaamaan erikseen. Esimerkiksi yksittäisen virtuaalikoneen ajo pystytään tauottamaan hetkeksi ja jatkamaan sen ajoa myöhemmin, tai kone voidaan siirtää kokonaan toiselle samassa virtuaalikoneympäristössä sijaitsevalle fyysiselle palvelimelle. Virtuaalikone voidaan siirtää myös sen ollessa käynnissä, jolloin kontrolliyhteys siihen on poikki parhaassa tapauksessa vain 60 millisekuntia.

2.1 Xen-järjestelmän rakenne

Xen-järjestelmä rakentuu alimmalla tasolla olevasta ja etuoikeutetuimmasta virtuaalikonemoottorista sekä sen päällä olevista virtuaalikoneista ja niissä ajettavista käyttöjärjestelmistä. Yksi virtuaalikoneista toimii etuoikeutettuna ja se käynnistetään aina virtuaalikonemoottorin kanssa. Etuoikeutetusta virtuaalikoneesta käytetään nimitystä 'Domain 0' ja sen päällä ajettavasta käyttöjärjestelmästä 'host'. Muista samalla fyysisellä palvelimella sijaitsevista virtuaalikoneista käytetään termiä 'Domain U' sekä niissä ajettavista käyttöjärjestelmistä 'guest'. Kuvassa 1 on esimerkki Xen-järjestelmästä, jossa on yksi Domain 0 sekä kaksi Domain U:ta, joista toiselle on annettu suora pääsy tiettyyn laitteistoon. Kuvassa punainen viiva edustaa suoraa yhteyttä laitteistoon ja sininen viiva kuvaa virtualisoitua yhteyttä.



Kuva 1 Xen-järjestelmän arkkitehtuuri

Domain 0 -virtuaalikoneessa ajettavalla host-käyttöjärjestelmällä on suora pääsy fyysisen palvelimen laitteistoon ja erityisoikeudet virtuaalikonemoottorin hallintaan. Järjestelmän ylläpitäjä voi host-käyttöjärjestelmän kautta käynnistää ja hallita muita virtuaalikoneita.

Domain 0 -virtuaalikoneessa voidaan tällä hetkellä käyttää hostina vain GNU/Linux-, NetBSD- tai Solaris-käyttöjärjestelmää, jonka ydin on muokattu tukemaan Xen-virtuaalikonemoottoria, mutta muiden virtuaalikoneiden käyttöjärjestelmien suhteen ei ole rajoituksia.

Xenin versiosta 3.0 lähtien Domain U -virtuaalikoneissa on voitu ajaa Unix-johdannaisten käyttöjärjestelmien lisäksi myös muita käyttöjärjestelmiä, kuten Microsoft Windowsia, mikäli fyysisen palvelimen prosessori sisältää joko Intel VT-x- tai AMD-V-virtualisointilaajennuksen. /2/

2.2 Virtuaalikonemoottori

Virtuaalikonemoottori on Xen-järjestelmän pääkomponentti, jonka tehtävänä on hallita ja jakaa järjestelmän muistia, varata resursseja uusille virtuaalikoneille sekä jakaa muita järjestelmäresursseja virtuaalikoneiden kesken. Lisäksi

virtuaalikonemoottori huolehtii Domain 0:n käynnistämisestä.

Virtuaalikonemoottori sijaitsee palvelimen fyysisten komponenttien ja palvelimella olevien käyttöjärjestelmien välissä ja tarjoaa käyttöjärjestelmille virtuaalikoneen, joka näyttää pääosin samalta kuin aito fyysinen palvelin. /4/

2.3 Domain 0

Domain 0 on fyysisellä palvelimella sijaitsevan virtuaalikoneympäristön ylläpitämiseen tarkoitettu etuoikeutettu virtuaalikone, jolla on erityisoikeuksia muihin samalla fyysisellä palvelimella sijaitseviin virtuaalikoneisiin nähden. Domain 0:ssa ajettavassa host-käyttöjärjestelmässä on tarkoitukseen erityisesti käännetty ydin, jossa on mukana tuki Xen-virtuaalikonemoottorille sekä erityinen backend-ajuri laitteiston virtualisoimiseksi Domain U -virtuaalikoneille.

Host-käyttöjärjestelmällä ohjataan virtuaalikonemoottoria erityisen xend-daemonin kautta. Tällä esimerkiksi luodaan uusia Domain U -virtuaalikoneita ja hallitaan niiden ajoa. Jos luotava Domain U käyttää massamuistinaan fyysisen palvelimen lohkolaitteita, niin tarvittavien levyosioiden tai levyimage-tiedostojen luonti on tehtävä host-käyttöjärjestelmässä.

Domain 0 -virtuaalikoneella on suora pääsy fyysisen palvelimen laitteistoon. Sen yhtenä tehtävänä on jakaa verkko- ja lohkolaitteet Domain U -virtuaalikoneille Xenin backend-ajurin avulla. Koska Domain U -virtuaalikoneet käyttävät vain virtuaalikonemoottorin tarjoamia laitteistorajapintoja erityisten frontend-ajurien avulla, on kaikki fyysisen palvelimen laitteistoajurit sisällytettävä host-käyttöjärjestelmän ytimeen. /1; 2/

2.4 Domain U

Domain U on normaali virtuaalikone, jolla ei yleensä ole suoraa pääsyä fyysisen koneen laitteistoon, kuten lohkolaitteisiin, verkko- ja äänikortteihin tai näytönohjaimeen, vaan se käyttää Domain 0 -virtuaalikoneen tarjoamia virtuaalilaitteita erityisen frontend-ajurin kautta. Tarvittaessa on mahdollista sallia

Domain U -virtuaalikoneelle yksinoikeutettu pääsy fyysisen palvelimen laitteelle, jolloin samaan laitteeseen ei pääse käsiksi edes Domain 0 -virtuaalikone. Kyseisen laitteen laitteistoajurit on tällöin sisällytettävä Domain U -virtuaalikoneen ytimeen.

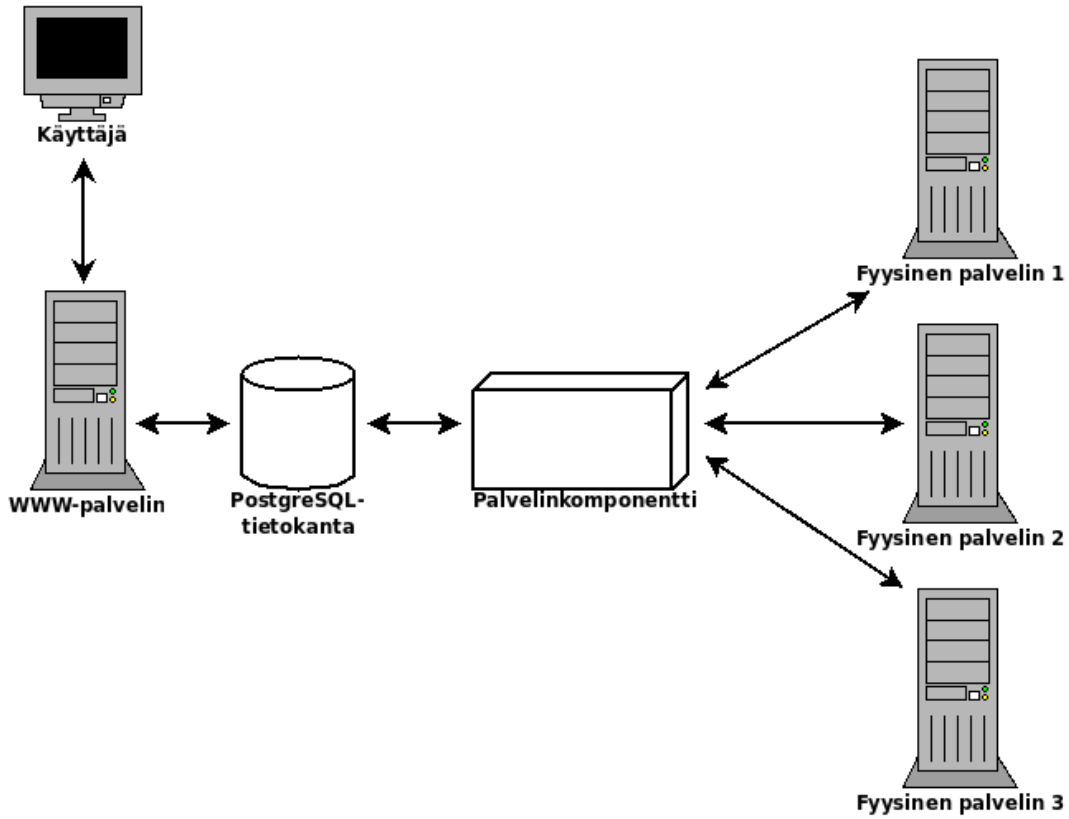
3 ESITUTKIMUS

Koska toteutettavaa sovellusta oltiin suunniteltu alustavasti jo ennen varsinaisen projektin aloittamista, keskityttiin työn alussa jo olemassa olevien vaatimusten tarkentamiseen. Lisäksi punnittiin, mitkä ominaisuudet voisi toteuttaa komponentin ensimmäisessä versiossa ja mitkä jätettäisiin myöhempiin versioihin. Eri toteutusvaihtoehtoja tutkittiin ja niiden toimivuutta testattiin käytännössä.

Tässä kappaleessa käsitellään komponentin käyttökohde, esitutkimuksen eri vaiheet sekä toteutusvaihtoehdot ja pohditaan valittujen toteutustekniikoiden hyötyjä ja haittoja.

3.1 Käyttökohde

Toteutettava palvelinkomponentti tulee olemaan osa Axxion VServer Manager -ohjelmistoa, jonka tarkoituksena on helpottaa Xen-virtuaalikonemoottoriin perustuvan virtuaalikoneympäristön valvontaa ja automatisoida sen hallintaa. Ohjelmiston avulla ylläpitäjä näkee yhdellä silmäyksellä koko virtuaalikoneympäristön tilan ja pystyy helposti suorittamaan tarvittavat ohjaustoimenpiteet. Järjestelmän ylläpitäjän lisäksi ohjelmistoa tulevat myöhemmin käyttämään myös yrityksen asiakkaat sekä mahdollisesti myyntihenkilöt. Kuvassa 2 on esitetty ohjelmiston rakenne, joka koostuu WWW-palvelimella olevasta käyttöliittymästä, tietokannasta sekä tässä työssä toteutettavasta palvelinkomponentista. Selkeyden vuoksi kuvaan on lisätty myös ohjelmiston käyttäjä sekä ohjattavaan virtuaalipalvelinverkkoon kuuluvat fyysiset palvelimet.



Kuva 2 Axxion VServer Manager -ohjelmiston rakenne

Koska ohjelmiston WWW-käyttöliittymät toteutetaan Axxion Oy:n aiemmin määrittelemän ja osin jo toteutetun Soil Webframe -alustalle, tullaan toteutettavassa palvelinkomponentissa käyttämään alustan käyttöön muokattuja kirjastoja komponentin tietokantakirjastojen pohjana. Soil Webframe on yrityksen sisäiseen käyttöön tarkoitettu portaalisovellusten toteuttamista yksinkertaistava pohjatoteutus ja ohjelmointirajapinta, joka tarjoaa siinä useimmin tarvittavat ohjelmistotyökalut ja -kirjastot.

Ohjelmistoon myöhemmin toteutettavan, asiakkaiden käyttöön tarkoitetun WWW-käyttöliittymän avulla yrityksen asiakkaille tarjotaan mahdollisuus seurata omien virtuaalikoneiden tilaa, lokeja ja resurssien käyttöä sekä tarvittaessa ohjata niiden suoritusta. Käyttöliittymän kautta asiakas voi käynnistää, uudelleen käynnistää tai sammuttaa virtuaalikoneen, samoin hän voi myös tauottaa sen suorituksen ja jatkaa sitä myöhemmin. Asiakkaan käyttöliittymään tullaan myöhemmin myös toteuttamaan graafinen statistiikkanäkymä, josta tämä voi seurata virtuaalikoneidensa resurssienkäyttöhistoriaa.

Myyjien käyttöön tarkoitetun käyttöliittymän pääasiallisena tarkoituksena on mahdollistaa uusien virtuaalipalvelimien luonti välittömästi tilauksen jälkeen, jolloin asiakas voi saada tilaamansa virtuaalipalvelimen nopeasti käyttöönsä. Tällöin uusien virtuaalipalvelimien luonti saadaan siirrettyä järjestelmän ylläpitäjiltä suoraan myyjien vastuulle. Virtuaalipalvelimen luonti kattaa uuden virtuaalikoneen luomisen sekä siihen tilatun käyttöjärjestelmän asennuksen ja peruskonfiguroinnin.

Ylläpitäjien käyttöliittymän tärkeimpinä käyttötarkoituksina ovat virtuaalikoneverkon valvonta ja ohjaus, kokonaistilanteen hahmottaminen sekä resurssien tarkkailu. Jos virtuaaliverkossa ilmenee jokin vikatilanne, pystyy ylläpitäjä käyttöliittymän kautta hahmottamaan nopeasti tilanteen ja korjaamaan vian. Resurssi-indikaattorien avulla virtuaalikoneverkon fyysisten koneiden kokonaiskuormaa sekä yksittäisten virtuaalikoneiden resurssien käyttöä voidaan seurata tehokkaasti. Käyttöliittymän kautta ylläpitäjä voi luoda ja poistaa virtuaalikoneita, siirtää niitä fyysiseltä palvelimelta toiselle sekä hallita niiden ajoa ja muokata niille annettuja resursseja, kuten lisätä muistia tai levytilaa. Ylläpitäjien käyttöliittymään tullaan toteuttamaan myöhemmin samanlainen graafinen resurssihistorian seuranta kuin asiakkaan käyttöliittymään.

3.2 Käyttöympäristö

Nykyinen virtuaalikoneympäristö, johon palvelinkomponentti toteutetaan, koostuu kolmesta Intel Xeon -palvelimesta, joista jokaisessa käytetään Xen-virtuaalikonemoottorin versiota 3.2.1. Etuoikeutettujen virtuaalikoneiden käyttöjärjestelmänä – eli hostina – on Debian GNU/Linux 4.0 (Etch) Linux-jakeluversio, joka on myös ohjelmiston alkuvaiheessa ainut tuettu Domain U:issa ajettava guest-käyttöjärjestelmä.

Fyysiset palvelimet on yhdistetty toisiinsa nopealla Ethernet-verkkotekniikalla, joka pystyy 1 Gb/s nopeuksiin, ja samaan verkkoon on liitetty myös tehokas ja suurikapasiteettinen iSCSI-levypalvelin, josta jaetaan Domain U:iden käyttämät lohkolaitteet. Domain 0 -virtuaalikoneet käyttävät lohkolaitteina fyysisten palvelimien omia kiintolevyjä.

Levypalvelimella sijaitsevien dynaamisten lohkolaitteiden luomisessa ja hallinnassa käytetään CLVM:ää, joka mahdollistaa samojen lohkolaitteiden käyttämisen ja hallitsemisen useammalta fyysiseltä palvelimelta samanaikaisesti.

Jaetun levypalvelimen ansiosta Domain U -virtuaalikone voidaan siirtää nopeasti fyysiseltä palvelimelta toiselle, sillä tällöin kohdepalvelimelle ei tarvitse siirtää kuin pelkästään siirrettävän virtuaalikoneen muisti, ja siirto voidaan tehdä sammuttamatta virtuaalikonetta. Nykyisessä ympäristössä suoritettujen kokeilujen perusteella siirtotapahtuma on asiakkaalle lähes huomaamaton. Jos siirrettävä virtuaalikone käyttäisi massamuistina fyysisen palvelimen kiintolevyjä, täytyisi siirrettävä virtuaalikone ajaa alas, siirtää kaikki data fyysiseltä palvelimelta toiselle ja käynnistää siirretty virtuaalikone uudella fyysisellä palvelimella. Jos fyysinen palvelin, jossa Domain U -virtuaalikone sijaitsee, hajoaa, niin toteutettava palvelinkomponentti lähettää ylläpitäjälle hälytyksen, ja jaetun levypalvelimen ansiosta virtuaalikone saadaan nopeasti takaisin käyttöön käynnistämällä se johonkin toiseen fyysiseen palvelimeen.

3.3 Kieli ja kehitysympäristö

Perl on Axxion Oy:n pääasiallinen ohjelmointikieli, ja sitä tullaan käyttämään hallintaohjelmiston muissa komponenteissa, kuten Soil Webframessa, joten se on valittu myös palvelinkomponentin toteutuskieleksi. Perlistä käytetään versiota 5.8.8, sillä se on tuettu sovellusympäristössä käytetyssä Debian GNU/Linux 4.0 (Etch) -käyttöjärjestelmässä.

Ohjelmoinnissa ei käytetä mitään tiettyä kehitystyökalua. Koodin tuottaminen tapahtuu millä tahansa tekstieditorilla, ja kaikki projektiin liittyvät materiaalit sekä lähdekoodit tallennetaan Axxion Oy:n Subversion-versionhallintajärjestelmään, joka pitää kirjaa materiaalin muutoshostoriasta. Projektin etenemistä seurataan tuntikirjanpidolla sekä viikoittaisilla palavereilla.

Palvelinkomponentin kehitystä ja testausta varten rakennetaan oma testipalvelin, jotta kehityksen aikana voitaisiin kokeilla Xen-virtuaalikonemoottorin eri versioita ja simuloida virhetilanteita ilman, että tuotantokäytössä olevat palvelimet

häiriintyisivät. Nykyiseen virtuaalikoneympäristöön hankitut kaksi fyysistä palvelinta ovat projektin alussa vielä testikäytössä, joten niitä voidaan käyttää eri tekniikoiden testaukseen.

3.4 Xen-virtuaalikonemoottorin ohjaustekniikat

Palvelinkomponentin ratkaisutapoja tutkiessa täytyi tutustua eri Xen-virtuaalikonemoottorin hallintatapoihin. Vaihtoehtoina oli Xenin omien shell-työkalujen käyttö erikseen toteutettavan daemon-komponentin kautta, valmiin kirjaston käyttäminen tai Xenin tarjoaman Xen APIn käyttö. Koska yhtenä aiemmin määritettynä vaatimuksena oli virtuaalikoneympäristöön tehtävien muutosten pitäminen mahdollisimman vähäisinä, jätettiin erillinen daemon-komponentti vaihtoehtojen ulkopuolelle.

Libvirt

Valmiista kirjastoista käyttökelpoisimmalta vaikutti libvirt, joka on tehty useampien eri virtuaalitekniikoihin pohjautuvien virtuaalikoneympäristöjen hallintaan. Libvirt pyrkii pitämään käyttäjille tarkoitetun rajapinnan mahdollisimman muuttumattomana, vaikka tuettujen virtualisointitekniikoiden rajapinta muuttuisi. Tästä olisi etuna se, että virtuaalikonemoottorin kehittyessä ja sen rajapintojen muuttuessa palvelinkomponenttiin ei tarvitsisi tehdä muutoksia, ja samalla toteutettava palvelinkomponentti tukisi useampaa virtualisointitekniikkaa.

Libvirtin ydin on toteutettu C-kielellä ja siihen on tarjolla komentorivityökalut sekä sidokset useammalle ohjelmointikielelle, kuten Pythonille, Perlille, OCamlille sekä Rubylle. Kirjastoa tutkiessa kuitenkin ilmeni, että sen tarjoamat Perl-sidokset olivat vanhoja eikä niiden kehitys vaikuttanut aktiiviselta. Tarvittavat sidokset olisi kuitenkin mahdollista luoda myös itse. /3/

Koska libvirtistä ei löytynyt suoraan ajantasaista asennuspakettia Debian Etchille, täytyi se kääntää käsin. Siinä vaiheessa kuitenkin ilmeni versioristiriitoja nykyisessä virtuaalikoneympäristössä käytettävän Xenin kanssa, minkä vuoksi libvirt päätettiin sulkea toistaiseksi kokonaan pois vaihtoehdoista.

Xen Management API

Xenin kehittäjät loivat oman Xen Management API -rajapinnan (myöh. Xen API) virtuaalikonemoottorin hallintaan, ja sen ensimmäinen versio 1.0.0 julkaistiin vuoden 2007 alkukesästä yhdessä Xen 3.0.5:n kanssa. Nykyisessä virtuaalikoneympäristössä käytettävä Xen 3.2.1 käyttää Xen APIn versiota 1.0.1.

Esitutkimuksen aikana tehtyjen kokeilujen perusteella Xen API valittiin toteutuksessa käytettäväksi virtuaalikonemoottorien ohjaustekniikaksi, koska se on monipuolinen ja helppokäyttöinen ja sen kehitys on aktiivista. Vaikka Xen APIn kehittäjät eivät takaa rajapinnan rakenteen ja Xen APIn luokkahierarkian vakautta, ei se vaikuta suuresti toteutukseen. Ohjausohjelmistoa voidaan päivittää myöhemmin samaan aikaan kun uusi Xen-versio on yrityksessä testausvaiheessa. Xen APIn käyttö ei kuitenkaan ollut ongelmatonta, sillä esitutkimuksen aikana siitä löytyi paljon toimintoja, jotka oli dokumentoitu, mutta joita ei ollut toteutettu.

Xen APIa ohjataan XML-RPC-kutsuilla. XML-RPC on tietoliikenneprotokolla, joka käyttää XML-kieltä etäproseduurikutsujen välittämiseen HTTP- tai HTTPS-protokollan yli. XML-RPC:n määrittely on hyvin pieni ja se kattaa vain perus-tietotyypit datan esittämiseen. Xen APIssa niistä käytettäviä ovat seuraavat: double, boolean, dateTime.iso8601, string, array sekä struct. Koska Xen APIssa käytettävät kokonaisluvut ovat 64-bittisiä eikä XML-RPC:ssä ole tietotyyppiä kuin 32-bittisille kokonaisluvuille, käytetään kokonaislukujen esityksessä XML-RPC:n string-tietotyyppiä.

Alla on esimerkki XML-RPC-pyynnöstä, joka muodostuu kun käytetään Xen APIn kirjautumiskomentoa `session.login_with_password('username', 'password')`:

```
<?xml version='1.0'?>
<methodCall>
<methodName>session.login_with_password</methodName>
<params>
  <param>
    <value><string>username</string></value>
  </param>
```

```
  <param>
    <value><string>password</string></value>
  </param>
</params>
</methodCall>
```

Vastauksena Xen API antaa struct-muotoisen paluuviestin, josta ilmenee kyselyn tila sekä kyselyn palauttamat arvot tai virheilmoitus. Alla on esimerkki onnistuneen kirjautumisen jälkeisestä XML-RPC-vastauksesta:

```
<?xml version='1.0'?>
<methodResponse>
<params>
<param>
<value>
<struct>
  <member>
    <name>Status</name>
    <value><string>Success</string></value>
  </member>
  <member>
    <name>Value</name>
    <value><string>
      59e7c2da-cd85-30dd-7d18-cbb7a8a48317
    </string></value>
  </member>
</struct>
</value>
 </param>
 </params>
 </methodResponse>
```

Kyselyn onnistuessa paluuviestissä tulleen tietorakenteen `Status`-tietueessa on arvo `Success` ja kyselyn data on tietueessa `Value`, joka voi kyselystä riippuen sisältää yksittäisen arvon, taulukon, struktuurin tai niiden yhdistelmän. Yllä olevassa esimerkkitapauksessa `session.login_with_password`-funktio palautti onnistuneen kirjautumisen yhteydessä yksilöllisen yhteystunnisteen.

Virhetilanteessa `Status`-tietueessa on arvo `Failure` ja `Value`-tietueen sijaan on `ErrorDescription`-tietue, joka sisältää taulukon, josta ilmenee virhekoodi ja virheen parametrit. Alla olevasta esimerkistä on poistettu kaikille paluuviesteille tyypilliset XML-tagit ja esitetty vain paluuviestissä tullut tietorakenne. Esimerkki kuvaa virhetilannetta, jossa jo käynnissä olevalle virtuaalikoneelle on yritetty antaa käynnistyskäsky:

```
<struct>
  <member>
    <name>Status</name>
    <value><string>Failure</string></value>
  </member>
  <member>
    <name>ErrorDescription</name>
    <value>
      <array>
        <data>
          <value><string>VM_BAD_POWER_STATE</string></value>
          <value><string>
            789ece10-0bf5-7c36-1401-f584ea037422
          </string></value>
          <value><string>Halted</string></value>
          <value><string>Running</string></value>
        </data>
      </array>
    </value>
  </member>
</struct>
```

Yllä olevassa esimerkissä paluuviestissä tulleen tietorakenteen `ErrorDescription`-tietueessa on taulukko, joka sisältää virhekoodin `VM_BAD_POWERSTATE`, komennon suorittamiseen vaaditun virtuaalikoneen tilan `Halted` sekä virtuaalikoneen nykyisen tilan `Running`.

Xen API käyttää oliokielen termejä 'luokka' ja 'olio' kuvaamaan Xen-järjestelmän eri komponentteja. Xen APIn terminologiassa luokat kuvaavat hierarkista nimiavaruutta ja oliot ovat luokkien ilmentymiä, joilla parametrit on asetettu tiettyihin arvoihin. Esimerkiksi yksi käynnissä oleva virtuaalikone on VM-olio, jolla on 'power_state'-parametrina arvo 'Running'.

Koska dokumentoitu Xen API sisältää paljon luokkia, joista kaikkia ei ole vielä implementoitu, ja luokkia, joita ei käytetä toteutettavassa palvelinkomponentissa, on taulukossa 1 listattu vain palvelinkomponentin toteutusta koskevat luokat:

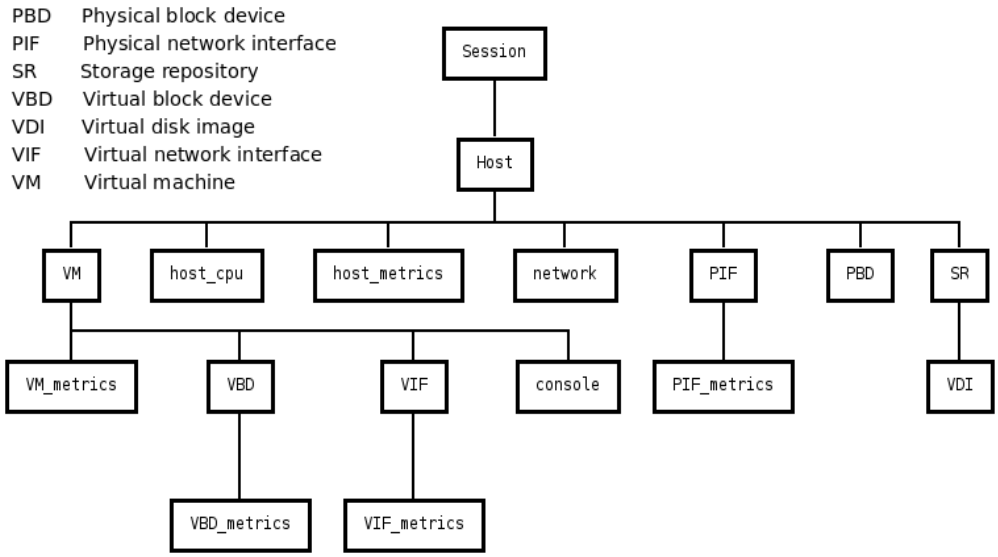
Taulukko 1 Palvelinkomponenttiin liittyvät Xen APIn luokat

Luokka	Selite
session	Xen APIn yhteysluokka
VM & VM_metrics	*Virtual machine,* Virtuaalikone sekä sen resurssit
host & host_metrics	Fyysinen palvelin sekä sen resurssit
host_cpu	Fyysisen palvelimen prosessori
network	Virtuaalinen verkko
VIF & VIF_metrics	*Virtual network interface,* Virtuaalisen verkon rajapinta sekä sen resurssit
PIF & PIF_metrics	*Physical network interface,* Fyysisen verkon rajapinta sekä sen resurssit
SR	*Storage repository,* Tietovarasto
VDI	*Virtual disk image,* Virtuaalinen levy
VBD & VBD_metrics	*Virtual block device,* Virtuaalinen lohkolaite sekä sen resurssit (lukujen ja kirjoitusten määrä)
PBD	*Physical block device,* Fyysinen lohkolaite
console	Virtuaalikoneen konsoli

Jokainen Xen APIn luokista sisältää sitä koskevat parametrit, metodit parametrien lukemiseen ja kirjoittamiseen sekä metodit luokan olion ohjaamiseen. Liitteenä 1 on nähtävissä esimerkki Domain U:n VM- ja VM_metrics-olioiden parametreista, jotka on saatu käyttämällä niiden get_record-metodia.

Jokaisella oliolla on oma yksilöllinen heksadesimaalimuotoinen UUID-arvo, jonka perusteella luokan komennot ohjataan oikealle luokan oliolle. UUID (Universally Unique Identifier) on standardi yksilöllisten 128-bittisten tunnisteiden luomiseen, jossa kahden samanlaisen tunnisteen esiintyminen on erittäin epätodennäköistä. /7/

UUID-tunnisteita käytetään myös olioiden viittauksissa toisiinsa. Koska Xen APIn toteutuksesta puuttuu monia dokumentoituja luokkia, hajoavat useat Xen APIn dokumentaatiossa määritetyt olioviittaukset, ja ne joudutaan korjaamaan luomalla puuttuvat viittaukset palvelinkomponentissa. Kuvassa 3 on kuvattu palvelinkomponentin toteutuksessa käytettävä Xen APIn luokkahierarkia. /5/, /6/



Kuva 3 Palvelinkomponentissa käytettävä Xen APIn luokkahierarkia

3.5 Tietokanta

Tietokantana käytetään Axxion Oy:n muissakin ohjelmistoprojekteissa yleisesti käyttämää PostgreSQL 7.4:ää, koska se on tuettu sovellusympäristössä käytetyssä Debian GNU/Linux 4.0 (Etch) -käyttöjärjestelmässä. PostgreSQL on monipuolinen olio-relaatiotietokannan hallintajärjestelmä, joka on julkaistu BSD-lisenssin alaisuudessa.

Tietokantaa ei käsitellä toteutettavasta palvelinkomponentista suoraan, vaan komponentin ja tietokannan välissä käytetään Perlin Rose::DB::Object ORM-kirjastoa (*Object Relational Mapper*). Kirjaston avulla tietokannan taulut saadaan kuvattua luokkina, tietokannan tiedot käsiteltyä luokista luotavien olioiden kautta ja tiedot tallennettua kantaan ja haettua kannasta olioiden metodien avulla. Tämä nopeuttaa komponentin toteuttamista huomattavasti, sillä kaikki tiedon käsittely voidaan suorittaa olio-ohjelmoinnin keinoin, eikä olioiden tallentamismetodeja ja tarvittavia SQL-lausekkeita tarvitse rakentaa itse.

Koska sovelluksen WWW-käyttöliittymien alustana käytettävä Soil Webframe sisältää valmiiksi sovelluksen käyttöön muokatut Rose::DB::Object-kirjastot, käytetään niitä pohjana palvelinkomponentin tietokantakirjastoille.

3.6 Vaatimukset

Tässä kappaleessa on listattu palvelinkomponentille jo aiemmin määritetyt sekä esitutkimuksen aikana esiintulleet vaatimukset.

Rajapinnan toiminnot

Kappaleessa 3.1 kuvatut käyttöliittymät määrittivät toteutettavalle palvelinkomponentille seuraavat toiminnalliset vaatimukset, joita on joiltakin osin tarkennettu:

Vaatimus 1: Virtuaalikoneen käynnistys, uudelleenkäynnistys ja sammutus
Toiminnallisuutta testattiin jo määrittelyvaiheessa. Testauksen perusteella vaatimus on helposti toteutettavissa.

Vaatimus 2: Virtuaalikoneen ajon tauottaminen ja ajon jatkaminen
Koska Xen API tarjoaa kaksitasoisen tauottamisen, toteutetaan ne molemmat. 'Pause'-tauotus keskeyttää vain suorituksen ajon, ja syvempi 'suspend'-tauotus tallentaa virtuaalikoneen muistin levylle, sammuttaa virtuaalikoneen ja vapauttaa sen käyttämät resurssit.

Vaatimus 3: Virtuaalikoneen siirtäminen fyysiseltä palvelimelta toiselle
Siirtämistä testattiin onnistuneesti jo projektin alkuvaiheessa. Siirrossa tuetaan sekä online- että offline-siirtoa.

Vaatimus 4: Virtuaalikoneen luonti, käyttöjärjestelmän asennus ja konfigurointi
Toteutettavan laajemman sovelluksen yksi tärkeimmistä toiminnallisista vaatimuksista on virtuaalipalvelimien luonnin helpottaminen ja siirtäminen mahdollisesti ylläpitäjiltä myyjille. Ensimmäisessä ohjelmistoversiossa tuetaan vain Debian GNU/Linux 4.0 (Etch) -käyttöjärjestelmää.

Vaatimus 5: Virtuaalikoneen poisto, jossa optio konfiguraation ja datan säilyttämiseen

Järjestelmästä poistettavan virtuaalikoneen konfiguraatio ja data voidaan haluta säilyttää esimerkiksi toiseen järjestelmään siirtämistä varten. Tämän vuoksi virtuaalikoneen poistamistoiminnon tulisi tukea tällaisia optioita.

Vaatimus 6: Virtuaalikoneen resurssien muokkaus, esim. muistin ja levytilan lisäys tai vähennys

Xen API tukee muistin määrän lisäämistä ja vähentämistä virtuaalivirtuaalikoneen ollessa käynnissä, joten asiakkaan virtuaalikonetta ei tarvitse uudelleenkäynnistää muistin määrän säätämisen jälkeen. CLVM:n käytön ansiosta myös käytettävissä olevan levytilan lisääminen tai vähentäminen on mahdollista. Koska virtuaalikone lukee lohkolaitteiden koot ainoastaan käynnistyksen yhteydessä, tulevat niihin tehdyt muutokset voimaan vasta virtuaalikoneen uudelleenkäynnistyksen jälkeen.

Vaatimus 7: Virtuaalikoneiden käytettävissä olevien prosessoriytimien määritys

Koska toimintoa ei ehditty testaamaan tuotantokäyttöön tulevilla moniytimisillä palvelimilla, ja nykyisessä testipalvelimessa ei ole kuin yksi ydin, jätetään toiminnallisuuden toteutus seuraavaan versioon.

Toiminnalliset vaatimukset

Toiminnalliset vaatimukset koskevat komponentin sisäistä toimintaa. Toiminnallisiksi vaatimuksiksi määritettiin seuraavat:

Vaatimus 8: Virtuaalikoneverkkoon tehtävät muutokset pysyttävä minimissään

Koska toteutettavan hallintaohjelmiston tarkoitus on vain tukea virtuaalikoneympäristön ylläpitotöitä, on ohjelmistoon liitettävä komponentti toteutettava niin, että virtuaalikoneympäristön toiminta ei häiriinny ja ylläpito esty, vaikka hallintaohjelmisto olisi poissa käytöstä. Tämän vuoksi ohjelmiston vaatimat muutokset fyysisille palvelimille ja virtuaalikoneille tulisi pitää mahdollisimman vähäisinä.

Vaatimus 9: Palvelinkomponentti lukee määritellyin väliajoin virtuaalikoneiden tilat sekä konfiguraatiot ja kirjoittaa ne tietokantaan

Tämä on komponentin yksi pääasiallisista toiminnoista. Virtuaalikoneiden tilat voidaan lukea helposti Xen APIn avulla, mutta se ei tue suoraan konfiguraatioiden lukemista siinä muodossa, missä Xenin omat työkalut niitä käyttävät.

Vaatimus 10: Palvelinkomponentti kirjoittaa Domain U -virtuaalikoneen konfiguraation tiedostoksi kyseisen fyysisen palvelimen host-käyttöjärjestelmään

Koska edellä kävi ilmi, ettei Xen API anna suoraan tarvittavia tietoja konfiguraatiotiedostojen luomiseen, vaatii tämä vaatimus tarkempaa tutkimista, ja se toteutetaan vasta komponentin seuraavassa versiossa.

Vaatimus 11: Palvelinkomponentin on havaittava järjestelmään käsin tehdyt muutokset

Koska virtuaalikoneita voidaan ohjata suoraan host-käyttöjärjestelmistä, tulee komponentin havaita muutokset ja tallentaa muutoksen tiedot tietokantaan.

Vaatimus 12: Palvelinkomponentti lukee määritellyin väliajoin tietokannassa olevaa komentojonotaulukkoa, välittää komennot virtuaalikoneille ja kirjoittaa suorituksen tilan ja mahdollisen virheilmoituksen tietokantaan

Vaatimus 13: Komentojonon käsittelyssä on kaksivaiheinen kuittaus (ymmärretty/käsittelyssä – valmis/virhe)

Kaksi edellistä vaatimusta saadaan täytettyä toteuttamalla erillinen komentojonoloki, jonne tiedot siirretään komentojonosta, kun komento otetaan suoritettaviksi.

Vaatimus 14: Jokaisella Domain 0 -virtuaalikoneella ajetaan erillistä daemon-prosessia, joka vastaanottaa palvelinkomponentilta tulevat komennot, ohjaa niiden suoritusta ja lähettää kuittauksen palvelinkomponentille

Tämä vaatimus hylättiin, koska virtuaalikonejärjestelmään tehtävät muutokset täytyy pitää mahdollisimman pieninä, ja saman toiminnallisuuden saa toteutettua myös ilman erillistä daemon-prosessia.

Vaatimus 15: Liikenne palvelinkomponentin ja virtuaalikoneiden välillä on salattua, ja jokaisella koneella on mahdollisesti oma salausavain

Palvelinkomponentin ja virtuaalikoneiden välissä voidaan käyttää SSH-tunnelia, jolloin liikenne niiden välillä on salattu.

Vaatimus 16: Palvelinkomponentti valvoo virtuaalikoneiden tilaa ja hälyttää virhetilanteiden ilmetessä

Jotta väärien hälytysten määrä saataisiin minimoitua, on virhetilanteiden määrityksessä otettava huomioon esimerkiksi tilanne, jossa käynnissä oleva virtuaalikone sammutetaan käsin järjestelmän ulkopuolelta. Tämä on määritelty vaatimuksessa 11.

Vaatimus 17: Palvelinkomponentti valvoo virtuaalikoneiden resurssien käyttöä ja hälyttää resurssien käydessä vähiin

Resurssien hetkellinen käyttöaste luetaan Xen APIn tarjoamilla metodeilla.

Vaatimus 18: Palvelinkomponentti kerää resurssistatistiikkaa tietokantaan

Statistiikkahistoria voidaan tallentaa tietokantaan round-robin-muodossa.

Vaatimus 19: Palvelinkomponentin täytyy tukea eri käyttäjäoikeustasoja

Eri käyttäjäoikeustasot tarvitaan varmistamaan, etteivät asiakkaat ja myyjät pääse vahingossakaan suorittamaan komentoja, joihin heillä ei ole oikeuksia.

Palvelinkomponentin toteutuksen vaatimukset

Toiminnallisten vaatimuksien lisäksi projektiin liittyy myös toteutuksellisia vaatimuksia.

Vaatimus 20: Selkeä, vakaa ja helppokäyttöinen rajapinta

Soil Webframe -portaalitoteutusalustan avulla tehtävät käyttöliittymät käyttävät yhteisiä tietokantarajapintoja toteutettavan palvelinkomponentin kanssa. Jotta käyttöliittymien kehitys olisi helppoa, tulisi palvelinkomponentin rajapintojen rakenteen olla selkeä ja pysyä mahdollisimman muuttumattomana.

Vaatimus 21: Koodin rakenne selkeä ja hyvin kommentoitu
Toteutettavan palvelinkomponentin koodin ylläpidettävyyden ja jatkokehittämisen vuoksi koodin rakenteen tulisi olla selkeää ja hyvin kommentoitua.

4 KOMPONENTIN SUUNNITTELU

Suunnitteluvaihe kulki osittain päällekkäin määrittely- ja toteutusvaiheen kanssa, koska komponenttien toiminta ja tietokannan rakenne alkoi hahmottumaan vasta valittujen toteutustekniikoiden testauksen aikana.

Kaikista aikaa vievin osa-alue oli tutustuminen Xen APIin, koska myös Perl-kieli ja XML-RPC vaativat tarkempaa perehtymistä, eikä Xen APIn käyttöön Perlillä löytynyt helposti esimerkkejä. Alkuvaikeuksien jälkeen päästiin tutustumaan tarkemmin Xen APIn luokkahierarkiaan, ja sen pohjalta kehitettiin alustava suunnitelma tietokannan sekä itse palvelinkomponentin rakenteista.

Tämä kappale kertoo tarkemmin komponentin suunnittelusta, joka käsittää palvelinkomponentin koodin rakenteen, sen ulospäin tarjoaman rajapinnan, tietokannan rakenteen sekä jaettujen tietokantakirjastojen suunnittelun.

4.1 Xen Management API

Xen APIn perusasiat ja XML-RPC:n toiminta kerrotaan kappaleessa 3.4, joten niitä ei käydä tässä enää tarkemmin läpi.

Xen-virtuaalikonemoottorin lähdekoodin mukana tulee kattava Xen APIn dokumentaatio, josta ilmenee Xen APIn luokat, niiden mahdolliset parametrit sekä metodit. Esitutkimusvaiheessa törmättiin kuitenkin ominaisuuksiin, jotka oli kirjoitettu jo dokumentaatioon, mutta joiden implementaatio puuttuu. Esimerkiksi kaikki Xen APIn käyttäjät listaava komento `user.get_all` palautti Xen APIlta virheen `MESSAGE_METHOD_UNKNOWN`, joka tarkoittaa sitä, että yritettiin kutsua metodia, jota ei ole olemassa. Xen APIsta puuttuvien implementaatioiden vuoksi osa komponentille asetetuista vaatimuksista jätetään toteutettaviksi seuraavaan versioon, koska puuttuvien toiminnallisuuksien toteuttaminen muilla keinoin

vaatisi liian paljon aikaa. Seuraavaan versioon siirretyt vaatimukset on listattu kappaleessa 6.2.

Xen APIn käyttöä varten komponenttiin toteutetaan oma Session-luokka, joka sisältää XML-RPC-käskyjen käsittelyn sekä Xen APIn yhteydenhallinnan. Toteutettavan luokan täytyy täyttää seuraavat vaatimukset:

- Jokaiselle hallittavalle fyysiselle palvelimelle luodaan oma Session-olio
- Session-luokan rakentajalle annetaan parametreina host-käyttöjärjestelmässä ajettavan Xen API -palvelun osoite, portti ja kirjautumistiedot
- Olio luodaan vain, jos kirjautuminen onnistuu
- Olioon tallennetaan kirjautumisen yhteydessä saatu yhteystunniste, joka liitetään automaattisesti jokaiseen lähetettävään Xen API -kyselyyn
- Luokka sisältää metodin, joka tekee parametrina annetun XML-RPC-kyselyn Xen APIlle ja palauttaa vastauksen tyypistä riippuen joko arvon (scalar), taulukon (array) tai avain-arvopareja sisältävän tietorakenteen (hash) vastauksena
- Luokan purkajassa suoritetaan uloskirjautuminen Xen APIsta, jonka jälkeen yhteystunniste ei ole enää voimassa

4.2 Tietokanta

Komponentissa käytettävä tietokanta on jaettu kahteen osioon, yleiseen ja Xen APIin. Xen API -osiota käytetään ainoastaan Xen APIn käyttämien tietojen säilyttämiseen ja sen rakenteen perustana käytetään suoraan Xen APIn luokkahierarkiaa ja nimiterminologiaa poislukien Session-luokka. Xen APIn luokkahierarkia ja nimiterminologia on kuvattu kappaleen 3.4 loppuosassa.

Xen API -osiossa jokainen taulu vastaa Xen APIn luokkaa ja taulun kenttä luokan parametria. Yhteisen rakenteen ansiosta Xen APIn antamat status-tiedot voidaan

kirjoittaa suoraan tietokantaan mahdollisimman pienillä muutoksilla. Ainoat tarvittavat muutokset ovat taulujen ja kenttien nimien muuttaminen pienimerkkisiksi sekä joidenkin Xen APIn palauttamien parametrien sisältämien tietorakenteiden muuntaminen JSON-muotoon. JSON-muunnoksella taulukko tai avain-arvopareja sisältävä tietorakenne saadaan tallennettua merkkijonona tietokantaan muuntamalla taulukot muotoon '["arvo1","arvo2",..]' ja avain-arvopareja sisältävät tietorakenteet muotoon '{"avain1":"arvo1","avain2":"arvo2",..}'. Tietorakenteet voivat esiintyä myös sisäkkäin.

Koska Xen API -osion taulut ja niiden kentät ovat suora kopio Xen APIn luokkahierarkiasta ja parametreista, niitä ei käsitellä tässä sen tarkemmin. Niiden kuvaus löytyy Xen APIn dokumentaatiosta.

Tietokannan yleisessä osiossa sijaitsevat kaikki muut palvelinkomponentin käyttämät taulut, jotka on listattu Taulukossa 2.

Taulukko 2 Tietokannan yleisen osan taulut

Taulu	Selite
command_log	Komentohistoria
command_queue	Komentojono
kernel	Tiedot virtuaalipalvelimissa käytettävistä kernel-imagetiedostoista
ramdisk	Tiedot virtuaalipalvelimissa käytettävistä initrd-imagetiedostoista
useraccount	Tiedot käyttäjistä
xend_server	Tiedot hallittavista fyysisistä palvelimista

'command_queue'-tauluun lisätään käyttöliittymän kautta virtuaalikoneille annettavat komennot, jotka palvelinkomponentti suorittaa. Taulukossa 3 on listattu taulun kentät, kenttien tietotyypit sekä selitteet.

Taulukko 3 'command_queue'-taulun kentät

Kenttä	Tietotyyppi	Selite
uid	integer (primary key)	Tietueen järjestysnumero
command	varchar(64)	Kohteena olevan luokan nimi sekä sille suoritettava komento pisteellä erotettuna
host_uuid	char(36)	Kohteena olevaa fyysistä palvelinta kuvaavan host-olion UUID-parametri
object_uuid	char(36)	Komennon kohteena olevan luokan olion UUID-parametri
parameters	text	Komentoon liittyvät parametrit JSON-muodossa avain-arvopareina
created	timestamp	Aika, jolloin komento lisättiin tauluun
useraccount_uid	integer	Viittaus useraccount-tauluun, josta käy ilmi komennon lisännyt käyttäjä

Kun palvelinkomponentti ottaa 'command_queue'-taulusta komennon käsittelyyn, siirretään se 'command_log'-tauluun ja sen arvoja päivitetään komennon edetessä. 'command_log'-taulu on muutoin vastaava 'command_queue'-taulun kanssa, mutta siihen on lisätty taulukossa 4 listatut kentät komennon tilan seuraamiseksi.

Taulukko 4 'command_log'-tauluun lisätyt kentät

Kenttä	Tietotyyppi	Selite
executed	timestamp	Aika, jolloin komento otettiin käsittelyyn
finished	timestamp	Aika, jolloin komennon suoritus on päättynyt
status	char(7)	Komennon suorituksen tila. Mahdolliset arvot: 'Running', 'Success', 'Failure'
value	text	Komennon palauttama informaatio tai virheilmoitus

'kernel'- ja 'ramdisk'-taulujen rakenne on identtinen. Rakenne on kuvattu taulussa 5.

Taulukko 5 'kernel'- ja 'ramdisk'-taulujen kentät

Kenttä	Tietotyyppi	Selite
uid	integer (primary key)	Tietueen järjestysnumero
location	text	Imagetiedosto sekä sen polku
name_label	varchar(64)	Tietueen yksilöivä nimi
name_description	varchar(255)	Tietueen selite

'useraccount'-taulu on tarkoitettu käyttäjätietojen tallentamiseen ja se sisältää vain kentät käyttäjätunnukselle, salasanalle ja tietueen järjestysnumerolle. Koska taululla ei ole merkitystä tässä työssä, sitä ei kuvata tarkemmin.

'xend_server'-taulussa on listattu tiedot kaikista hallittavista fyysisistä palvelimista. Taulun rakenne on esitetty taulukossa 6.

Taulukko 6 'xend_server'-taulun kentät

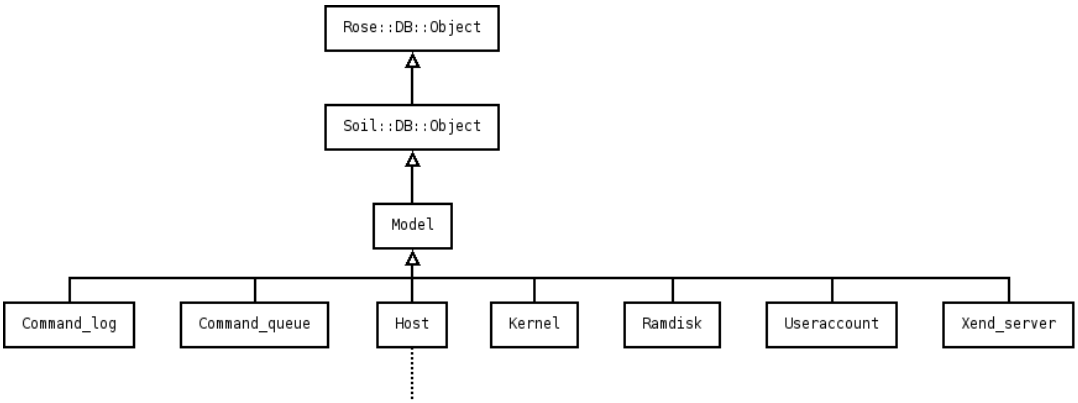
Kenttä	Tietotyyppi	Selite
uid	integer (primary key)	Tietueen järjestysnumero
address	varchar(254)	Ohjattavan fyysisen palvelimen host-käyttöjärjestelmässä olevan XML-RPC-palvelun osoite (Xen API)
port	integer	XML-RPC-palvelun portti (Xen API)
ssh_user	varchar(64)	Ohjattavan fyysisen palvelimen host-käyttöjärjestelmässä olevan SSH-palvelun käyttäjätunnus
ssh_password	varchar(64)	SSH-palvelun salasana
ssh_port	integer	Portti, jota käytetään XML-RPC-palveluun luotavassa SSH-tunnelissa
xenduser	varchar(64)	Xen API -palvelun käyttäjätunnus
xendpassword	varchar(64)	Xen API -palvelun salasana
enabled	boolean	
name_label	varchar(64)	Tietueen yksilöivä nimi
name_description	varchar(255)	Tietueen selite

4.3 Tietokantaluokat

Palvelinkomponentin tietokantaluokkien ylimpänä yliluokkana, eli juuriluokkana, käytetään Soil::DB::Object-luokkaa. Tämä luokka on periytetty Rose::DB::Object-luokasta ja siinä on määritelty tietokannan joissakin Xen API -osion taulujen kentissä tarvittava JSON-konversio, jonka avulla taulukko tai avain-arvopareja sisältävä tietorakenne saadaan talletettua text-tietotyypin kenttään.

Soil::DB::Object-luokasta periytetään Model-luokka, jossa määritetään tietokantayhteys, ja joka toimii kaikkien tietokannan tauluja kuvaavien luokkien yliluokkana. Tietokannan Xen API -osion tauluja kuvaavissa luokissa käytetään

Xen APIn luokkahierarkiaa, ja yleisen osan tauluja kuvaavat luokat on sijoitettu suoraan Model-luokan aliluokiksi. Kuvassa 4 on kuvattu osa tietokantaluokkien periytymishierarkiasta. Selkeyden vuoksi kuvasta on jätetty pois Host-luokan aliluokat, koska ne noudattavat kappaleessa 3.4 olevassa kuvassa 3 näkyvää hierarkiaa.



Kuva 4 Model-luokkien periytymishierarkia

4.4 Ulkoinen rajapinta

Palvelinkomponentin ulkoisena rajapintana toimii ainoastaan tietokanta. Palvelinkomponentille annetaan komentoja lisäämällä ne tietokannan 'command_queue'-tauluun ja komentojen suoritusta valvotaan lukemalla 'command_log'-taulua. Virtuaalikoneverkon tiedot saadaan luettua tietokannan Xen API -osiosta.

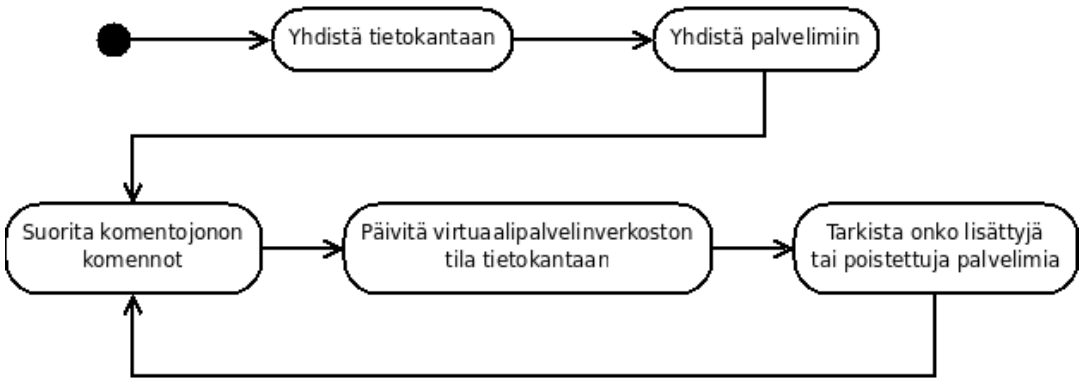
5 TOTEUTUS

Palvelinkomponentin toteutus aloitettiin osittain jo esitutkimus- ja suunnitteluvaiheessa, kun tutustuttiin valittujen tekniikoiden toimintaan. Toteutus keskittyi alkupäässä XML-RPC:n ja Xen APIn toimintaan, jotta tietokannan sekä tietokantaluokkien rakenteet saataisiin lopulliseen muotoonsa. Koska suunnittelu oli vaikeaa tuntemattoman ympäristön takia, jouduttiin toteutuksen aikana jatkuvasti tutkimaan ja kokeilemaan tarkemmin toteutuksessa käytettyjä menetelmiä. Tämän vuoksi toteutusvaiheesta muodostui pitkä prosessi.

Tässä kappaleessa käydään läpi komponentin toimintalogiikkaa sekä kerrotaan eri suoritusvaiheiden toteutuksesta.

5.1 Komponentin toimintalogiikka

Komponentin perustoimintalogiikka on hyvin yksinkertainen. Käynnistysvaiheessa komponentti ottaa yhteyden tietokantaan, lukee sieltä listan fyysisistä palvelimista, yhdistää jokaisessa olevaan XML-RPC-palveluun ja aloittaa toistamaan sekvenssiä: suorita komentojonon komennot, päivitä järjestelmän kaikkien Xen API -luokkien parametrit tietokantaan, tarkista onko palvelinlistassa muutoksia, odota ennalta määrätty aika ja toista alusta. Toimintalogiikka on esitetty kuvassa 5.



Kuva 5 Komponentin toiminta tilakaaviona

Tietokantayhteys

Komponentin ensimmäinen tehtävä on avata tietokantayhteys. Jos tietokantaan ei saada yhteyttä, komponentti antaa virheilmoituksen ja suoritus lopetetaan.

Tietokantayhteyden muodostamisessa tarvittavat asetukset on tallennettu komponentin käyttämään konfiguraatiotiedostoon, josta ne luetaan ja tallennetaan staattisesti Rose::DB-luokkaan. Aina kun tietokantakirjaston Model-luokkaa kutsutaan, tarkistetaan tietokantayhteyden tila, ja yhteys muodostetaan tarvittaessa. Tästä toimintamallista on se etu, että tietokantayhteys voidaan tarvittaessa katkaista missä kohdin koodia tahansa, eikä yhteyden uudelleenaukaisemisesta tarvitse huolehtia erikseen, koska se tapahtuu automaattisesti.

Palvelinlistan nouto

Tietokantakonfiguraation jälkeen komponentti lukee tietokannan 'xend_server'-taulusta listan, joka sisältää jokaisessa fyysisessä palvelimessa ajettavan host-käyttöjärjestelmän XML-RPC- sekä Xen API -palvelun tiedot. Koska Rose::DB::Object-luokan avulla voidaan hakea vain yksittäisiä tietueita tietokannasta, on kirjastoon toteutettu erillinen Manager-luokka useampien tietueiden käsittelyyn. Tietokannan 'xend_server'-taulua vastaava tietokantaluokka on Model::Xend_server. Alla on esimerkki kaikkien 'xend_server'-taulussa olevien tietueiden hakemisesta `@xen_servers`-taulukkoon Manager-luokan avulla:

```
@xen_servers =
     @{ Model::Xend_server::Manager->get_xend_servers() };
```

Esimerkissä käytetty `get_xend_servers`-metodi on luotu automaattisesti Manager-luokkaan tietokannan nimen perusteella. Koska metodi palauttaa viittauksen taulukkoon, joudutaan komento sijoittamaan `@{}`-lohkon sisään, joka palauttaa viittauksen osoittaman taulukon.

Palautetun taulukon alkioina on jokaista tietuetta kuvaava Rose::DB::Object-luokan olio, jonka avulla tietueen kenttiin päästään käsiksi. Alla olevassa esimerkissä taulukon ensimmäisessä alkiossa olevasta tietueesta tallennetaan 'address'-kentän arvo erilliseen `$address`-muuttujaan:

```
$address = $xen_servers[0]->address;
```

Yhdistäminen palvelimiin

Seuraavaksi manager-luokan avulla saatu taulukko käydään läpi ja yhdistetään tietueissa oleviin palvelimiin, jos tietueen 'enabled'-kentän arvo on tosi.

Koska yhteydessä käytetään SSH-tunnelia, ei yhteyttä oteta suoraan XML-RPC-palveluun. SSH-tunneli muodostetaan ottamalla SSH-yhteys 'address'-tietueessa olevaan osoitteeseen ja käyttämällä tietueissa 'ssh_user' ja 'ssh_password' olevia tunnistetietoja. Jos 'ssh_password'-kentässä ei ole arvoa, yritetään kirjautumisessa käyttää SSH-avainta. Koska SSH-yhteyden muodostamisessa käytetään Net::SSH::Expect-kirjastoa, joka on vain rajapinta järjestelmän omalle SSH-

komennolle, on SSH-avaimet tallennettava sille koneelle, jolla palvelinkomponentti ajetaan.

Tunnistetietojen yhteydessä SSH-yhteydelle määritellään myös tunneli, jonka paikallinen portti määritetään 'ssh_port'-kentässä. Toinen pää yhdistetään host-käyttöjärjestelmän XML-RPC-palvelun porttiin, joka on määritetty tietueen 'port'-kentässä.

SSH-tunnelin muodostamisen jälkeen luodaan fyysistä palvelinta kuvaava Host-objekti, jonka yhteydessä luodaan myös Session-objekti. Session-objekti muodostaa yhteyden XML-RPC-palveluun sille annettujen osoite-, portti- ja käyttäjätunnustietojen avulla. Koska yhteys otetaan SSH-tunnelin kautta, annetaan sille osoitteeksi tietokannassa olevan osoitteen sijaan paikalliseen koneeseen osoittava '127.0.0.1' ja portiksi 'ssh_port'-kentässä oleva arvo. Tällöin XML-RPC-yhteys tehdään salattua tunnelia pitkin. Jos yhteyden muodostus epäonnistuu, tulostetaan virheilmoitus, tuhotaan luodut objektit ja siirrytään seuraavaan taulukon alkioon.

Koska palvelinkomponentissa toistettavaan sekvenssiin kuuluu palvelinlistan läpikäynti sekä listaan tehtyjen muutosten tarkkailu, tarkistetaan tässä vaiheessa, ettei komponentista ole muodostettu yhteyttä palvelimiin, joiden 'enabled'-kentän arvo on epätosi, tai joiden määritystä ei löydy lainkaan tietokannasta.

Tietojen päivittäminen virtuaalikoneverkosta

Palvelinlistan tarkistuksen jälkeen kaikki yhdistetyt palvelimet käydään läpi. Jokaisesta palvelimesta kerätään lista kaikista sen sisältämistä Xen APIn luokkien olioista kutsumalla XML-RPC:n kautta jokaisen Xen API -luokan `get_all`-metodia. Saatu lista käydään yksitellen läpi, ja jokaisen olion kaikki parametrit haetaan kutsumalla Xen APIn `get_record`-metodia, joka palauttaa viittauksen Perlin hash-muotoiseen tietorakenteeseen, jossa olion parametrit ovat avain-arvopareina.

Seuraavassa esimerkissä on kutsuttu Xen APIn VM-luokan get_record-metodia olemassa olevan Session-olion query-metodilla.

```
$my_record = $session->query('vm', 'get_records',
      '6488fbae-0ff0-f05d-cd5c-e6ffb7e95f27');
```

Session-olion query-metodille on annettu parametreina kyselyn kohteena oleva Xen APIn luokka, annettava komento sekä luokan objektin UUID-arvo. Session-olio liittää lopulliseen XML-RPC-kyselyyn yhteystunnisteen sekä muuntaa Xen APIn luokan nimen oikeaan muotoon, joka on tässä tapauksessa 'VM'.

Alla on esimerkki edellä olleen `get_records`-metodikutsun palauttamasta hash-taulusta. Täydelliset VM- ja VM_metrics-objektien hash-taulut ovat liitteessä 1.

```
$vm_record = {
      "uuid" => "6488fbae-0ff0-f05d-cd5c-e6ffb7e95f27",
      "name_label" => "clvm-testixeni",
      "power_state" => "Running",
      "PV_kernel" => "/boot/vmlinuz-2.6.18-4-xen-amd64",
      "memory_dynamic_max" => 201326592,
      "VCPUs_at_startup" => 1,
      "VCPUs_params" => {
            "cap" => 0,
            "weight" => 256
            },
      "VIFs" => ["ff9e6557-6b07-6a05-2900-06eb33ada670"],
      "other_config" => {}
};
```

Tietojen päivittäminen tietokantaan

Kun palvelinkomponentti on noutanut koko Xen-virtuaalikoneverkon tiedot, ne tallennetaan tietokantaan. Tallennus tapahtuu luomalla jokaiselle päivitettävälle Xen API -oliolle oma tietokantaolio, jonka rakentajan parametriksi annetaan edellisessä vaiheessa saatu hash-taulu. Tämän jälkeen kutsutaan tietokantaolion `insert_or_update`-metodia, joka hash-taulussa olleen `uuid`-kentän perusteella joko luo uuden tietueen olion tietokannan tauluun tai päivittää vanhan tietueen arvot. Seuraavassa esimerkissä päivitetään tietokantaan edellisessä vaiheessa haetun Xen APIn VM-olion tiedot tietokantaan.

```
my $db_vm = Model::Host::VM->new(
      lchashkeys( %{ $vm_record } )
);
$db_vm->insert_or_update;
```

Koska tietokannassa käytetään taulujen ja kenttien nimissä vain pieniä kirjaimia, joudutaan edellisessä esimerkissä käyttämään `lchashkeys`-funktiota, joka muuntaa parametrina annetun hash-taulun avaimet pieniksi kirjaimiksi. Koska `query`-metodin palauttama `$vm_record`-arvo on viittaus hash-tauluun, laitetaan se `%{}`-lohkon sisään, joka palauttaa viitatun hash-taulun.

Komentojonon käsittely

Komentojonon lukemisessa käytetään Manager-luokan `get_command_queues`-metodin sijaan `get_command_queues_iterator`-metodia, joka palauttaa 'command_queue'-taulun tietueeseen viittaavan iteraattorin. Koska komentojonossa olevat komennot halutaan käsitellä luontiajan mukaan vanhimmasta aloittaen, annetaan seuraavassa esimerkissä olevalle `get_command_queues_iterator`-metodille parametrina lajitteluperuste 'created ASC'.

```
my $item_iterator =
      Model::Command_queue::Manager->
            get_command_queues_iterator(
                  sort_by => 'created ASC'
);
my $item = $item_iterator->next;
```

Esimerkin viimeisellä rivillä oleva `next`-metodi palauttaa viittauksen ensimmäiseen tietueeseen. Koska komentojen suorituksesta halutaan myös lokimerkintä, luodaan 'command_log'-tauluun tietue kopioimalla siihen arvot nykyisen iteraattorin osoittamasta tietueesta ja lisäämällä tietueeseen tilatiedot.

```
my $log = Model::Command_log->new( $item->column_value_pairs
);
$log->uid(undef);
$log->status('Running');
$log->executed('now()');
$log->save;
```

Esimerkissä oleva `column_value_pairs`-funktio palauttaa tietokannan tietueet ja niiden arvot hash-tauluna. Koska mukana kopioituu myös tietueen `uid`-arvo, on se nollattava ennen lokin kirjoittamista tietokantaan, asettamalla kentän arvoksi `undef`. Esimerkissä käytetty `now()`-funktio on PostgreSQL:n funktio, joka palauttaa sen hetkisen ajan, kun tietoa kirjoitetaan tietokantaan.

Seuraavaksi tarkistetaan komennossa annettujen arvojen oikeellisuus. Jos komennossa tai jossain sen osassa on virhe, merkataan virhe tietokantaan, poistetaan komento 'command_queue'-taulusta ja luetaan seuraava komento. Seuraavassa esimerkissä on esitetty virhetilanne, jossa komennon kohdetta ei ole olemassa.

```
$log->finished('now()');
$log->status('Failure');
$log->value('Failure: Unknown object');
$log->save(changes_only => 1);
$item->delete;
$item = $item_iterator->next;
```

Esimerkissä save-metodille annettava `'changes_only => 1'`-parametri tarvitaan, jotta tietokantaan tallennettaisiin vain muuttuneiden kenttien arvot. Ilman parametria myös aiemmin määritetty `'$log->executed('now()')'` suoritettaisiin uudelleen ja `executed`-kentän arvo päivittyisi virheellisesti.

Komentojonotaulussa olevaa komentoa ei poisteta heti komennon suorituksen alussa, koska jos komponentin suoritus keskeytyy komennon suorittamisen aikana, saattaa komento tällöin jäädä suorittamatta, eikä sitä yritettäisi suorittaa komponentin uudelleenkäynnistyksen yhteydessä.

Seuraavaksi komento ohjataan oikealle oliolle. Komennon onnistuessa merkitään tietokantaan komento suoritetuksi, poistetaan se komentojonotaulusta ja luetaan seuraava komento.

```
$log->finished('now()');
$log->status('Success');
$log->save(changes_only => 1);
$item->delete;
$item = $item_iterator->next;
```

Komentojonon käsittelyn jälkeen komponentti odottaa ennalta määritetyn ajan ja aloittaa käsittelyn uudelleen kohdasta 'Palvelinlistan nouto'.

5.2 Komentojonon kautta annettavat komennot

Komentojonon ensimmäisessä versiossa tuetut komennot ja niiden parametrit on lueteltu taulukossa 7.

Taulukko 7 Komentojonon tukemat komennot sekä niiden parametrit

Komento	Parametrit	Selite
vm.create	ks. erillinen kappale	Uuden virtuaalikoneen luonti. Ainoa komento, jolle ei anneta object_uuid-arvoa
vm.destroy	delete_data [0/1]	Virtuaalikoneen tuhoaminen
vm.start	start_paused [0/1]	Virtuaalikoneen käynnistys
vm.shutdown	hard [0/1]	Virtuaalikoneen sammutus
vm.reboot	hard [0/1]	Virtuaalikoneen uudelleenkäynnistys
vm.pause		Virtuaalikoneen ajon tauotus
vm.unpause		Virtuaalikoneen ajon jatkaminen
vm.suspend		Virtuaalikoneen tilan siirto muistiin ja alasajo
vm.resume	start_paused [0/1]	Virtuaalikoneen tilan palautus muistista ja ajon jatkaminen
vm.migrate	Live [0/1], target <address>	Virtuaalikoneen siirto fyysiseltä palvelimelta toiselle
vm.change	mem <size_in_MB> name_label <name>	Virtuaalikoneen parametrien muutos

Komentojen parametrit annetaan avain-arvopareja sisältävässä tietorakenteessa ja tallennetaan tietokantaan JSON-muodossa, josta on esimerkki seuraavassa kappaleessa 'vm.create'.

Komennon ja parametrien lisäksi, komentojonotauluun pitää lisätä myös kohteena olevan palvelimen host-olion UUID kenttään 'host_uuid', sekä kohteena olevan Xen API -olion UUID kenttään 'object_uuid'. Ensimmäisessä versiossa ainut tuettu Xen API -olio on `VM`.

vm.create

'vm.create'-komentoa käytetään uuden virtuaalikoneen luontiin. Ensimmäisessä versiossa komento tukee vain taulukossa 8 listattuja parametreja.

Taulukko 8 'vm.create'-komennon parametrit, niiden selitteet sekä esimerkit

Parametri	Esimerkki	Selite
name_label	"test01"	Virtuaalikoneen nimi
memory_static_max	134217728	Muistimäärä
PV_kernel	"/boot/xen0-kernel	Käytettävä kernel-imagetiedosto
PV_args	"root=/dev/sda1 ro"	Kernelille annettavat parametrit
partition		Sisältää taulukon, jonka alkioina on virtuaalikoneen massamuistien tiedot
partition[n]->device	"sda1"	Virtualisoidun lohkolaitteen nimi
partition[n]->location	"phy:/dev/vg/test01_disc"	Polku lohkolaitteeseen, joka virtualisoidaan
partition[n]->mount	"/"	Virtuaalikoneen osio, johon virtualisoitu lohkolaite liitetään

Annettujen parametrien perusteella Xen APIin luodaan tarvittavat oliot ja luotu virtuaalikone lisätään automaattisesti tietokantaan. Esimerkkinä on JSON-muotoinen parametrilista, jonka avulla on luotu toimiva virtuaalikone.

```
{
      "name_label":"test01",
      "memory_static_max":"134217728",
      "PV_kernel":"/boot/xen0-linux-2.6.18-nonpae-6-xen-686",
      "PV_ramdisk":
            "/boot/initrd.img-2.6.18-nonpae-6-xen-686",
      "PV_args":"root=/dev/sda1 ro",
      "partition":
      [
            {     "device":"sda1",
                  "location":"phy:/dev/xenvg/test01_disk",
                  "mount":"/"
            },
            {     "device":"sda2",
                  "location":"phy:/dev/xenvg/test01_swap",
                  "mount":"swap"
            }
      ]
}
```

Muiden komentojono-komentojen parametrit

'vm.destroy'-komennolle annettava 'delete_data'-parametri määrittää tuhotaanko virtuaalikoneen Xen APIsta poistamisen yhteydessä myös kaikki virtuaalikoneeseen liittyvä data, kuten vapautetaanko lohkolaiteresurssit ja tuhotaanko virtuaalikoneen konfiguraatiotiedosto. Parametri ei ole pakollinen ja

sen oletusarvo on 'epätosi'. Toiminnallisuus on luotu täyttämään vaatimus 5: *Virtuaalikoneen poisto, jossa optio konfiguraation ja datan säilyttämiseen.*

'vm.start'- ja 'vm.resume'-komennoille annettava 'start_paused'-parametri määrittää, siirtyykö virtuaalikone välittömästi 'paused'-tilaan, käynnistyksen yhteydessä. Parametri ei ole pakollinen ja sen oletusarvo on 'epätosi'. Parametri luotiin, koska Xen API tukee sitä ja se oli helppo toteuttaa.

'vm.shutdown'- ja 'vm.reboot'-komennoille annettava 'hard'-parametri määrittää suoritetaanko komento välittömästi, vai normaalisti. Jos parametrin arvo on 'true', niin komento suoritetaan välittömästi, mikä tarkoittaa samaa kuin virran välitön katkaiseminen, ja 'vm.reboot'-komennon yhteydessä myös sen välitön palauttaminen. Parametri ei ole pakollinen ja sen oletusarvo on 'epätosi'. Parametri luotiin, koska Xen API tukee sitä ja se oli helppo toteuttaa.

'vm.migrate'-komennolle annettava 'live'-parametri määrittää, suoritetaanko virtuaalikoneen siirto lennossa, jolloin kontolliyhteys virtuaalikoneeseen on poikki vain pienen hetken, vai ajetaanko virtuaalikone ennen siirtoa 'suspended'-tilaan. Kontrolliyhteys on jälkimmäisessä tapauksessa poikki pidemmän ajan, mutta siirtotapahtuma on vakaampi. Parametri ei ole pakollinen ja sen oletusarvo on 'false'. Parametri luotiin, koska Xen API tukee sitä ja se oli helppo toteuttaa.

'vm.migrate'-komennolle annettava 'target'-parametri määrittää kohteena olevan palvelimen host-käyttöjärjestelmän osoitteen, johon virtuaalikone halutaan siirtää. Parametri on pakollinen.

'vm.change'-komennolle voi antaa kaksi parametria, joista 'mem'-parametri määrittää virtuaalikoneelle allokoitavan muistin määrän megatavuina, ja 'name_label'-paramerti määrittää virtuaalikoneen nimen.

6 LOPPUTUOTTEEN ARVIOINTIA

Työn tulos oli käyttökelpoinen Xen-virtuaalikoneympäristön hallintaan ja valvontaan tarkoitettu palvelinkomponentti, jonka avulla päästään kehittämään laajempaa Axxion VServer Manager -ohjelmistoa.

Komponentin valmistuminen venyi huomattavasti alkuperäisestä suunnitelmasta, mutta loppujen lopuksi kaikki osapuolet olivat tyytyväisiä työn tulokseen.

Komponentille on jatkokehitysideoita ja tarvetta, joten kehitys jatkuu lähitulevaisuudessa. Näistä jatkokehitysideoista kerrotaan lisää kappaleessa 6.3.

6.1 Aikataulu

Työhön kokonaisuudessaan kuluva aika oli arvioitu liian pieneksi, mutta loppujen lopuksi se ei ollut haittaava tekijä, koska tärkeämpää oli saada työ tehtyä kunnolla. Aikataulun venyminen ei myöskään haitannut laajemman Axxion VServer Manager -ohjelmiston kehitystä, sillä muista projekteista johtuen, sovelluksen muilla kehittäjillä ei ollut vielä aikaa sovelluksen kehittämiseen.

Työn venymiseen oli varauduttu, sillä ohjelmistoprojektien aikataulut tahtovat venyä syystä tai toisesta. Loppua kohden työtahti kuitenkin nopeutui, koska tekemisen ohessa oppi lisää, eikä komponentista yritetty tehdä liian monimutkaista. Ohjelmakoodin kommentointi on vielä kesken, koska koodi muuttui paljon vielä toteutuksen loppuvaiheessa. Kommentoinnin puutteesta ei toistaiseksi ole haittaa, koska komponentilla on vain yksi kehittäjä, mutta lopullisen version on oltava kattavasti kommentoitu Perlin kommentointityylejä noudattaen.

Projekti palvelinkomponentin parissa kesti yhteensä viisi kuukautta. Ensimmäiset kaksi kuukautta käytettiin esitutkimukseen ja toiminnallisen määrittelyn kirjoittamiseen. Komponentin suunnittelu ja toteutus kesti viimeiset kolme kuukautta.

6.2 Vaatimusten täyttyminen

Koska komponentin toteutukseen meni huomattavasti enemmän aikaa kuin määrittelyvaiheessa oltiin arvioitu, ehdittiin komponentin ensimmäiseen versioon toteuttamaan vain osa vaadituista toiminnallisuuksista. Alla on listattu vaatimuksista, joiden toteutus siirrettiin komponentin seuraavaan versioon.

Vaatimus 4: Virtuaalikoneen luonti, käyttöjärjestelmän asennus ja konfigurointi
Vaatimuksesta jäi toteuttamatta käyttöjärjestelmän asennus ja konfigurointi.

Vaatimus 7: Virtuaalikoneiden käytettävissä olevien prosessoriytimien määritys
Projektin toteutusvaiheessa tätä toiminnallisuutta ei voitu enää testata, koska kaikki moniydinpalvelimet olivat jo tuotantokäytössä, eikä testipalvelimessa ollut kuin yksi ydin. Toissijaisena vaatimuksena tämä jätettiin pois nykyisestä versiosta.

Vaatimus 16: Palvelinkomponentti valvoo virtuaalikoneiden tilaa ja hälyttää virhetilanteiden ilmetessä

Vaatimus 17: Palvelinkomponentti valvoo virtuaalikoneiden resurssien käyttöä ja hälyttää resurssien käydessä vähiin
Koska Xen APIn kautta ei päästy käsiksi läheskään kaikkiin dokumentaatiossa mainittuihin resurssiparametreihin, joudutaan suurin osa resurssienkäyttötiedoista hakemaan erillisillä sovelluksilla. Ongelman ratkaisu siirrettiin seuraavaan versioon.

Vaatimus 19: Palvelinkomponentin täytyy tukea eri käyttäjäoikeustasoja
Eri käyttäjätasojen mahdollisuus otettiin huomioon jo komponentin suunnitteluvaiheessa, mutta ensimmäisessä versiossa ei tueta kuin yhtä käyttäjää, jolla on kaikki oikeudet.

6.3 Jatkokehitysideoita

Vaatimuksessa 16 määritetty toiminto virtuaalikoneiden tilan valvontaan ja hälyttämiseen voi sisältää logiikkaa vikatilanteiden automaattiseen korjaamiseen.

Jokaiselle fyysiselle palvelimelle voidaan luoda erillinen komponentti, joka lähettää palvelinkomponentille vain Xen-ympäristössä tapahtuneet muutokset, jolloin tietojen päivitys palvelimilta olisi nopeampaa.

7 YHTEENVETO

Työn esitutkimus, suunnittelu ja toteutus oli hyvin aikaa vievää kokeilujen ja uuden oppimisen vuoksi. Komponentin toteutus ei onnistunut sovittujen aikataulujen mukaisesti, mutta koko ohjelmistoprojektin mittakaavassa se ei haittaa, koska suunnitteluun ja tutkimiseen käytetty aika helpottaa projektin myöhempiä toteutusvaiheita.

Toteutettu komponentti otetaan käyttöön Axxion VServer Manager -ohjelmistossa. Kokonaisuudessaan projekti palvelinkomponentin parissa onnistui hyvin ja vastasi sisällöltään tyypillisiä ohjelmistoinsinöörin työtehtäviä.

Työn aikana opin paljon uutta Perl-kielestä ja sen eri kirjastoista, XML-RPC- ja verkkoprotokollista sekä tietokannoista. Komponentin toteutuksessa käytetyt teknologiavalinnat osoittautuivat oikeiksi, koska komponentin ensimmäisen version valmistuessa libvirt on edelleen vajaatoiminen eikä Xen Management APIn sisältö ole merkittävästi muuttunut.

LÄHDELUETTELO

Sähköiset lähteet

1 Jyrki Muukkonen, *Xen virtuaalikonemoottori – Käyttöjärjestelmien uudet haasteet-seminaari kevät 2006* [PDF-dokumentti].
[viitattu 10.3.2008] Saatavissa:
http://www.cs.helsinki.fi/u/kraatika/Opetus/OS06/JyrkiMuukkonen.pdf

2 Wikipedia – *Xen* [www-sivu].
[viitattu 22.5.2008] Saatavissa:
http://en.wikipedia.org/wiki/Xen

3 *libvirt: The virtualization API* [www-sivu]
[viitattu 23.05.2008] Saatavissa:
http://libvirt.org

4 Xen Wiki – *Hypervisor* [www-sivu]
[viitattu 24.05.2008] Saatavissa:
http://wiki.xensource.com/xenwiki/hypervisor

5 Wikipedia – *XML-RPC* [www-sivu]
[viitattu 25.05.2008] Saatavissa:
http://en.wikipedia.org/wiki/XML-RPC

6 Mellor, Sharp, Scott, *Xen Management API, revision 1.0.1* [pdf-dokumentti].
[viitattu 26.05.2008] Saatavissa Xen 3.2.1 lähdekoodin mukana.

7 Wikipedia – *Universally Unique Identifier* [www-sivu]
[viitattu 26.05.2008] Saatavissa:
http://en.wikipedia.org/wiki/UUID

VM- ja VM_metrics-olioiden esimerkkiparametrit

Xen APIn VM- ja VM_metrics-olioiden get_record-metodin palauttama tietorakenne Perlin Data::Dumper-käsittelijän antamassa muodossa.

```
$vm_record = {
          "power_state" => "Running",
          "uuid" => "6488fbae-0ff0-f05d-cd5c-e6ffb7e95f27",
          "VIFs" => [
                      #0
                      "ff9e6557-6b07-6a05-2900-06eb33ada670"
                    ],
          "PV_ramdisk" => "/boot/initrd.img-2.6.18-4-xen-amd64",
          "memory_dynamic_min" => 201326592,
          "memory_static_min" => 0,
          "VCPUs_max" => 1,
          "name_description" => "clvm-testixeni",
          "HVM_boot_params" => {},
          "memory_static_max" => 201326592,
          "PV_bootloader_args" => "",
          "security_label" => "",
          "consoles" => [
                          #0
                          "6a395194-a09b-ded3-d40f-46ec8905515e"
                        ],
          "HVM_boot_policy" => "",
          "is_control_domain" => 0,
          "tools_version" => {},
          "VBDs" => [
                      #0
                      "dd10b5b5-d6f4-4022-82a2-36ee0b260953"
                    ],
          "is_a_template" => 0,
          "resident_on" => "3a6d6da8-5a8a-3e0d-819c-b9fcf938c5b1",
          "domid" => 1,
          "other_config" => {},
          "PV_args" => "root=/dev/hda1 ro 4",
          "VCPUs_params" => {
                              "cap" => 0,
                              "weight" => 256
                            },
          "user_version" => 1,
          "actions_after_crash" => "restart",
          "actions_after_shutdown" => "destroy",
          "VTPMs" => [],
          "metrics" => "33a22e01-2a1d-9af8-9025-f5d71cfa053d",
          "platform" => {},
          "crash_dumps" => [],
          "PV_bootloader" => "",
          "actions_after_reboot" => "restart",
          "PV_kernel" => "/boot/vmlinuz-2.6.18-4-xen-amd64",
          "auto_power_on" => 0,
          "actions_after_suspend" => "restart",
          "PCI_bus" => "",
          "VCPUs_at_startup" => 1,
          "name_label" => "clvm-testixeni",
          "memory_dynamic_max" => 201326592
        };
```

```
$vm_metrics_record = {
          "last_updated" => "20080320T20:07:31",
          "VCPUs_params" => {
                               "cap" => 0,
                               "cpumap0" =>
"0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28
,29,30,31",
                               "weight" => 256
                             },
          "start_time" => "20080210T16:51:26",
          "uuid" => "33a22e01-2a1d-9af8-9025-f5d71cfa053d",
          "memory_actual" => 201248768,
          "state" => [
                       #0
                       "blocked"
                     ],
          "VCPUs_CPU" => {
                           0 => 1
                         },
          "VCPUs_utilisation" => {
                                   0 => "0.00015649826507733194"
                                 },
          "VCPUs_number" => 1,
          "VCPUs_flags" => {
                             0 => [
                                    #0
                                    "blocked",
                                    #1
                                    "online"
                                  ]
                           }
        };
```