TEKNIKAN JA LIKENTEEN TOIMIALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

VIRTUAALIKONEIDEN HALLINTA OHJELMOINTIRAJAPINTAA KÄYTTÄEN

Työn tekijä: Tomi Tuhkanen
Työn valvoja: Matti Luukkainen
Työn ohjaaja: Matti Luukkainen

Työ hyväksytty: __. __. 2007

Matti Luukkainen
yliopettaja
ALKULAUSE

Haluan kiittää vanhempiani, jotka eivät rajoittaneet liikaa pelaamistani lapsena, sekä enoani Commodore 64 -tietokoneesta.

Espoossa 12.11.2007

Tomi Tuhkanen
Työssä tutustutaan VMwaren Vix- ja VmCOM-rajapintoihin sekä luodaan ohjelmia, joilla käyttäjä voi hallita VMware Server –ohjelmaa rajapintojen avulla.

Vix-rajapinnan avulla luodaan C++-kielellä komentoliittymä-ohjelma, joka on suorassa yhteydessä VMware Server –ohjelmana.

Työssä luodaan palvelin joka käyttää yhteydenhallintaan käyttäjän ja palvelimen välillä TCP-teknologiaa. Palvelin keskustelee VMware Serverin kanssa käyttäen VmCOM-rajapintaa.

Windows käsittäjä-yhdistelmälle luodaan Client-ohjelma. Ohjelmasta luodaan suoraan VMware Serverin yhteydessä oleva versio sekä palvelinpohjainen versio.

ASP.NET:lla luodaan dynaamiset web-sivut, joka käyttää myös yhteydenhallintaa palvelinta. Dynaamisten web-sivujen avulla voidaan myös muokata SQL Serverillä olevia käyttäjien tietotauluja.

Windows–ohjelmien dynaamiset web-sivut sekä TCP-palvelin on kirjoitettu C#-ohjelmointikielellä.

Avainsanat: VMware, VMware Server, Virtuaalikone, Vix API, VmCOM API
**ABSTRACT**

<table>
<thead>
<tr>
<th>Name: Tomi Tuhkanen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title: Managing virtual machines using programming API</td>
</tr>
<tr>
<td>Date: 12.11.2007</td>
</tr>
<tr>
<td>Department: Information Technology</td>
</tr>
<tr>
<td>Instructor: Matti Luukkainen</td>
</tr>
<tr>
<td>Supervisor: Matti Luukkainen</td>
</tr>
</tbody>
</table>

Purpose of this thesis was to examine VMware’s Vix and VmCOM programming API’s and to create applications and scripts to control VMware Server with those interfaces.

In this thesis was created 4 different applications and dynamic web-pages.

Command prompt application was created with C++, that is connected straight to VMware Server with usage of Vix API.

TCP-server, that works between applications and VMware Server. It communicates with VMware Server with usage of VmCOM API.

Two versions of Windows Forms application. One is connected straight to VMware Server with usage of VmCOM API and one uses TCP-server.

Dynamic web-pages with ASP.NET. Web-pages also use TCP-server to connect to VMware Server. With dynamic web-pages people with admin rights can maintain user tables in SQL Server.

Windows applications, TCP-server and code for dynamic web-pages were written with C#.

Keywords: VMware, VMware Server, Virtual Machine, Vix API, VmCOM API
# AS-setti 1

## 1 JOHDANTO

## 2 VIRTUALISOINTI

2.1 Virtuaalikonevalvoja

2.2 Rautavirtuaalikoneet

2.3 Ohjemistovirtuaalikoneet

2.4 Laitteiden virtualisointi

2.5 Ohjelmien virtualisointi

2.6 Esityksen virtualisointi

2.7 Hyödyt
   - Ositus
   - Eristyvyys
   - Kapselointi
   - Muokattavuus

## 3 VMWARE

3.1 VMware Server

3.2 Ohjemointirajapinnat

## 4 OHJELMISTOJEN ASENNUS

4.1 Server

4.2 Programming API:n (Vix) käyttöönotto
   - Clientin käänäminen VMware Server asennettuna
   - Clientin käänäminen ilman VMware Serveria

4.3 VMware COM API:n käyttöönotto

## 5 HALLINTAOHJELMIEN JA –SKRIPTIEN SUUNNITTELU JA TOTEUTTAMINEN
5.1 Ohjelmointi- ja perustukset

5.2 Tavoitteet

5.3 Programmin API-skriptit

5.4 Vix API -ohjelma

5.5 VmCOM API-skriptit

5.6 VmCOM API stand-alone ohjelma

5.7 TCP-palvelin

5.8 VmCOM API palvelintyöryhmäohjelma

5.9 ASP.NET Dynaamiset web-sivut

5.10 Logger

6 YHTEENVETO

6.1 Ongelmat

6.2 Parannus- ja asiakirjatukset

VIITELUETTELO
1 JOHDANTO

Virtualisoinnin yleistyttyä työpaikoilla ja kouluissa virtualisoinnin tuomiin mahdollisuksiin on tutustuttava yhä paremmin. Työssä käsitetään lyhyesti, mitä virtualisointi on ja miksi virtualisointia käytetään.

Työn pääpaino on tutustua VMwaren Vix- ja VmCOM-ohjelmointirajapintoihin, sekä luoda ohjelmia, joilla käyttäjä voi etähallita VMware Server -ohjelmaa rajapintojen avulla.

Ohjelmat joilla VMware Server –ohjelmaa voi hallita etänä ovat yleensä suunniteltu ylläpitäjien käyttöön, joten ne ovat erittäin monipuolisia ja samalla kalliita. Näinöllä ne eivät sovellu jo pelkästään hintansa puolesta tavallisille käyttäjille. Työssä luodaan ohjelmia, joiden tarkoitus on antaa tavalliselle käyttäjälle yleiset toiminnot, joiden avulla voidaan hallita VMware Serverillä olevia virtuaalikoneita.

2 VIRTUALISOINTI


2.1 Virtuaalikonevalvoja

Virtuaalikonevalvoja, eng. Virtual Machine Monitor (VMM) tai Hypervisor, on ohjelmistokerroks, jota tarvitaan virtualisointiin. Tämä ohjelmistokerros luo illuusio "oikeasta" tietokoneesta tai resurssista, jota aiotaan virtualisoida. Virtuaalikonevalvoja voi toimia joko normaallissa käyttöjärjestelmässä (hosted-arkkitehtuuri) tai se voi toimia ilman käyttöjärjestelmää, jolloin se on itsessään käyttöjärjestelmä (Bare-Metal-arkkitehtuuri). [5]
2.2 Rautavirtuaalikoneet


![A VMware virtual machine](image)

**Kuva 1. Ohjelmistosäiliö**

Käyttöjärjestelmät, ohjelmat tai muut tietokoneet eivät pysty erottamaan tavallista tietokonetta virtuaalikoneesta. Virtuaalikone luuleekin olevansa aivan tavallinen tietokone. [4.]

2.3 Ohjemistovirtuaalikoneet

Ohjemistovirtuaalikoneella tarkoitetaan ohjelmaa, joka eristää käytössä olevan ohjelman käyttöjärjestelmältä. Ohjemistovirtuaalikoneista on yleensä tehty versiot eri tietokonealustoille, joten ohjelmat, jotka on kirjoitettu kyseiselle ohjemistovirtuaalikoneelle, toimivat kaikilla alustoilla. Tässä säästytään vaivalta, jolloin jokainen yksittäinen ohjelma tulisi kääntyä jokaiselle alustalle. Ohjemistovirtuaalikoneesta hyvä esimerkki on emulaattori, jolla voi ajaa toiselle käyttöjärjestelmällä tai tietokoneelle kehitettyjä ohjelmia. [8.]

### 2.4 Laitteiden virtualisointi

Laitteiden virtualisoinnilla tarkoitetaan laitteen tai laitteiden resurssien jakamista tai yhdistämistä eri kokonaisuuksiksi. Osituksessa yksi resurssi jaetaan moneksi resurssiksi. Useimmissa tietokoneenkäyttäjille tutuun muoto laitteiden virtualisoinnista on kovalevyn jakaminen pienempiin osiin (partitioihin) (kuva 3).
Laitteiden resursseja voi myös yhdistellä suuremmiksi konaisuuksiksi. Esimerkiksi verkkolaitteet voivat yhdistää useita kanavia suuremmaksi kanavaksi, ja RAID-levyjärjestelmät yhdistävät useita levyjärjestelmiä suureksi levyksi (kuva 4). Tietokoneita yhdistettäessä käytetään nimitystä klusteri, mutta myös siinä tapauksessa luodaan yksi tehokkaampi tietokone virtualisoinnin avulla. [5.]

Virtualisoinnin avulla voidaan myös piilottaa tietoa palveluilta, joilla ei ole oikeuksia sitä käyttää. [11.]
2.5 Ohjelmien virtualisointi

Ohjelmien virtualisointi eristää ohjelman konfiguraatiotason käyttöjärjestelmästä. Tämän tekniikan avulla ohjelmat voivat olla asennettuna keskitetyille palvelimille sen sijaan, että jokainen ohjelma olisi asennettuna jokaiselle koneelle.

*Kuva 5. Ohjelmat asennettuna käyttöjärjestelmään*


*Kuva 6. Ohjelmat asennettuna omiin virtuaaliympäristöihin*
2.6 Esityksen virtualisointi

Esityksen virtualisointi eristää prosessoinnin grafiikasta ja muista loppukäyttäjän siirrännäisistä, kuten hiirestä ja näppäimistöstä. Ohjelma on asennettuna palvelimelle, mutta sitä käytetään etänä toiselta koneelta (kuva 7).

Kuva 7. Ohjelmat ovat käynnissä erillisellä palvelimella

Palvelin luo virtuaalisia istuntoja, jossa käynnissä olevat ohjelmat projisoivat käyttölittymänsä etänä toiselle koneelle. Istunnoissa voi olla käynnissä yhdestä ohjelmasta koko työpöydän kattavaan ohjelmistoon.

Monet istunnot voivat käyttää samanaikaisesti samoja palvelimelle asennettuja ohjelmia. [11.]

2.7 Hyödyt

Virtuaalikoneiden pääasialliset hyödyt tulevat yritykselle taloudellisina hyötyinä. Ohjelmistokehityksessä varsinkin testaus hyötyy virtualisoinnin antamista mahdollisuuksista.

2.7.1 Ositus

Monilla yrityksillä on lähes kaikille tehtäville oma palvelimensa (kuva 8). Tällöin palvelimet käyttävät arviolta vain noin 5-30 % tehostaan. Jokainen palvelin myös kuluttaa paljon virtaa, vaikka ne eivät toimiskaan mahdollisimman tehokkaasti.
Kuva 8. Perinteinen tapa, erillinen palvelin kaikille järjestelmiille [6.]

Virtualisoinnilla voidaan palvelimet keskittää yhdelle tehokkaammalle palvelimelle, jolloin yhdestä koneesta saadaan maksimaalinen teho / käyttösuhte (kuva 9).

Resurssien siirtely eniten tarvitsevalle helpottuu virtualisoinnin avulla huomattavasti, sillä enää ei tarvitse alustaa turhia koneita ja asentaa uusia konfiguraatioita. Vanhan resurssin voi sammuttaa ja asentaa uuden resurssin toimimaan sen tilalle. Mikäli vanhat resurssit pitää jostain syystä palauttaa, takaisinpäin vaihtaminen käy yhtä helposti.
Kuva 9. Sama tilanne virtualisoinnilla [6.]

Samalla fyysisellä tietokoneella voi olla samaan aikaan monia käyttöjärjestelmiä. Näin voidaan käyttää esimerkiksi ohjelmaversioita, jotka eivät nykyisellä käyttöjärjestelmällä enää toimisi.

Ostituksessa tietokoneen resurssejä käsitellään poolina, josta ohjelmallisesti jaetaan resursseja tarvitseville virtuaalikoneille. Tämän avulla vähän käytettävä virtuaalikone ei hidasta pääkoneen toimintaa juuri ollenkaan. [5;10.]

2.7.2 Eristvyys


Eristvyysdellä saadaan luotua turvallinen ympäristö, ns. hiekkalaatikko (sandbox). Tällöin esimerkiksi haitallisten ja turvattomien ohjelmien testauksessa ohjelmat eivät pääse vaikuttamaan muihin koneisiin eikä
virtuaalikoneen kaatuminen ei vaikuta muiden koneiden toimintaa lainkaan. [5.]

Ohjelmien ollessa eristettyinä toisistaan ne voivat käyttää eri versioita ohjelmistokirjasoista, jotka tavallisesti asentuisivat toistensa päälle käyttöjärjestelmässä näin estäen molempien ohjelmien samanaikaisen käytön. [11.]

2.7.3 Kapselointi

Virtuaalinen tietokone tallentuu yhteen erilliseen tiedostoon, joten sitä on helppo käsitellä, siirtää tai kopioida.

Testitapauksissa, joissa tarvitaan monia täysin identtisiä kokoonpanoja, on helpoa luoda tekemällä yhdestä virtuaalikoneesta monia kopioita. Oman virtuaalikoneen saa myös helposti kulkemaan aina mukana, esim. usb-muistitikulla.

2.7.4 Muokattavuus

Virtuaalikoneen resursseista saadaan muokattua tasan omia toiveita vastaava. Virtuaalikoneet voivat luoda illusioon raudasta tai rautakonfiguraatioista joita oikeasti ei ole käytössä. Nämä testitilanne saadaan luotua täysin vaaditulle konfiguraatiolle. [5.]
3 VMWARE

VMware on vuonna 1998 perustetty yhdysvaltalainen yritys, joka on yksi maailman johtavista virtualisointisovellusten toimittajista x86-pohjaisille palvelimille ja työpöytäkoneille.

3.1 VMware Server

Kuva 10. VMware Server logo

VMware Server keskittyy rautavirtuaalikoneisiin ja sillä voi muokata, sekä luoda virtuaalikoneita. VMware Server perustuu hosted –arkkitehtuuriin, eli se tarvitsee toimia käyttöjärjestelmän käyttöjärjestelmän. VMware Server toimii sekä virtuaalikonevalvojana että apuohjelmana, joka on käynnissä käyttöjärjestelmän päällä eristäen virtuaalikonevalvojan käyttöjärjestelmästä, jolloin se voi käyttää resursssejaan paremmin virtuaalikoneiden hallintaan (kuva 11). [5.]

Kuva 11. VMware Server jakaa fyysisen koneen moniin virtuaalisiin koneisiin [3.]
3.2 Ohjelmointirajapinnat

VMwaren Programming API on ohjelmointirajapinta, jonka avulla käyttäjät voivat kirjoittaa skripteja ja ohjelmia virtuaalisten koneiden hallintaan ja ohjaamiseen. Rajapinta on kirjoitettu C-kielellä, ja se toimii Microsoft Windows- ja Linux-alustoilla. Ohjelmointirajapinta on suunniteltu siten, että sitä voivat käyttää VMware serverin käyttäjät sekä ohjelmistojen kehittäjät. VMwaren Programming API:stä käytetään usein nimeä Vix. [2, s. 1.]

VMware VmCOM API -rajapinta on myös kirjoitettu C-kielellä, mutta se perustuu Component Object Modeliin (COM). COM on pääosin vain Microsoft Windows käytössä. Insinööriyön pääasiallinen koodi tullaan tekemään VmCOM API:a käyttäen, C#-ohjelmointikielellä, jolloin dynaamiset web-sivut voidaan toteuttaa ASP.NET teknologiailla. [9, s. 7.]

VMwarella on myös ohjelmointirajapinta Perl-ohjelmointikielelle, VmPerl API. Rajapintaa voi hyödyntää Windows- ja Linux-alustoilla. Työssä ei tutustuta VmPerl API -rajapintaan. [9, s. 7.]

VMware Tools ei ole rajapinta, vaan paketti työkaluja ja ajureita, jotka edesauttavat virtuaalikytöjärjestelmän suorituskykyä ja toiminnallisuutta. Paketti sisältää muunmuassa hiiri- ja näytönohjainajurit käyttöjärjestelmille. [2, s. 36.]

4 OHJELMISTOJEN ASENNUS

4.1 Server


Ohjelma asennettiin Windows Vista Home Premium 32-bittiseen käyttöjärjestelmään. Asennettavassa koneessa on AMD Turion(tm) 64 X2
Mobile Technology TL-50 (1.60GHz) suoritin ja 1982 megatavua keskusmuistia.

Mikäli Microsoft Internet Information Server (IIS) ei ole asennettuna käyttöjärjestelmään, ei VMware Management Interfacea voida asentaa. Management Interfacen avulla käyttöjärjestelmän valvoja voi hallita virtuaalikoneita etänä verkon yli.

Asennusopas löytyy liitteestä 1.

4.2 Programming API:n (Vix) käyttöönotto

Vix sisältää komponentit koneelle, jolle VMware Server on asennettu, sekä koneelle, jolla aiotaan vain käyttää API:lla luotuja skriptejä tai ohjelmia.


4.2.1 Clientin käänäminen VMware Server asennettuna

1. Lisää Client–ohjelman kooditiedoston alkuun include-määritys

```
#include "vix.h"
```

2. Aseta kehitysypäräistön include pathiin [asennushakemisto]\VMware\Vix\ Project -> Properties -> Configuration Properties
Kuva 12. Project Property Page

Lisää `[asennushakemisto]\VMware Vix` kohtaan, C/C++ -> General -> Additional Include Directories.

3. Aseta Windows-ympäristö lisäämään DLL-tiedostot `[asennushakemisto]\VMware VIX\` hakemistosta.

Kuva 13. System Properties
System Propertiesty lista Enviromental Variables. System Properties löytyy My Computerin alta (My computer -> Properties).

Kuva 14. Enviroment Variables

Enviromential Variablesta lisää PATH variableen [asennushakemisto]\VMware Vix (erota hakemistot puolipisteellä).

Vaihtoehtoisesti voit myös kopioida vix.dll, ssleay32.dll ja libeay32.dll tiedostot kääntäjän debug kansioon.

4. Aseta kääntäjä yhdistämään vix.lib-tiedostoon staattisesti

Project -> Properties -> Configuration Properties
Lisää \(\text{asennushakemisto}\)/VMware Vix\ kohtaan, Linker -> General -> Additional Library Directories.

Lisää Vix.lib kohtaan, Linker -> Input -> Additional Dependecies.
4.2.2 Clientin kääntäminen ilman VMware Serveriä

Tärkeät tiedostot ovat:

- vix.h
- vm_basic_types.h
- vix.lib
- vix.dll
- ssleay32.dll
- libeay32.dll.

Tiedostot löytyvät koneelta, johon VMware Server on asennettuna hakemistosta [asennushakemisto]\VMware VIX\tiedostot kopioidaan vapaastivalittavaan kansioon asiakas koneeseen. Katso malliesimerkit kohdasta 4.2.1.

1. Lisätään ohjelman kooditiedostoon include-määritys.

2. Asetetaan include path kehitysympäristöön hakemistoon johon kopioidaan vaadittavat tiedostot.

3. Asetetaan kääntäjä lisäämään DLL-tiedostot hakemistosta johon kopioidaan vaadittavat tiedostot.

4. Aseta kääntäjä yhdistämään vix.lib tiedostoon staattisesti.

Client-ohjelma käynnistetään hakemistosta, joka sisältää tarvittavat DLL-tiedostot.

Kolme DLL-tiedostoa voidaan myös asentaa system hakemistoon, jolloin ohjelma voidaan käynnistää mistä tahansa.

4.3 VMware VmCOM API:n käyttöönotto

Tarvitaan seuraavat tiedostot:

- VmCOM.dll
- VmControlLib.dll
- libeay32.dll
- ssleay32.dll.

Tiedostot löytyvät hakemistosta [asennushakemisto]\VMware VmCOM Scripting API\.

1. Asetetaan ympäristö lisäämään DLL-tiedostot [asennushakemisto]\VMware VmCOM Scripting API\ hakemistosta tai kopioidaan tiedostot projektin debug-kansioon.

2. Rekisteröidään komentoliittymästä VmCOM.dll komennolla regsvr32 "[tiedoston hakemisto]\vmcom.dll".

3. Lisätään referenssi VMware VmCOM 1.0 Type Libraryyn

   Project -> Add Reference…

   ![Add Reference](image)

   Kuva 17. Add Reference

4. Lisätään using rivi ohjelman kooditiedoston alkuun

   ```
   using VMCOMLib;
   ```
5 HALLINTAOHJELMIEN JA –SKRIPTIEN SUUNNITTELU JA TOTEUTTAMINEN

Ohjelmien lähdekoodi löytyy liitteenä olevalta cd-levyltä.

Koodiesimerkissä

```c
// Koodia
```

tarkoittaa, että esimerkistä on jätetty tilan säästämisen takia toistavaa tai epäolennaista koodia pois.

5.1 Ohjelmointiälympäristöt

![Visual C# 2005 logo](image)

*Kuva 18. Visual C# 2005 logo*


![Visual C++-ikkuna](image)

*Kuva 19. Visual C++-ikkuna*


Kuva 20. Ohjelmien väliset suhteet

Tilanteessa jossa ohjelma on suorassa yhteydessä VMware Serveriin, käyttäjä tietää palvelinkoneen IP-osoitteen, sekä käyttäjällä on oltava...

Kuva 21. Ilman välityspalvelinta


TCP-palvelin hakee käyttäjien tiedot tietokannasta, joka on tallennettu Microsoft SQL Serverille. Tietokannassa on taulut käyttäjille (users), VMware Serverille (servers) ja virtuaalikoneille (virtualmachines).

**USERS** (id, username, firstname, lastname, password, server, admin)

**SERVERS** (id, name, ip, username, password)

**VIRTUALMACHINES** (id, name, folder, owner)

Users-taulu:

<table>
<thead>
<tr>
<th>ID</th>
<th>UserName</th>
<th>FirstName</th>
<th>LastName</th>
<th>Password</th>
<th>Server</th>
<th>Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sami</td>
<td>Sami</td>
<td>Niemelä</td>
<td>G4hxy</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>tiinap</td>
<td>Tiina</td>
<td>Penttilä</td>
<td>oi4Gjh</td>
<td>2</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>jussiv</td>
<td>Jussi</td>
<td>Vihavainen</td>
<td>9h7jfa</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>4</td>
<td>admin</td>
<td>Antti</td>
<td>Jussila</td>
<td>root</td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>

Servers-taulu:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>IP</th>
<th>UserName</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U315/2</td>
<td>212.24.56.1</td>
<td>admin</td>
<td>root</td>
</tr>
<tr>
<td>2</td>
<td>A204/1</td>
<td>212.21.58.1</td>
<td>admin2</td>
<td>root2</td>
</tr>
</tbody>
</table>

Virtualmachines-taulu:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Folder</th>
<th>Owner</th>
</tr>
</thead>
</table>
Tavoitteena oli myös tutkia, olisiko käyttäjän mahdollista ottaa yhteyttä palvelinkoneella sijaitsevaan VMware Server -ohjelmaan, vaikka käyttäjällä itsellään ei olisi oikeuksia palvelinkoneelle. Tämä osoittautui kuitenkin mahdottomaksi, sillä ohjelmointirajapinnan Connect()-metodi vaatii, että kirjautuvalla käyttäjällä on oikeudet käynnistää VMware Server, jolloin myös samalla käyttäjällä on oltava oikeudet kirjautua palvelimeen sisään.

5.3 Programmin API-skriptit

Kappaleessa esitellään yleisimmät ja Vix API -ohjelmassa (katso luku 5.4) käytettyjä yhteydenhallintafunktioita Vix:llä toteutettuna.

5.3.1 Handlet


```c
VixHandle hostHandle;
VixHandle jobHandle;
VixHandle vmHandle;
```

5.3.2 Yhteydenotto VMware Serveriin

Yhteydenotto palvelimiin tapahtuu VixHost_Connect()-funktiolla. Palvelimeen voi ottaa yhteyden kolmella tavalla:

- paikalliseen VMware Serveriin tämänhetkisellä käyttäjällä, jolloin hostNamen, userName ja passwordin arvoiksi asetetaan NULL.
- paikalliseen VMware Serveriin määritetyllä käyttäjällä, jolloin hostName on NULL, mutta userName ja password tulee olla määritettyinä.
• määritettyyn VMware Serveriin määritetyllä käyttäjällä, jolloin 
  hostName, userName ja password tulee olla määritettyinä.

HostPort on vakiona 902, mutta jotkut asennukset saattavat asentua eri 
portiin, mikäli 902 on ollut jo aiemmin käytössä.

VixHost_Connect() on asynkroninen funktio, joten funktio palauttaa signaalin 
kun se on valmis. Funktionkutsu täytyy lopettaa takaisinkutsulla (callback 
function) tai VixJob_Wait()-funktiola, joka odottaa signaalia (katso 5.3.6).

```
jobHandle = VixHost_Connect(VIX_API_VERSION, 
  VIX_SERVICEPROVIDER_VMWARE_SERVER, 
  NULL, // hostName 
  902, // hostPort 
  NULL, // userName 
  NULL, // password 
  0, // options 
  // propertyListHandle 
  NULL, // callbackProc 
  NULL); // clientData 
```

// Odotetaan jobHandleen valmistumista
err = VixJob_Wait(jobHandle, // Työhandle
  VIXPROPERTY_JOB_RESULT_HANDLE, 
  &hostHandle, // Palvelinhandle
  VIXPROPERTY_NONE);

5.3.3 Virtuaalikoneen rekisteröinti

VMware Server vaatii, että virtuaalikoneet löytyvät sen omasta 
virtuaalikonehakemistosta ennenkuin niitä voi käyttää. Rekisteröityminen 
tapahtuu VixHost_Register()-funktioilla.

```
jobHandle = VixHost_RegisterVM(hostHandle, 
  vmxFilePath, // Virtuaalikoneen tiedosto 
  NULL, // callbackProc 
  NULL); // clientData
```

Jos virtuaalikoneen poistaa hakemistosta, sille ei voi suorittaa mitään 
operaatioita. Virtuaalikoneen poisto hakemistosta on hyvä tapa estää 
muutokset virtuaalikoneeseen, mikäli virtuaalikone ei ole usein käytössä.

5.3.4 Handlen ottaminen virtuaalikoneeseen

Suurin osa virtuaalikoneiden operaatioista vaatii handlen, jotta virtuaalikone 
voidaan identifioida. Kun operaatio on suoritettu, handle tulee vapauttaa, 
jotta toinen operaatio voi ottaa handlen käyttöön.
VixVM_Open()-funktio kääntää virtuaalikoneen hakemiston handlelle, jotta sitä voi käyttää myöhemmmissä funktiokutsuissa. VixVM_Open() on myös asynkroninen funktio.

```c
jobHandle = VixVM_Open(hostHandle,
                       vmxFilePath, // Virtuaalikoneen tiedosto
                       NULL, // VixEventProc *callbackProc
                       NULL); // void *clientData
```

5.3.5 Virtuaalikoneiden käynnistus / sammutus

VixVM_PowerOn()-funktiolla on kaksi käyttötarkoitusta:

- Käynnistää virtuaalikoneen käyttöjärjestelmä, joka on sammutetussa tilassa.

- Jatkaa käyttöjärjestelmän suoritusta, joka on suspended tilassa.

```c
jobHandle = VixVM_PowerOn(vmHandle,
                          VIX_VMPOWEROP_NORMAL,
                          VIX_INVALID_HANDLE,
                          NULL, // *callbackProc
                          NULL); // *clientData
```


```c
jobHandle = VixVM_PowerOff(vmHandle,
                           VIX_VMPOWEROP_NORMAL,
                           NULL, // *callbackProc
                           NULL); // *clientData
```


```c
jobHandle = VixVM_Suspend(vmHandle,
                         VIX_VMPOWEROP_NORMAL,
                         NULL, // *callbackProc
                         NULL); // *clientData
```

VixVM_Reset()-funktio vastaa samaa kuin reset-napin painallus. Jos virtuaalikone ei ole käynnissä, palautetaan virhe.

```c
jobHandle = VixVM_Reset(vmHandle,
                        VIX_VMPOWEROP_NORMAL,
                        NULL, // *callbackProc
                        NULL); // *clientData
```

```c
VixError err;
err = VixJob_Wait(jobHandle, // työhandle
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle, // palvelinhandle
    VIX_PROPERTY_NONE);
```

Jos virhe tapahtuu

```c
if (VIX_OK != err)
{
    // Jos virhe tapahtuu
}
```

Vix_GetErrorText()-funktio voidaan Vix-virheen sisältö palauttaa merkkijonona, jolloin se saadaan helposti ymmärtettävään muotoon.

```c
const char* errText;
errText = Vix_GetErrorText(err, NULL);
printf(errText);
printf("\n");
```

5.4 Vix API -ohjelma

VMwaren ohjelmointirajapintaan lähdettiin tutustumaan tekemällä ohjelma Vix API –rajapinnalle. Vix API –rajapinnalle löytyy monipuolisin dokumentaatio, sillä se on ylläpitäjien keskuudessa suositui VMwaren ohjelmointirajapinnoista monipuolisuutensa takia.
Kuva 23. Ohjeman käynnistysnäkymä

Kuva 24. Käyttäjälle näkyvät valinnat

Käynnistäessä ohjelma annetaan parametreina käyttäjätunus, salasana, IP-osoite ja virtuaalikoneen hakemisto.

VMApp.exe Tomi tomi localhost c:\VMs\Ubuntu\Ubuntu.wmx

Ohjelma ottaa automaattisesti yhteyden VMware Serveriin. Käyttäjä voi hallita parametreina annettua virtuaalikonetta. Ohjelma näyttää käyttäjälle valikkona, mitä funktioita käyttäjä voi suorittaa.

Ennen kunkin funktion suoritusta ohjelma tarkastaa että virtuaalikone on oikeassa tilassa (PowerState). Allaoleva ehtolause suoritetaan ennen virtuaalikoneen käynnistystä.

```c
// 32 == OFF, 2 == SUSPENDED
if (powerState == 32 || powerState == 2)
```
// Koodia
} else {
    cout << "Virtual machine not off or suspended" << endl;
}

Jokaisen suoritettavan metodin lopussa tarkastetaan ja päivitetään virtuaalikoneen PowerState allaolevalla funtiolla.

// Tarkasta PowerState muuttuja
err = Vix_GetProperties(vmHandle, // virtuaalikoneen handle
                       VIX_PROPERTY_VM_POWER_STATE,
                       &powerState, // PowerState muuttuja
                       VIX_PROPERTY_NONE);

Virtuaalikoneen sammutus-, nukutus- ja resetointifunktioiden jälkeen jäädään odottamaan / tarkistetaan, että virtuaalikone on varmasti sammunut / nukahtanut.

// Varmistetaan että virtuaalikone on sammunut
powerState = 0;
while (VIX_POWERSTATE_POWERED_OFF != powerState) {
    // Tarkasta PowerState muuttuja
}

Kuvassa 24 komentoliittymässä näkyvät käyttäjän valikon vaihtoehdot.

1. Yhdistää virtuaalikoneeseen.
   void open()

2. Vapauttaa yhteyden virtuaalikoneesta.
   void releaseHandle()

3. Yhdistää palvelimeen.
   void connect(char *ip, char *userN, char *password)

   void disconnect()

5. Käynnistää virtuaalikoneen.
   void powerOn()

   void powerOff()

7. Nukuttaa virtuaalikoneen.
void suspend()

8. Resetoi virtuaalikoneen.

void reset()


void checkState()

10. Lisää virtuaalikoneen palvelimen hakemistoon.

void regVM()

11. Poistaa virtuaalikoneen palvelimen hakemistosta.

void unregVM()

12. Tuhoaa virtuaalikoneen.

void deleteVM()

13. Listaa käynnissäolevat virtuaalikoneet.

void listRunningVM()


void alusta(char* vmPath)

Funktioiden suorittavat toiminnot ovat hyvin samanlaiset kuin aiemmin esitetyt Programming API:n skriptit. Funktioiden koodi löytyvät liitteenä olevalta cd:ltä.

5.5 VmCOM API -skriptit

C#-ohjelmat käyttävät virtuaalikoneiden etähallintaan VmCOM API:a.

5.5.1 API:n oliot

VmCOM API:n tärkeimmät oliot ovat:

- VmConnectParams

  Kokonaisuus yhteysparametreille, joka helpottaa Connect()-metodin käyttöä. Sisältää voidaan tallettaa ip, portti, käyttäjätunnus ja salasana.

- VmServerCtl
Kontrolli VMware Serveriin. Sisältää kokelman Serverille rekisteröidystä virtuaalikoineista, sekä kolme metodia Connect(), RegisterVm() ja UnregisterVm().

- VmCtl

Kontrolli yksittäiseen virtuaalikoneeseen. Kontrollista saa virtuaalikoneen tärkeimmät tiedot, sekä se sisältää tärkeimmät hallintametodit, kuten Connect(), Start(), Stop(), Suspend() ja Reset().

5.5.2 Yhteydenotto palvelimeen

Palvelimeen voi ottaa yhteyden kolmella tavalla:

- paikalliseen VMware Serveriin tämänhetkisellä käyttäjällä, jolloin kaikiksi arvoiksi määritetään NULL.

- paikalliseen VMware Serveriin määritetyllä käyttäjällä, jolloin username ja password määritetään, mutta hostname jätetään arvoksi NULL.

- määritettyyn palvelimeen, määritetyllä käyttäjällä, jolloin username, password, sekä hostname määritetään.

Porttina on vakiona 902. Mikäli kyseinen portti ei toimi, on VMware Server asentunut johonkin toiseen porttiin, koska portti 902 on ollut asennushetkellä jonkin toisen ohjelman käytössä.

```java
VmConnectParams vmConn = new VmConnectParams();
VmServerCtl vmServC = new VmServerCtl();

vmConn.Hostname = "212.10.6.1";
vmConn.Username = "Käyttäjä";
vmConn.Password = "Salasana";

try {
    vmServC.Connect(vmConn);
} catch {} 
```

Mikäli yhteys epäonnistuu, palautetaan virhe.
5.5.3 Virtuaalikoneen rekisteröinti

VMware Server vaatii, että virtuaalikoneet löytyvät sen omasta virtuaalikonehakemistosta ennenkuin niitä voi käyttää. Rekisteröinti vaatii virtuaalikoneen konfiguraationimen.

```
try {
    vmServC.RegisterVm("name");
} catch {} // register virtual machine
```

Jos virtuaalikoneen poistaa hakemistosta, sillä ei voi suorittaa mitään operaatioita. Virtuaalikoneen poisto hakemistosta on hyvä tapa estää muutokset virtuaalikoneeseen, mikäli virtuaalikone ei ole usein käytössä.

```
try {
    vmServC.UnRegisterVm("name");
} catch {} // unregister virtual machine
```

Mikäli toimenpide epäonnistuu, palautetaan virhe.

5.5.4 Yhteydenotto virtuaalikoneeseen

Yhteydenotto vaatii, että virtuaalikone on rekisteröity VMware Serverin hakemistoon. Connect()-metodille annetaan yhteysparametrit sekä virtuaalikoneen konfiguraationimi, eli virtuaalikoneen vmx-tiedoston täydellisen nimen.

```
try {
    vm.Connect(vmConn, "name"); // connect to virtual machine
} catch {} // connect to virtual machine
```

Mikäli yhteydenluonti epäonnistuu, palautetaan virhe.

5.5.5 Virtuaalikoneiden käynnistus / sammutus

Virtuaalikoneen tilanvaihteluun on seuraavat metodit; Start(), Stop(), Suspend() ja Reset(). Metodeille välitetään ns. voimaoptio (VmPowerOpMode) parametriina. Se määrittelee, kuinka kovasti VMware Server koittaa ajaa tahdotun metodin läpi. VmPowerOpModilla on kolme eri tilaa:

- Soft

```
VmPowerOpMode.vmPowerOpMode_Soft
```
- **Hard**
  \[\text{VmPowerOpMode.vmPowerOpMode\_Hard}\]

- **TrySoft**
  \[\text{VmPowerOpMode.vmPowerOpMode\_TrySoft}\]

TrySoft modessa koitetaan ensin Softia, mutta mikäli se epäonnistuu vaihdetaan mode Hardiksi.

**Start()-metodilla on kaksi käyttötarkoitusta:**

- Käynnistää virtuaalikoneen käyttöjärjestelmä, joka on sammutetussa tilassa.

- Jatkaa virtuaalikoneen suoritusta, joka on suspended-tilassa.

```java
try
{
    vm.Start(VmPowerOpMode);
}
catch{}
```

**Stop()-metodin** sammuttaa virtuaalikoneen. Metodi vastaa samaa, kuin tietokone sammutettaisiin virtakytkimestä. Jotkut virtuaalikoneen operaatiot, kuten virtuaalisen raudan konfiguroiminen, vaativat, että virtuaalikone on pois päältä.

```java
try
{
    vm.Stop(VmPowerOpMode);
}
catch{}
```

**Suspend()-metodi** vastaa samaa kuin kannettavien nukkumistila. Käyttöjärjestelmä lopettaa kaiken suorittamisen, mutta Start()-metodi palauttaa käyttöjärjestelmän samaan tilaan, missä se oli ennen Suspend()-metodin kutsua. Jos virtuaalikone ei ole käynnissä, palautetaan virhe.

```java
try
{
    vm.Suspend(VmPowerOpMode);
}
catch{}
```

**Reset()-metodi** vastaa samaa, kuin reset-napin painallus. Jos virtuaalikone ei ole käynnissä, palautetaan virhe.

```java
try
```
Mikäli toimenpide epäonnistuu, palautetaan virhe.

5.6 VmCOM API stand-alone ohjelma

VmCOM API:in tutustuminen aloitettiin tekemällä Windows Forms -pohjainen sovellus, jossa kaikki yhteydenhallinta käsitellään ohjelman sisällä.

Ohjelmassa on pääluokka (MainForm) sekä VmManager-luokka. Käyttäjä antaa IP-osoitteen sekä käyttäjätunnuksen ja salasanan tietokoneelle, jossa VMware Server sijaitsee. Ohjelma lukkee käytössä olevat virtuaalikoneet tekstitiedostosta joka sijaitsee käyttäjän omalla koneella. Tekstitiedostoon annetaan jokaisen virtuaalikoneen vmx-tiedoston täydellinen nimi omalle rivilleen. Täydellinen nimi sisältää myös hakemiston, kuten alla esimerkissä.

C:\VMs\FreeBSD 64-bit\FreeBSD 64-bit.vmx
C:\VMs\Ubuntu 64-bit\Ubuntu 64-bit.vmx

5.6.1 VmManager-luokka

```
weit.t();

Mikäli toimenpide epäonnistuu, palautetaan virhe.

5.6 VmCOM API stand-alone ohjelma

VmCOM API:in tutustuminen aloitettiin tekemällä Windows Forms -pohjainen sovellus, jossa kaikki yhteydenhallinta käsitellään ohjelman sisällä.

Ohjelmassa on pääluokka (MainForm) sekä VmManager-luokka. Käyttäjä antaa IP-osoitteen sekä käyttäjätunnuksen ja salasanan tietokoneelle, jossa VMware Server sijaitsee. Ohjelma lukkee käytössä olevat virtuaalikoneet tekstitiedostosta joka sijaitsee käyttäjän omalla koneella. Tekstitiedostoon annetaan jokaisen virtuaalikoneen vmx-tiedoston täydellinen nimi omalle rivilleen. Täydellinen nimi sisältää myös hakemiston, kuten alla esimerkissä.

C:\VMs\FreeBSD 64-bit\FreeBSD 64-bit.vmx
C:\VMs\Ubuntu 64-bit\Ubuntu 64-bit.vmx

5.6.1 VmManager-luokka

```
weit.t();

Mikäli toimenpide epäonnistuu, palautetaan virhe.
ConnectToVM(), ControlVM(), Register() ja Unregister(). Metodien suoriutuessa loppuun ne kutsuvat pääikkunan delegaatteja.

```java
public VmManager(MainForm main, VmConnectParams vmConn, VmServerCtl vmServC)
{
    this.main = main;
    this.vmConn = vmConn;
    this.vmServC = vmServC;
}
```

ConnectToServer()-metodi ottaa yhteyden palvelinkontrollilla käyttäen annettuja yhteysparametrejä.

```java
public void ConnectToServer()
{
    connected = true;

    try
    {
        vmServC.Connect(vmConn);
    }
    catch
    {
        connected = false;
    }

    main.Invoke(main.connThreadFinished, null);
}
```

ConnectToVM()-metodi ottaa yhteyden virtuaalikoneen kontrolliolioon (VmCtl) annettujen yhteysparametrien ja virtuaalikoneen nimen (string) avulla.

```java
public void ConnectToVM(VmCtl vm, String s)
{
    success = true;
    this.vm = vm;

    try
    {
        vm.Connect(vmConn, s);
    }
    catch
    {
        success = false;
    }

    main.Invoke(main.vmConThreadFinished, null);
}
```

Reigister()-metodi rekisteröi virtuaalikoneen nimen (string) avulla, VMware Serverin virtuaalikonehakemistoon, palvelinkontrollin (VmServCtl) avulla, jolloin sitä voidaan hallita.
internal void RegisterVM(string name, VmServerCtl vmServC)
{
    success = true;

    try
    {
        vmServC.RegisterVm(name);
    }
    catch
    {
        success = false;
    }

    main.Invoke(main.vmThreadFinished, null);
}

Unregister()-metodi poistaa virtuaalikoneen hakemistosta.

internal void UnregisterVM(string name, VmServerCtl vmServC)
{
    success = true;

    try
    {
        vmServC.UnregisterVm(name);
    }
    catch
    {
        success = false;
    }

    main.Invoke(main.vmThreadFinished, null);
}

ControVM()-metodilla voidaan käynnistää, sammuttaa, nukuttaa tai palauttaa virtuaalikone. Metodi valitsee oikean voimamoden ja suoritettavan toiminnon annettujen stringien avulla.

internal void ControlVM(string type, VmCtl vm, string powerop)
{
    this.vm = vm;
    VmPowerOpMode mode;
    switch (powerop)
    {
        case "Soft":
            mode = VmPowerOpMode.vmPowerOpMode_Soft;
            break;
        case "TrySoft":
            mode = VmPowerOpMode.vmPowerOpMode_TrySoft;
            break;
        default:
            mode = VmPowerOpMode.vmPowerOpMode_Hard;
            break;
    }
    success = true;

    try
    {
        switch (type)
{"case ("Start"):
    vm.Start(mode);
    break;
    case ("Stop"):
    vm.Stop(mode);
    break;
    case ("Reset"):
    vm.Reset(mode);
    break;
    case ("Suspend"):
    vm.Suspend(mode);
    break;
    case ("Resume"):
    vm.Start(mode);
    break;
    default:
    break;
}"

} catch (Exception ex) {
    success = false;
} }

main.Invoke(main.vmThreadFinished, null); }
5.6.2 MainForm-luokka

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+connThreadFinished : DelegateThreadFinished</td>
<td></td>
</tr>
<tr>
<td>+vmConThreadFinished : DelegateThreadFinished</td>
<td></td>
</tr>
<tr>
<td>+vmThreadFinished : DelegateThreadFinished</td>
<td></td>
</tr>
<tr>
<td>+vmControlThreadFinished : DelegateThreadFinished</td>
<td></td>
</tr>
<tr>
<td>-vmConn : VmConnectParams</td>
<td></td>
</tr>
<tr>
<td>-vmServC : VmServerCtl</td>
<td></td>
</tr>
<tr>
<td>-vm : VmCtl</td>
<td></td>
</tr>
<tr>
<td>-worker : Thread</td>
<td></td>
</tr>
<tr>
<td>-manager : VmManager</td>
<td></td>
</tr>
<tr>
<td>-VMs : ArrayList</td>
<td></td>
</tr>
<tr>
<td>-conVMs : ArrayList</td>
<td></td>
</tr>
<tr>
<td>-name : string</td>
<td></td>
</tr>
<tr>
<td>-type : string</td>
<td></td>
</tr>
<tr>
<td>-powerop : string</td>
<td></td>
</tr>
<tr>
<td>-line : string</td>
<td></td>
</tr>
<tr>
<td>-file : FileStream</td>
<td></td>
</tr>
<tr>
<td>-sr : StreamReader</td>
<td></td>
</tr>
<tr>
<td>-form : LoadingForm = new LoadingForm()</td>
<td></td>
</tr>
</tbody>
</table>

Kuva 26. MainForm:in luokkakaavio

Ohjelman suoritus on samanlainen halutusta toimenpiteestä riippumatta. Esimerkiksi rekisteröitäessä virtuaalikone toimitaan seuraavasti:

1. Luodaan uusi säie:

   ```csharp
   private void regBtn_Click(object sender, EventArgs e)
   {
       // koodia
       worker = new Thread(new ThreadStart(this.ControlVM));
       worker.Start();
   }
   ```

2. Säieen sisällä kutustaan VmManager luokkaa:

   ```csharp
   public void ControlVM()
   {
   ```
3. VmManagerin metodi kutsuu MainFormin delegaattia:

```csharp
internal void ControlVM(string type, VmCtl vm, string pop)
{
    // koodia
    main.Invoke(main.vmThreadFinished, null);
}
```

4. Suoritetaan metodi johon delegaatti ohjaa:

```csharp
vmThreadFinished = new DelegateThreadFinished(this.VmThreadFinished);
private void VmThreadFinished()
{
    // koodia
    if (manager.Success)
    {
        updateVMs();
    }
    else
    {
        MessageBox.Show("Failed");
    }
}
```

Kuva 27. Käynnistysnäkymä


```csharp
worker = new Thread(new ThreadStart(this.ConnectToServer));
worker.Start();
```
ConnectToServer()-metodi luo uuden VmManagerin ja kutsuu managerin ConnectToServer()-metodia.

```java
manager = new VmManager(this, vmConn, vmServC);
manager.ConnectToServer();
```

Kun manager on saanut suoritettua metodin loppuun, se kutsuu pääikkunaa delegaatilla, joka ohjautuu ConnThreadFinished()-metodiin. Metodi tarkistaa managerin connected booleanista onko metodi onnistunut yhteydenottossa ja mikäli oli, päivittää virtuaalikoneiden listan (kuva 28).

![Kuva 28. Yhdistettynä VMware Serveriin](image)

Kun käyttäjällä on virtuaalikone valittuna ja hän painaa Connect-nappia, luodaan uusi säie.

```csharp
worker = new Thread(new ThreadStart(this.ConnectToVM));
worker.Start();

ConnectToVM()-metodi kutsuu managerin ConnectToVM() metodia,

manager.ConnectToVM(vm, name);

joka valmisutessaan kutsuu delegaattia, joka ohjautuu VmConThreadFinished()-metodiin. Mikäli VmManager onnistui yhteydenluonnissa, lisätään virtuaalikoneen kontrolli (VmCtl) kokoelmaan ja päivitetään virtuaalikonelista.
Kuva 30. Valittu virtuaalikone käynnistettynä

Yhdistetystä virtuaalikoneesta (kuva 30) voi hakea lisätietoa painamalla Infonappia. Ohjelma etsii oikean virtuaalikoneen kontrollin (VmCtl) kokoelmasta ja tulostaa tietoja Info-välilehteen (kuva 31).

Kuva 31. Valitun virtuaalikoneen lisätieto
5.7 TCP-palvelin

Palvelin tarjoaa yhteydanhallintapalveluita asiakasohjelmille. Palvelin ei välitä asiakasohjelman tyyppistä, vaan toimii aina samalla tavalla asiakkaasta riippumatta. Palvelinta käyttää VmCOM API Client -ohjelma (katso luku 5.8) sekä dynaamiset ASP.NET-sivut (katso luku 5.9).

Palvelin sisältää Server-, Handler-, JobManager- ja VmManager-luokat. Server-luokassa on palvelimen toiminnallisuus, ja se pitää tallessa käyttäjän kokemat. Jokaiselle käyttäjälle luodaan oma Handler-luokka, joka hoitaa käyttäjän pyynnöt ja pitää huolta käyttäjän omista olikokoelmista. Jokaisella käyttäjällä on myös omat JobManager- ja VmManager-luokat, joissa tapahtuu virtuaalikoneiden hallinta. Palvelimen toiminta on sääkeistetty, jolloin se pystyy käsitlemään monien käyttäjien toimintoja "samanaikaisesti".


```xml
<?xml version="1.0" encoding="utf-8" ?>
```
Ylläolevan Settings-tiedoston Trusted Connection String

Server=IP547;Database=VMware;Trusted=true;

Standard Connection String

Server=ttest;Database=CPMDEMO8;Uid=sa;Password=admin;

Jokainen käyttäjän pyytämä toiminto palauttaa käyttäjälle kaksi stringiä, viestin, joka sisältää suoritetun pyynnön kuvausen sekä virtuaalikonelistauksen tai virtuaalikoneen infon XML-muodossa.

Esimerkki virtuaalikonelistasta.

```xml
<?xml version="1.0"?>
<List>
  <VM>
    <Name>
      C:\VMs\FreeBSD 64-bit\FreeBSD 64-bit.vmx
    </Name>
    <State>
      Stopped
    </State>
  </VM>
  <VM>
    <Name>
      c:\VMs\Ubuntu 64-bit\Ubuntu 64-bit.vmx
    </Name>
    <State>
      NotConnected
    </State>
  </VM>
</List>
```

Esimerkki virtuaalikoneen infosta.

```xml
<?xml version="1.0"?>
.getInfo>
  <Name>
```
Sekvenssikaaviossa on esitetty luokkien välinen kommunikaatio client-ohjelman lähettäessä pyyntö palvelimelle.

Kuva 33. Sekvenssikaavio

Palvelin kirjoittaa toimissaan toimenpiteitään konsolin ikkunaan, jolloin sen toimintaa voi seurata ilman debuggeria. Palvelimella on käytössä myös pieni Logger–apuohjelma (katso luku 5.10), joka kirjoittaa tekstitiedostoon palvelimen tapahtumia, sekä virheilmoituksia.
Kuva 34. Toimintojen kirjautuminen konsolin ikkunaan

Samaan aikaan lokkiin kirjautuu seuraavat tiedot.

<Message Type='Action' Time='3.10.2007 16:56:09' 
User='System'>Filled user lists</Message>
<Message Type='Action' Time='3.10.2007 16:56:09' 
User='System'>Server started</Message>
<Message Type='Action' Time='3.10.2007 16:56:55' 
User='tomi'>Logged in</Message>
<Message Type='Action' Time='3.10.2007 16:56:55' 
User='tomi'>Created handler</Message>
<Message Type='Action' Time='3.10.2007 16:56:56' 
User='tomi'>ConnectToServer</Message>
<Message Type='Action' Time='3.10.2007 16:57:00' 
User='tomi'>Connection Success</Message>
<Message Type='Action' Time='3.10.2007 16:57:24' 
User='tomi'>Logged in</Message>
<Message Type='Action' Time='3.10.2007 16:57:24' 
User='tomi'>Found Handler</Message>
<Message Type='Action' Time='3.10.2007 16:57:25' 
User='tomi'>ConnectToVM</Message>
<Message Type='Action' Time='3.10.2007 16:57:41' 
User='tomi'>Connection Success</Message>
<Message Type='Action' Time='3.10.2007 16:57:41' 
User='tomi'>Added VM: D:\VMs\FreeBSD\FreeBSD.vmx</Message>
5.7.1 Server-luokka

<table>
<thead>
<tr>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>-listener : TcpListener</td>
</tr>
<tr>
<td>-port : int = 5050</td>
</tr>
<tr>
<td>-UserHandler : Dictionary&lt;string, TcpServer.Handler&gt;</td>
</tr>
<tr>
<td>-runningThreads : ArrayList = new ArrayList()</td>
</tr>
<tr>
<td>+SqlConnection : SqlConnection</td>
</tr>
<tr>
<td>+SqlCommand : SqlCommand</td>
</tr>
<tr>
<td>+SqlReader : SqlDataReader</td>
</tr>
<tr>
<td>-userNama : string</td>
</tr>
<tr>
<td>-password : string</td>
</tr>
<tr>
<td>-conString : string</td>
</tr>
<tr>
<td>-vmConn :VmConnectParams</td>
</tr>
<tr>
<td>-VMs : ArrayList</td>
</tr>
<tr>
<td>-log : Log = Logging.Log.GetInstance()</td>
</tr>
</tbody>
</table>

private void getSettings()
{
  XmlTextReader textReader = new XmlTextReader("Settings.xml");
  XmlDocument doc = new XmlDocument();
  doc.Load(textReader);

  XPathNavigator nav = doc.CreateNavigator();

  // Get port
  XPathNodeIterator node = nav.Select("settings/port");
  while (node.MoveNext())
  {
    port = System.Convert.ToInt32(node.Current.Value);
  }

  // Get connection string
  node = nav.Select("settings/connectionstring");
  while (node.MoveNext())
  {
    switch (node.Current.GetAttribute("type", null))
    {
      case ("trusted"):
        conString += "Database=" + node.Current.SelectSingleNode("database").Value + ";";
        conString += "Trusted_Connection=" + node.Current.SelectSingleNode("trusted").Value + ";";
        break;
      case ("standard":
    
Kuva 35. Luokkakaavio

Käynnistyessään ohjelma lukee asetuksia Settings.xml tiedostosta.

private void getSettings()
{
  XmlTextReader textReader = new XmlTextReader("Settings.xml");
  XmlDocument doc = new XmlDocument();
  doc.Load(textReader);

  XPathNavigator nav = doc.CreateNavigator();

  // Get port
  XPathNodeIterator node = nav.Select("settings/port");
  while (node.MoveNext())
  {
    port = System.Convert.ToInt32(node.Current.Value);
  }

  // Get connection string
  node = nav.Select("settings/connectionstring");
  while (node.MoveNext())
  {
    switch (node.Current.GetAttribute("type", null))
    {
      case ("trusted"):
        conString += "Database=" + node.Current.SelectSingleNode("database").Value + ";";
        conString += "Trusted_Connection=" + node.Current.SelectSingleNode("trusted").Value + ";";
        break;
      case ("standard":
    
Kuva 35. Luokkakaavio

Käynnistyessään ohjelma lukee asetuksia Settings.xml tiedostosta.
Server-luokka luo kokoelman käyttäjille (kuva 36).

Dictionary<string, ArrayList> UserList;


**Kuva 36. Käyttäjäkokoelman rakenne**

1. Luetaan tietokannasta kaikki käyttäjät ja talletetaan käyttäjätunnus ja salasana.

```csharp
string cmd = "Select * FROM Users";
SqlCommand = new SqlCommand(cmd, SqlConn);
SqlReader = SqlCommand.ExecuteReader();
while (SqlReader.Read()) {
    string name = SqlReader["UserName"].ToString();
    string password = SqlReader["Password"].ToString();
    UserList.Add(name, new ArrayList());
    UserList[name].Add(password);
}
```
2. Käydään foreach-silmukalla läpi jokainen käyttäjä ja päivitetään jokaiselle käyttäjälle VMs-kokoelma.

```csharp
foreach (string s in UserList.Keys)
{
    SqlCommand.CommandText = "Select * FROM VirtualMachines, Users WHERE VirtualMachines.Owner = Users.ID AND Users.UserName = '" + s + "'";
    SqlDataReader = SqlCommand.ExecuteReader();
    VMs = new ArrayList();
    while (SqlReader.Read())
    {
        string name = SqlReader["Folder"].ToString();
        VMs.Add(name);
    }
    UserList[s].Add(VMs);
    SqlDataReader.Close();
}
```


```csharp
foreach (string s in UserList.Keys)
{
    vmConn = new VmConnectParams();
    SqlDataReader = SqlCommand.ExecuteReader();
    while (SqlReader.Read())
    {
        vmConn.Hostname = SqlReader["IP"].ToString();
        vmConn.Username = SqlReader["UserName"].ToString();
        vmConn.Password = SqlReader["Password"].ToString();
    }
    UserList[s].Add(vmConn);
    UserList[s].Add(new VmServerCtl());
    SqlDataReader.Close();
}
```

Virtuaalikoinet lisätään käyttäjän kokoelmaan sitämukaan kun niihin otetaan yhteyksiä käyttäjän toimesta. Myös käyttäjän Handler talletetaan omaan kokoelmaan, jossa avaimena on käyttäjätunnus ja tietona käyttäjän oma Handler-luokka.

```csharp
Dictionary<string, Handler> UserHandler;
```

Kun käyttäjäkokoelma on täytetty, ohjelma luo uuden TCP-kuuntelijan ja asettuu kuuntelemaan määritettyä porttia.
TcpListener listener;

listener = new TcpListener(port);
listener.Start();

Kuuntelijassa ohjelma asetuu kuuntelemaan sockettia ja luo tietoviralle lukijan ja kirjoittajan.

Socket socket = listener.AcceptSocket();

NetworkStream stream = new NetworkStream(socket);
StreamReader reader = new StreamReader(stream);
StreamWriter writer = new StreamWriter(stream);


Thread thread = new Thread(new ThreadStart(handler.Start));
thread.Name = userName;
runningThreads.Add(thread);
thread.Start();

Säikeet tallitetaan myös runningThreads nimiseen kokoelmaan ja nimetään käyttäjän mukaan. Tällä hetkellä tästä ei vielä varsinaista hyötyä ole, mutta mikäli säikeiden toimintaa halutaan enemmän hallita voidaan tätä kokoelmaa käyttää myöhemmin apuna. Kun Handler-luokka on valmis, se kutsuu Serverin delegaattia ThreadFinished, jossa se poistetaan kokoelmasta.
5.7.2 Handler-luokka

<table>
<thead>
<tr>
<th>Class</th>
<th>Public methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handler</td>
<td>public Handler(Socket socket, ArrayList list)</td>
</tr>
</tbody>
</table>

Kuva 37. Luokkakaavio

Handler-luokan pääasiallinen tehtävä on kuunnella käyttäjän komento ja ohjastaa JobManager tekemään oikeaa tehtävää. Handler-luokalle annetaan konstruktorissa socket ja käyttäjän kokoelma. Kokelmasta etsitään yhteyssparametrit (VmConnectParams) ja palvelinkontrolli (VmServerCtl), jotka välitetään JobManagerille sen konstruktorissa.

```java
public Handler(Socket socket, ArrayList list) {
    this.socket = socket;
    this.list = list;

    vmConn = FindVmConnParams();
    vmServc = FindVmServCtl();

    manager = new JobManager(this, vmConn, vmServc);
    manager.finished += new JobManager.ThreadFinished(manger_finished);
}
```
stream = new NetworkStream(socket);
reader = new StreamReader(stream);
writer = new StreamWriter(stream);
}

Tietovirrat päivitetään aina, kun käyttäjä ottaa uudestaan yhteyden palvelimeen.

internal void UpdateSocket(Socket socket)
{
    stream = new NetworkStream(socket);
    reader = new StreamReader(stream);
    writer = new StreamWriter(stream);
}

Handler-luokan ollessa käynnissä se kuuntelee käyttäjältä toiminnon, päivittää managerin tiedot ja käynnistää managerin säikeessä.

action = reader.ReadLine();
Console.WriteLine("Task: "+action);
switch (action)
{
    case "ConnectToServer":
        // Koodia
        break;
    case "ConnectToVM":
        vmName = reader.ReadLine();
        manager.CurrentVmName = vmName;
        thread = new Thread(new ThreadStart(manager.ConnectToVM));
        thread.Name = action;
        runningThreads.Add(thread);
        thread.Start();
        break;
        // Koodia
}

Kun JobManager on valmis, se kutsuu Threadfinished-delegaattia. JobManager kapseloi tiedot ConnectionEventArgs-luokan sisään.

Kuva 38. ConnectionEventArgs-luokkaavio

```csharp
private void manager_finished(object sender, ConnectionEventArgs ev)
{
    Console.WriteLine(ev.Message);

    // If arguments have new vm control add it to list
    if (ev.Vm != null)
        list.Add(ev.Vm);

    // Remove Thread from runningthreads
    foreach (Thread th in runningThreads)
    {
        if (th.Name == ev.Name)
        {
            runningThreads.Remove(th);
            break;
        }
    }

    updateLists();
    createXML();

    sendMessage(ev.Message, xmlDoc.OuterXml);
    // set active state to false
    active = false;
}
```
5.7.3 JobManager-luokka

<table>
<thead>
<tr>
<th>JobManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>+active : bool</td>
</tr>
<tr>
<td>-vmConn : VmConnectParams</td>
</tr>
<tr>
<td>-vmServC : VmServerCtl</td>
</tr>
<tr>
<td>-vm : VmCtl</td>
</tr>
<tr>
<td>-currentVmName : string</td>
</tr>
<tr>
<td>-powerOp : string</td>
</tr>
<tr>
<td>-type : string</td>
</tr>
<tr>
<td>-manager : VmManager</td>
</tr>
<tr>
<td>-master : Handler</td>
</tr>
<tr>
<td>+finished() : ThreadFinished</td>
</tr>
<tr>
<td>+JobManager( master : Handler, in vmConn : VmConnectParams, in vmServC : VmServerCtl)</td>
</tr>
<tr>
<td>+ConnectToServer()</td>
</tr>
<tr>
<td>+ConnectToVM()</td>
</tr>
<tr>
<td>+ControlVM()</td>
</tr>
<tr>
<td>+RegisterVM()</td>
</tr>
<tr>
<td>+UnRegisterVM()</td>
</tr>
<tr>
<td>+CurrentVmName() : string</td>
</tr>
<tr>
<td>+PowerOp() : string</td>
</tr>
<tr>
<td>+Type() : string</td>
</tr>
<tr>
<td>+VmConn() : VmConnectParams</td>
</tr>
<tr>
<td>+VmServC() : VmServerCtl</td>
</tr>
<tr>
<td>+vm() : VmCtl</td>
</tr>
</tbody>
</table>

Kuva 39. Luokkakaavio

JobManager-luokka sisältää Handlerin käyttämät yhteydenhallintametodit. Konstruktorissa luokalle annetaan käyttäjän yhteydenhallintaoliot. JobManager välittää nämä oliot VmManager-luokalle sen konstruktorissa.

```java
public JobManager(Handler master, VmConnectParams vmConn, VmServerCtl vmServC)
{
    this.master = master;
    this.vmConn = vmConn;
    this.vmServC = vmServC;

    // Create new manager
    manager = new VmManager(this, vmConn, vmServC);
}
```

JobManagerin yhteydenhallintametodit luovat yhteydenhallinta-argumentin, jossa välitetään booleanina onnistuminen, yhteydenhallintaviesti sekä mahdollisesti myös uuden virtuaalikoneen kontrolliolio, mikäli kyseessä on uusi yhteys virtuaalikoneeseen.

```java
public void ConnectToVM()
{
    // Create new control for Vm
    vm = new VmCtl();

    // Connect
    manager.ConnectToVM(vm, currentVmName);

    // Update masters properties
    master.UpdateProperties();
}
```
// Create new event arguments
ConnectionEventArgs ev = new ConnectionEventArgs();
ev.Name = "ConnectToVM";
ev.Message = manager.Message;
ev.Success = manager.Success;

// Add vm to event args only if connection was success
if (manager.Success)
    ev.Vm = vm;

if (finished != null)
    finished(this, ev);
}

Metodikutsuja JobManager ja VmManager-luokan välillä ei ole säikeistetty, sillä JobManager ei voi tehdä monia pyytöjä samanaikaisesti VMware Serverille.

5.7.4 VmManager-luokka

<table>
<thead>
<tr>
<th>VmManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmConn : VmConnectParams</td>
</tr>
<tr>
<td>vmServC : VmServerCtl</td>
</tr>
<tr>
<td>vm : VmCtl</td>
</tr>
<tr>
<td>master : JobManager</td>
</tr>
<tr>
<td>connected : bool</td>
</tr>
<tr>
<td>success : bool</td>
</tr>
<tr>
<td>message : string</td>
</tr>
</tbody>
</table>
| +VmManager(in master : JobManager, in vmConn : VmConnectParams, in vmServC : VmServerCtl)
+VmManager(in vmConn : VmConnectParams, in vmServC : VmServerCtl)
+ConnectToServer() : bool |
+RegisterVM(in name : string, in vmServC : VmServerCtl) : bool |
+UnRegisterVM(in name : string, in vmServC : VmServerCtl) : bool |
+ControlVM(in type : string, in vm : VmCtl, in powerop : string) : bool |
+Message() : string |
+Connected() : bool |
+Success() : bool |

Kuva 40. Luokkakaavio


public bool ConnectToServer()
{
    connected = true;
    success = true;

    try
    {
        vmServC.Connect(vmConn);
    }
    catch
    { 
        // Handle exception
    }
}
Message = "Connection Success";

} // catch
{
    connected = false;
    success = false;
    Message = "Connection Failed";
}

return success;

5.8 VmCOM API palvelinpohjainen ohjelma

Client-ohjelmalla ei ole enää yhteyttä VMware Serveriin, vaan client lähetää pyynnöt TCP-palvelimelle, joka on yhteydessä VMware Serveriin. Client-ohjelmalla ei ole enää hallussa kontrolli- ja virtuaalikoneoloiota. Se toimii vain näyttöön, joka näyttää käyttäjälle tämänhetkisen tilanteen.

Käyttäjä antaa vain oman käyttäjätunnuksen ja salasanan. Ohjelma lukee Settings.xml-tiedostosta tarvittavat yhteysparametrit, TCP-palvelimen IP-osoitteen, sekä portin. XML-tiedosto haetaan samasta kansiosta, missä sijaitsee ohjelman käynnistystiedosto.

```xml
<?xml version="1.0" encoding="utf-8" ?>

<settings>
  <ip>192.168.181.1</ip>
  <port>5050</port>
</settings>
```
TCP-palvelimella on tietokannassa käyttäjälle määrityn VMware Serverin IP-osoite, salasana sekä käyttäjän virtuaalikoneet, joten käyttäjän ei tarvitse itse huolehtia kyseisistä tiedoista.

Ulkoisesti käyttöliittymä (kuva 42) on stand-alone -ohjelman kanssa lähes identtinen. Vain virtuaalikonetiedoston haku puuttuu.

![Kuva 42. Clientin käyttöliittymä](image)

Ohjelma on toteutettu MVC (Model-View-Controller) -arkkitehtuuria mukailen, jolloin itse näytöllä (MainForm) ei ole mitään hallintakontrolleja, vaan näytön hallintaa hoitaa FormManager-luokka. Ohjelmalla ei ole tallessa varsinaista dataa, joten Model-komponentti puuttuu kokonaan. TCP-palvelimen yhteydenhallintaa hoitaa myös oma luokka, ThreadHandler. Ohjelmassa on säikeiden sijaan käytetty .NET Frameworkin Background Workeria, joka on tehty helpottamaan säikeidenhallintaa.
5.8.1 MainForm-luokka

Kuva 43. Luokkakavio

Pääikkunalla ei kontrollien ja tarvittavien kontrollien public-viitattavien lisäksi ole mitään muuta toiminnallisuutta. Alla esimerkkinä on viittaus Start-Button-kontrolliin.

```csharp
public Button StartBtn
{
    get { return startBtn; }
    set { startBtn = value; }
}
```
5.8.2 FormManager-luokka

Kuva 44. Luokkakaavio

FormManager on hallitsee kaikkia pääikkunan näkyviä контроллеja. FormManager myös luo uuden MainFormin sekä ottaa hallintaansa kaikki sen kontrollit.

```csharp
public FormManager()
{
    form = new MainForm();

    // Set handlers for controls
    userBox = form.UserBox;
    pwdBox = form.PwdBox;
}
```
startBtn = form.StartBtn;
startBtn.Click += new EventHandler(startBtn_Click);
// Koodia

bgWorker = new BackgroundWorker();
bgWorker.DoWork += new DoWorkEventHandler(bgWorker_DoWork);
bgWorker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(bgWorker_RunWorkerCompleted);

getSettings();
comboBox.SelectedIndex = 0;
form.Show();
}


private void getSettings()
{
    XmlTextReader textReader = new XmlTextReader("Settings.xml");
    XmlDocument doc = new XmlDocument();
    doc.Load(textReader);

    XPathNavigator nav = doc.CreateNavigator();

    // Get port
    XPathNodeIterator node = nav.Select("settings/port");
    while (node.MoveNext())
    { port = System.Convert.ToInt32(node.Current.Value); }

    // Get ip
    node = nav.Select("settings/ip");
    while (node.MoveNext())
    { ip = node.Current.Value; }
}


private void startBtn_Click(object sender, EventArgs e)
{
    tHandler = new ThreadHandler(this, userBox.Text, pwdBox.Text, ip, port);
tHandler.TaskReady += new ThreadHandler.TaskReadyHandler(tHandler_servReady);
tHandler.InfoReady += new ThreadHandler.InfoReadyHandler(tHandler_infReady);
tHandler.ListReady += new ThreadHandler.DelegateList(tHandler_listReady);
tHandler.ThreadReady += new ThreadHandler.DelegateThreadFinished(tHandler_thrReady);
tHandler.TaskFailed += new ThreadHandler.TaskFailedHandler(tHandler_TaskFailed);

startBtn.Enabled = false;

EnabledAll(false);

workToDo = "ConnectToServer";

bgWorker.RunWorkerAsync("ConnectToServer");
}

Kun Background Worker käynnistyy, se kutsuu omaa DoWork()-metodiaan ja käynnistää käskeyään vastaavan metodon ThreadHandleristä.

void bgWorker_DoWork(object sender, DoWorkEventArgs e)
{
    e.Result = e.Argument;

    switch ((string)e.Argument)
    {
    case "ConnectToServer":
        tHandler.ConnectToServer();
        break;
    case "ConnectToVM":
        tHandler.ConnectToVM();
        break;
    case "ControlVM":
        tHandler.ControlVM();
        break;
    case "Info":
        tHandler.Info();
        break;
    case "Register":
        tHandler.Register();
        break;
    case "UnRegister":
        tHandler.UnRegister();
        break;
    default:
        break;
    }
}

Sää nostaa TaskReady-eventin, kun ThreadHandler on saanut TCP-palvelimelta vastauksen, jolloin päivitetään TCP-palvelimelta saadut vastaus- ja XML-string.

void tHandler_servReady(string answer, string xml)
{
    this.answer = answer;
    this.xml = xml;
}
Mikäli käyttäjä oli pyytänyt palvelimeltä virtuaalikoneen lisäinfoa, säie nostaa InfoReady-eventin. ParseInfo()-metodi käy läpi saadun XML-stringin ja muokkaa sen käyttäjälle sopivaan esitysmuotoon.

```csharp
void tHandler_infReady(string info)
{
    this.info = ParseInfo(info);
}
```

Kun Background Worker on saanut tehtävän suoritettua loppuun, se kutsuu omaa RunWorkerCompleted()-metodiaan. Metodi asettaa näytön Button-kontrollien enabled tiloonsa trueksi, asettaa TCP-palvelimelta saadun vastuksen käyttäjälle näkyviin sekä suorittaa virtuaalikonepuunäkymän päivityksen muissa tapauksissa paitsi infon haussa.

```csharp
void bgWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    EnabledAll(true);
    lblCon.Text = answer;

    // If answer not error continue
    if (answer.Substring(answer.IndexOf(" ")) != "Error")
    {
        switch ((string)e.Result)
        {
            case "Info":
                ResetButtons();
                infoBox.Clear();
                infoBox.Text = info;
                vmTab.SelectedIndex = 1;
                break;
            case "ConnectToServer":
            case "ConnectToVM":
            case "ControlVM":
            case "Register":
            case "UnRegister":
                updateTree();
                break;
            default:
                break;
        }
    }
}
```

Puunäkymän päivityksessä ohjelma asettaa kaikki Button-kontrollit vakiotiloihin, puhdistaa vanhan puunäkymän, asettaa puulle perussolmun ja käy läpi virtuaalikone XML-tiedoston. Tiedostosta ohjelma hakee virtuaalikoneen nimen, sekä tilan ja tilan mukaan asettaa virtuaalikoneen TreeView-kontrollin oikeaan solmuun.

```csharp
public void updateTree()
{
    ResetButtons();
}```
allVMTree.Nodes.Clear();

// Fill Tree with parent nodes
FillTree();

XmlDocument xmlDoc = new XmlDocument();
XPathNavigator nav;
XPathNodeIterator nodes;
XPathItem name;
XPathItem state;

xmlDoc.LoadXml(xml);
navigation = xmlDoc.CreateNavigator();
nodes = nav.Select("List/VM");
while (nodes.MoveNext())
{
    nav = nodes.Current.CreateNavigator();
    name = nav.SelectSingleNode("Name");
    state = nav.SelectSingleNode("State");

    switch (state.Value)
    {
        case ("Started"):
            allVMTree.Nodes[0].Nodes[0].Nodes[0].Nodes.Add(name.Value);
            allVMTree.Nodes[0].Expand();
            allVMTree.Nodes[0].Nodes[0].Nodes[0].Expand();
            allVMTree.Nodes[0].Nodes[0].Nodes[0].Nodes[0].Expand();
            break;
        case ("Stopped"):
            // Koodia
            default:
                break;
    }
}

ResetButtons()-metodi asettaa kaikkien näytön Button-kontrollien enable-
tilaksi falsen.

private void ResetButtons()
{
    comboBox.Enabled = false;
    conBtn.Enabled = false;
    regBtn.Enabled = false;
    infoBtn.Enabled = false;
    updateBtn.Enabled = true;
}

Käyttäjän valitessa solmun puunäkymästä, ohjelma asettaa Button-kontrollit
vakiotiloihin ja tarkistaa valitun solmun vanhemman tekstin, jonka
perusteella asetetaan käyttäjälle näkymään oikeat valinnat.

private void allVMTree_NodeMouseClick(object sender,
TreeNodeMouseClickEventArgs e)
{
    ResetButtons();
if (e.Node.Parent != null)
{
    if (e.Node.Parent.Text == "Not Connected")
    {
        infoBtn.Enabled = false;
        conBtn.Text = "Connect";
        conBtn.Tag = States.NotConnected;
        conBtn.Enabled = true;
        regBtn.Text = "Unregister";
        regBtn.Tag = States.NotConnected;
        regBtn.Enabled = true;
    }
    else if (e.Node.Parent.Text == "Stopped")
    {
        // Koodia
    }
}

Kuva 45. Yhdistämätön virtuaalikone valittuna


```csharp
public enum States
{
    Started,
    Stopped,
    Suspended,
    Connected,
    NotConnected,
    Registered,
    Unregistered
}

// Koodia
```
NotRegistered
}


private void conBtn_Click(object sender, EventArgs e)
{
    EnabledAll(false);

    if (conBtn.Tag.Equals(States.NotConnected))
    {
        tHandler.vmName = allVMTree.SelectedNode.Text;

        workToDo = "ConnectToVM";
        bgWorker.RunWorkerAsync("ConnectToVM");
    }
    else
    {
        switch ((States)conBtn.Tag)
        {
            case (States.Started):
                type = "Stop";
                break;
            case (States.Stopped):
                type = "Start";
                break;
            case (States.Suspended):
                type = "Resume";
                break;
            default:
                break;
        }

        tHandler.vmName = allVMTree.SelectedNode.Text;
        tHandler.type = type;
        tHandler.powerOp = comboBox.SelectedItem.ToString();

        workToDo = "ControlVM";
        bgWorker.RunWorkerAsync(workToDo);
    }
}
5.8.3 ThreadHandler-luokka

Kuva 47. Luokkakaavio
ThreadHandler hoitaa yhteydet TCP-palvelimeen. Handlerin konstruktorissa annetaan viittaus FormManageriin sekä annetaan ohjelman tarvitsevat tiedot, joilla saadaan yhteys TCP-palvelimeen.

```csharp
public ThreadHandler(FormManager main, string userName, string password, string ip, int port)
{
    this.main = main;
    this.userName = userName;
    this.password = password;
    this.ip = ip;
    this.port = port;
}
```


```csharp
public delegate void TaskReadyHandler(string answer, string xml);
public event TaskReadyHandler TaskReady;
public delegate void TaskFailedHandler(string answer);
public event TaskFailedHandler TaskFailed;
public delegate void InfoReadyHandler(string xml);
public event InfoReadyHandler InfoReady;
```

Luokan jokainen metodi kutsuu Connect()-metodia, jossa luodaan yhteys TCP-palvelimeen. Ohjelma lähettää palvelimelle käyttäjätunnuksen ja salasanan.

```csharp
public void Connect()
{
    client = new TcpClient();
    client.Connect(ip, port);
    stream = client.GetStream();
    write = new StreamWriter(stream);
    reader = new StreamReader(stream);
    
    write.WriteLine(userName);
    write.Flush();
    write.WriteLine(password);
    write.Flush();
}
```

oli hyväksynyt sisääkirjautumista, mutta vastuksena näkyy silloin
virheilmoitus.

```csharp
public void ConnectToServer()
{
    try
    {
        Connect();

        string ok = reader.ReadLine();
        if (ok == "ok")
        {
            write.WriteLine("ConnectToServer");
            write.Flush();
            answer = reader.ReadLine();
            xml = reader.ReadLine();
        }
        else
        {
            answer = ok;
        }

        if (TaskReady != null)
            TaskReady(answer, xml);
    }
    catch (Exception ex)
    {
        if (TaskFailed != null)
            TaskFailed("Error");
    }
}

Muut metodit toimivat aivan samalla tavalla kuin ConnectToServer()-metodi,
ainoana eroavaisuutena, mikäli yhteys on onnistunut, TCP-palvelimelle
välitetään enemmän hallintaan liittyviä parametreja. Esimerkkinä
virtuaalikoneen kontrolloimisessa käytettävä metodi, jossa välitetään
virtuaalikoneen nimi, käytettävä Power Option, sekä onko kyseessä
käynnistys, sammutuks vai jatkaminen (resume).

```
Info()-metodi eroaa muista metodeista hiukan, sillä se valmistuttuaan laukaisee valmistuttuaan InfoReady-eventin, TaskReady-eventin sijaan.

5.9 ASP.NET Dynaamiset web-sivut

5.9.1 Login page (Default.aspx)

Sivuille yhdistettäessä näkyviin tulee kirjautumissivu (kuva 48). Käyttäjän syötettyä käyttäjätunnuksen ja salasanan, otetaan yhteys SQL-palvelimelle ja haetaan käyttäjän salasan, sekä tieto onko käyttäjä normaalikäyttäjä vai onko käyttäjällä admin-oikeudet.

```csharp
SqlConnection SqlConn = new SqlConnection(conString);
SqlConn.Open();

// Selects password and admin from usertable for selected user
string cmd = "SELECT Password, Admin FROM Users WHERE UserName = "+ UserName +";"

SqlCommand SqlCmd = new SqlCommand(cmd, SqlConn);
SqlReader = SqlCmd.ExecuteReader();

string password = SqlReader["Password"].ToString().Trim();
if (Password == password)
{
    // Correct password, check admin state
    if (SqlReader["Admin"].ToString() == "True")
    
admin = true;
    
else
    
admin = false;

    Console.WriteLine("Admin: "+ admin);

    if (admin)
        
        // Redirect to admin page
        
        // Else redirect to normal user page
    
else
        
        // Redirect to normal user page

        // Else redirect to admin page
}
```

Kuva 48. Kirjautumissivu
Mikäli salasana ei vastaa palvelimen salasanaa, näytetään virheilmoitus (kuva 49). Myös tapauksessa, jolloin SQL-komento ei palauta yhtään riviä, todetaan, että käyttäjätunnus on virheellinen ja näytetään virheilmoitus.

```
return true;
```

Kuva 49. Virheilmoitus

Lopuksi käyttäjä ohjataan käyttöoikuuksia vastaavalleen sivulle.

```
if (admin)
   // Admins go to admin page
   Server.Transfer("AdminView.aspx", true);
else
   // Users to user page
   Server.Transfer("UserView.aspx", true);
```

5.9.2 Admin page (AdminView.aspx)

Kuva 50. Käyttäjätaulu valittuna

Ohjelma aloittaa hakemalla tietokannassa olevien taulujen nimet ja tallettaa tiedot käyttäjälle datasettiin. DataSetin tiedot käydään läpi ja jokainen rivi (taulun nimi) talletetaan DropDown-listaan.

```csharp
string cmd = "SELECT name FROM dbo.sysobjects WHERE xtype = 'U';
SqlCommand SqlCmd = new SqlCommand(cmd, SqlConn);
SqlDataAdapter dataA = new SqlDataAdapter(SqlCmd);
DataSet tables = new DataSet("Tables");
dataA.Fill(tables);

foreach (DataRow row in tables.Tables[0].Rows)
{
    string table = row.ItemArray[0].ToString();
    ddList.Items.Add(new ListItem(table));
}
```

GridView täytetään myös DropDown-listan valitun taulun mukaan. Ohjelma täyttää tietokannasta uuden DataGensetin hakemalla valitun taulun kaikki tiedot ja asettaa datasetin GridViewin tietolähteeksi.

```csharp
public DataSet GetSet(string command)
{
    SqlConnection SqlConn = new SqlConnection(conString);
    return GetSet(command);
}
```

SqlConn.Open();
SqlCommand = new SqlCommand(command, SqlConn);
dataA = new SqlDataAdapter(SqlCmd);
set = new DataSet("set");
dataA.Fill(set);

return set;
}

GridView4.DataSource = set;
GridView4.DataBind();

Aktiivinen näkymä valitaan automaattisesti listasta valitun taulun mukaan. Metodi käy läpi kaikki MultiViewin kontrollit ja vertaa jokaisen kontrollin ID:tä DropDown-listasta valitun taulun nimeen.

private void SetActiveView(DropDownList dropDownList)
 {
  // Goes through all controls in MultiView
  foreach (Control c in MultiView1.Controls)
  {
    // If controls ID is match with selected value
    if (c.ID == dropDownList.SelectedValue)
    {
      MultiView1.SetActiveView((View)c);
      break;
    }
  }
}


private void GetUserData()
 {
  fillList(ddServer);

}
chkBoxAdmin.Checked = ((CheckBox)GridView4.SelectedRow.Cells[7].Controls[0]).Checked;
}

private void fillList(DropDownList list)
{
    string cmd = "";

    switch (list.ID)
    {
        case ("ddOwner"):
            cmd = "SELECT ID, userName FROM Users";
            break;
        case ("ddServer"):
            cmd = "SELECT ID, Name FROM Servers";
            break;
        default:
            break;
    }

    set = GetSet(cmd);

    list.Items.Clear();

    foreach (DataRow row in set.Tables[0].Rows)
    {
        // Display in combox "1, UserName" and value 1
        list.Items.Add(new ListItem(row.ItemArray[0].ToString() + ", " + row.ItemArray[1], row.ItemArray[0].ToString()));
    }
}
Kuva 51. Virtuaalikone taulusta tehty valinta

Käyttäjän valitseman toiminnon (update, insert, delete) mukaan ohjelma tekee SQL-lauseen, suorittaa sen ja päivittää GridViewin vastaamaan uutta tulosta (kuva 51). Alla esimerkissä luodaan komento, jolla päivitetään VirtualMachines-taulu.

```csharp
string command = "UPDATE " + table + " ";

switch (table)
{
    case "VirtualMachines":
        command += "SET Name = '" + txtVmName.Text + ", ";
        command += "Folder = '" + txtVmFolder.Text + ", ";
        command += "Owner = '" + ddOwner.SelectedValue + ", ";
        command += "WHERE ID = " +
        break;
    case "Users":
        // Koodia
        break;
}

SqlConn = new SqlConnection(conString);
SqlConn.Open();
SqlCommand = new SqlCommand(command, SqlConn);
SqlCommand.ExecuteNonQuery();
```
Ennen suoritusta ohjelma tarkastaa myös TextBoxien tekstit, jottei yksikään ole jäänyt tyhjäksi, sillä tyhjä päivitysteksti aiheuttaa virheilmoituksen SQL-palvelimella. Jos tyhjä TextBox löytyy, ohjelma laukaisee javascriptillä huomautusikkunan eikä suorita päivitystä.

```
foreach (Control c in MultiView1.Views[MultiView1.ActiveViewIndex].Controls)
{
    if (c.GetType().Name == "TextBox")
    {
        if (((TextBox)c).Text == "" || ((TextBox)c).Text == null)
        {
            Response.Write("<script language='Javascript'>" + "alert("'" + ((TextBox)c).ID + " cant be empty");" + "</script>"));
            return false;
        }
    }
}
```

**Kuva 52. Virtuaalikone valittuna päivitystä varten**
Kuva 53. Käyttäjän aloitusnäkymä

Sivun latauessa (kuva 53) käyttäjälle luodaan oma tunnisteavain, joka koostuu käyttäjän IP-osoitteen sekä käyttäjätunnuksen HashCodesta. Tunnisteavainta käytetään tiedon tallentamisen koneen välimerkistä.

```csharp
private int CreateHashCode()
{
    // Create Hashcode using ip and logon name
}
```


```csharp
if (!Page.IsPostBack)
{
    ASP.default_aspx def = new ASP.default_aspx();
    def = (ASP.default_aspx)Context.Handler;
    userName = def.UserName;
    password = def.Password;
    ip = def.Ip;
    port = def.Port;
}
tHandler = new ThreadHandler(userName, password, ip, port);
tHandler.ready += new ThreadHandler.threadHandler(tHandler_ready);

lblWelcome.Text = "Welcome " + userName;

time = DateTime.Now.Hour + ":" + DateTime.Now.Minute + ":" + DateTime.Now.Second;
lblTime.Text = "Login time: " + time;

ConnectToServer();
}

Yhteys palvelimeen otetaan käynnistämällä uudessa säikeessa ThreadHandlerin Connect()-metodi.

public void ConnectToServer()
{
    thread = new Thread(new ThreadStart(tHandler.Connect));
    thread.Start();
}

Mikäli sivu on ladattu jo aiemmin, eli kutsun tyyppi on PostBack, haetaan käyttäjän ThreadHandler-luokka välimuistista käyttäjälle luotua avainta. Lisäksi asetetaan virtuaalikone näkymän datalähteeksi ThreadHandlerin XML-data.

// Get handler from cache
Handler = (ThreadHandler)Cache[hash.ToString() + ":Handler"];  
xml = tHandler.Xml;

XmlDataSource.Data = tHandler.Xml;
Kuva 54. Virtuaalikone valittuna

Kun valitaan virtuaalikone TreeViewistä (kuva 54), päivitetään käyttäjälle näkyvillä olevat valinnat. Ohjelma vertaa puusta valitun solmun parentin tekstiä ja lisää DropDown-listaan käyttäjälle oikeat valinnat.

```csharp
protected void VmTree_SelectedNodeChanged(object sender, EventArgs e)
{
    // Selected virtual machine
    tHandler.vmName = VmTree.SelectedNode.Text;
    ddList.Items.Clear();

    if (VmTree.SelectedNode.Parent != null)
    {
        if (VmTree.SelectedNode.Parent.Text.Equals("Not Connected"))
        {
            ddList.Items.Add("Connect");
            ddList.Items.Add("Unregister");
        }
        else if (VmTree.SelectedNode.Parent.Text.Equals("Stopped"))
        {
            ddList.Items.Add("Start");
            Info();
        }
        // Koodia
    }
}
```
Kuva 55. Yhteys luotuna virtuaalikoneeseen

Mikäli virtuaalikoneeseen on jo yhteys (kuva 55), kun virtuaalikone valitaan, päivitetään virtuaalikoneen info-teksti käynnistämällä ThreadHandlerin Info()-metodi uudessa säikeessä (kuva 56).

```csharp
public void Info()
{
    thread = new Thread(new ThreadStart(tHandler.Info));
    thread.Start();
    lblInfo.DataBind();
}
```

DataBind()-metodi käskee tekstikenttää päivittämään tekstisisältönsä. Päivityskäsky odottaa, että sääie saa suoritettua infonhakuna loppuun. Sen jälkeen se päivittää tekstiksensä ThreadHandlerin hakeman info-tekstin.

```csharp
protected void lblInfo_DataBinding(object sender, EventArgs e)
{
    while (thread.IsAlive)
    { }
    lblInfo.Text = ParseInfo(tHandler.InfoString);
}
```
Kuva 56. Näytetään virtuaalikoneen lisäinfo

Kun käyttäjä painaa Do-nappia, valitaan oikea toimenpide DropDown-listan valinnan mukaan. ThreadHandler luokalle välitetään tarvittavat hallintaparametrit samaan tapaan kuin Forms-pohjaisessa sovelluksessa. Toimenpide käynnistetään samalla tavalla kuin aiemmin esitetyt Connect()-ja Info()-metodit.

protected void btnPerform_Click(object sender, EventArgs e)
{
    // If null not vm selected
    if (ddList.SelectedItem != null)
    {
        powerop = ddPowerOp.SelectedItem.Text;
        tHandler.powerOp = powerop;

        switch (ddList.SelectedItem.Text)
        {
            case "Connect":
                ConnectToVM();
                break;
            case "Register":
                RegisterVM();
                break;
            case "Unregister":
                UnRegisterVM();
                break;
            case "Start":
                StartVM();
                break;
        }
    }
}
tHandler.Type = "Start";
ControlVM();
break;
// Koodia
}

VmTree.DataBind();
}


protected voidVmTree_DataBound(object sender, EventArgs e)
{
    // Wait that tHandler is ready
    while (thread.IsAlive)
    {
    
    Cache[hash.ToString() + "Handler"] = tHandler;
this.xml = tHandler.Xml;

    ddList.Items.Clear();
    FillTree();
    
    XmlDocument xmlDoc = new XmlDocument();
    XPathNavigator nav;
    XPathNodeIterator nodes;
    XPathItem name;
    XPathItem state;
    xmlDoc.LoadXml(xml);
    nav = xmlDoc.CreateNavigator();
    nodes = nav.Select("List/VM");
    while (nodes.MoveNext())
    {
    nav = nodes.Current.CreateNavigator();
    name = nav.SelectSingleNode("Name");
    state = nav.SelectSingleNode("State");

    TreeNode t = new TreeNode(name.Value);
    t.SelectAction = TreeNodeSelectAction.Select;
    //t.NavigateUrl = "javascript:void(0);";
    switch (state.Value)
    {
    case ("Started"):
5.9.4 ThreadHandler-luokka


```csharp
public void ConnectToServer()
{
    try
    {
        Connect();

        string ok = read.ReadLine();
        if (ok == "ok")
        {
            write.WriteLine("ConnectToServer");
            write.Flush();
            answer = read.ReadLine();
            xml = read.ReadLine();
        }
        else
        {
            answer = ok;
        }

        stream.Close();
    }
    catch (Exception ex)
    {
        answer = "Error";
    }
}
```
5.10 Logger

Logger::Log

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log()</td>
<td>- writMessage(in type : string, in user : string, in action : string)</td>
</tr>
<tr>
<td></td>
<td>+ getInstance() : Log</td>
</tr>
<tr>
<td></td>
<td>+ Write(in action : string)</td>
</tr>
<tr>
<td></td>
<td>+ Write(userName : string, in action : string)</td>
</tr>
<tr>
<td></td>
<td>+ WriteError(in error : string)</td>
</tr>
<tr>
<td></td>
<td>+ WriteError(in userName : string, in error : string)</td>
</tr>
</tbody>
</table>

![](image)

**Kuva 57. Luokkakaavio**

Logger-ohjelma kirjoittaa tekstitiedostoon tapahtumia sekä virheilmoituksia. Loki-tiedosto (ServerLog.txt) löytyy käyttäjän väliaikaistentiedostojen kansiosta. Windowsissä kansio löytyy helpoiten kirjoittamalla osoiteriville %temp% Luo-merkintä on kirjoitettu XML-muodossa allaolevan esimerkin mukaan.

```xml
<Message Type='Action' Time='28.9.2007 15:49:23'
User='System'>Server started</Message>
```


Logger–ohjelma on totetutettu Singleton Patternilla, jolloin ohjelmasta voi olla olemassa vain yksi instanssi. Tämä estää saman tiedoston samanaikaisen käytön, mikäli monilla olioilla on logger-ohjelma käytössä.

Ohjelman konstruktiosissa haetaan vain käyttäjän temp-hakemisto.

```csharp
tempFolder = Path.GetTempPath();
```

Ohjelman joka käyttää logger-ohjelmaa, pyytää logger ohjelman-instanssin getInstance()-metodilla.

```csharp
private static Log log;

public static Log getInstance()
{
    // Prevents two thread to access at the same time
    lock (typeof(Log))
    {
        if (log == null)
            log = new Log();
    }
```

6 YHTEENVETO

Työlle annetuissa tavoitteissa onnistuttiin hyvin ja työssä onnistuttiin luomaan hyvä pohja, josta on helppo jatkaa ohjelmointirajapintojen syvällisempää tutkimista ja ohjelmien jatkokehitystä.

Vix API ja VmCOM API stand-alone-ohjelmien tarkoituksena oli auttaa rajapintoihin tutustumisessa, joten niiden kehitys lopetettiin melko aikaisessa vaiheessa. VmCOM API -palvelinpohjaisen ohjelman pääasiaalinen tarkoitus oli selvittää TCP-palvelimen ja ohjelman yhteystyötä ja helpottaa skriptien siirtämistä ASP.NET-toteutukseen.

TCP-palvelin ratkaisuun päädyttiin, jotta käyttäjänhallintaa saadaan helpotettua. TCP-palvelimen avulla käyttäjän ei itse tarvitse huolehtia tietojen oikeellisuudesta, vaan vastuu saadaan siirrettyä päakäyttäjille. Samalla saatiin myös eristettyä VMware Serverin yhteys varsinaisesta client-ohjelmasta.

Tiedonvälitys päättettiin hoitaa XML-muodossa, jolloin monien ohjelmien olisi mahdollisimman helppo käyttää samaa dataa. Näin ollen sekä VmCOM API palvelinpohjainen ohjelma ja dynaamiset web-sivut voivat käyttää samaa palvelinta.

6.1 Ongelmat

Alun suurimmat ongelmat olivat dokumentaation puutteessa, erityisesti kehityssympäristön toimintakuntoon laittamisesta oli erityisin vaikea löytää dokumentaatiota. Suureksi avuksi olivat internetin monet keskustelupalstat. Tästä syystä kappaleessa 4 käsittelään käyttöönotto varsin perinpohjaisesti.
VMwaren omat dokumentaatiot ohjelmointirajapinnoista (lähteet 1, 2 ja 9) olivat varsin kattavat.

Työn suurimpia haasteita olivat uudet ohjelointikielet C# ja ASP.NET. Varsinkin ASP.NET aiheutti ongelmia, sillä web-ohjelmoinnin opetus opiskeluakanani oli varsin vähäistä.

TCP-palvelimen kanssa oli monia ongelmia, joista suurin oli socket-yhteyksien katoaminen. Lopulta päädyttiin ratkaisuun, jolloin käyttäjän toimenpide ottaa uuden yhteyden palvelimeen. Tämä kuluttaa kuitenkin turhaa kaistaa, mutta näin myös välttyään toimenpidettä, jossa pitää ennen jokaista toimenpidettä tutkia yhteyden tila ja se, onko yhteys vielä käytettävissä.

ASP.NET-käyttäjänhallintasivut oli aluksi tarkoitus luoda täysin dynaamisesti, mutta se olisi vaatinut muutoksia tietokantatauluihin. Silloin taulun jokaiselle sarakeelle oltaisiin luotu dynaamisesti näkymään TextBox-kontrolli. Mikäli sarake viittaa toiseen tauluun, luotaisiin ComboBox-kontrolli. Tässä versiossa ainoaksi dynaamisuudeksi jäi taulujen haku ComboBox-kontrolliin ja jokaiselle taululle jouduttiin luomaan etukäteen näkymän MultiView-kontrollin sisään.

6.2 Parannusehdotukset

Dynaamisten web-sivujen käyttäjäsivuihin olisi hyvä tuoda näkyviin jonkinlainen progress-kontrolli toimenpiteiden suorituksen ajaksi. Nyt käyttäjälle saattaa jäädä epäselväksi, onko ohjelma kaatunut vai tapahtuuko taustalla jotain.

Käyttäjänhallintasivu voisi lähettää TCP-palvelimelle viestin, kun kannassa on tapahtunut muutos, jolloin TCP-palvelin kävisi käyttäjäkokoelmanssä läpi ja päivittäisi muuttuneet tiedot. Nykyisessä versiossa TCP-palvelin on käynnistettävää uudestaan, jotta käyttäjäkokoelmat saadaan päivitettyä.

Jokaisella käyttäjällä voi olla vain yhdekkä VMware Serverillä virtuaalikoneita. Tämä tulisi muuttaa siten, että käyttäjälle ei olisi määrittynä omaa VMware Serveria, vaan jokaiselle virtuaalikoneelle olisi tauluissa määritetty, mille palvelimelle virtuaalikone kuuluu. Tämä muutos aiheuttaa myös muutoksia
TCP-palvelimen käyttäjäkokoelmaan sekä Handler-luokan yhteydenhallintakoodeihin.

VmManager-luokan voisin muuttaa toimimaan enemmän rajapintana client-ohjelman ja TCP-palvelimen välillä, jolloin sitä ei tarvitsisi erikseen muokata jokaiselle ohjelmalle, joka TCP-palvelinta tulee käyttämään.

Client-ohjelmat voisivat sisältää asetuksille oman ikkunan, jolloin käyttäjän ei pitäisi tekstieditorilla muokata asetuksien XML-tiedostoa.

Parannettavaa jäi myös ohjelmien virheidenhallintaan ja virheenvälitykseen. Tällä hetkellä palvelin palauttaa usein pelkistetyn virheilmoituksen, jolloin käyttäjän on vaikea tietää, mikä on vialla.
VIITELUETTELO


VIRTUAALIKONEIDEN ASENNUS


Kuva 1. Connect to Host

1. Valitaan yhdistettäväksi hostiksi Local Host, mikäli yhdistetään käytössä olevaan tietokoneeseen asennettuun VMware Serveriin.

Kuva 2. Päänäkymä


4. Valitaa Custom konfiguraatio, jolloin voidaan antaa tarkempia määritelyksiä asennuksen suhteen.
Kuva 5. Select a Guest operating System

5. Valitaan käyttöjärjestelmä asennettavan käyttöjärjestelmän mukaan. Tässä asennuksessa valitaan käyttöjärjestelmäksi Linux ja Ubuntu 64-bit.

Kuva 6. Name the Virtual Machine

Kuva 7. Set Access Rights

7. Laitetaan virtuaalikone vain oman käyttäjätunnuksen käyttöön.

Kuva 8. Startup / Shutdown Options

Kuva 9. Processor Configuration

9. Luodaan virtuaalikoneelle kaksi virtuaalista prosessoria.

Kuva 10. Memory for the Virtual Machine

Kuva 11. Network Type

11. Asetetaan Network Address Translator käyttöön.

Kuva 12. Select a Disk
12. Luodaan uusi virtuaalinen kovalevy.

Kuva 13. Select a Disk Type

13. Asetetaan kovalevyn tyypiksi SCSI.

Kuva 14. Specify Disk Capacity

14. Asetetaan kovalevyn kooksi 5GB.
Kuva 15. Specify Disk File


Kuva 16. Creating Disk Progress


1. CD-ROM asema tuplaklikkaamalla avautuu cd-aseman asetusvalikko.

2. Valitaan Use ISO image ja valitaan käyttöjärjestelmän image-tiedosto.
3. Klikataan Start the virtual machinea.

4. Valitaan Ubuntun aloitusvalikosta Install-pikakuva, josta käynnistyy käyttöjärjestelmän asennus.