STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN 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

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ALKULAUSE

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

Espoossa 12.11.2007

Tomi Tuhkanen

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Tomi Tuhkanen	
Työn nimi: Virtuaalikoneiden hallinta ohjelmointirajapintaa käyttäen	
Päivämäärä: 12.11.2007	Sivumäärä: 86 s. + 10 liitettä
Koulutusohjelma: Tietotekniikka	Suuntautumisvaihtoehto: Ohjelmistotekniikka
Työn valvoja: yliopettaja Matti Luukkainen Työn ohjaaja: yliopettaja Matti Luukkainen	

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 –ohjelmaan.

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

Windows käyttöjärjestelmälle luodaan Client-ohjelma. Ohjelmasta luodaan suoraan VMware Serveriin 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–ohjelmat, dynaamiset web-sivut sekä TCP-palvelin on kirjoitettu C#-ohjelmointikielellä.

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

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Tomi Tuhkanen

Title: Managing virtual machines using programming API

Date: 12.11.2007 Number of pages: 86 + 10

Department: Information Technology Study Programme: Software Engineering

Instructor: Matti Luukkainen

Supervisor: Matti Luukkainen

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

ALKULAUSE

TIIVISTELMÄ

ABSTRACT

SISÄLLYS

1 JOHDANTO 1

2 VIRTUALISOINTI 1

2.1 Virtuaalikonevalvoja 1

2.2 Rautavirtuaalikoneet 2

2.3 Ohjemistovirtuaalikoneet 2

2.4 Laitteiden virtualisointi 3

2.5 Ohjelmien virtualisointi 5

2.6 Esityksen virtualisointi 6

2.7 Hyödyt 6
2.7.1 Ositus 6
2.7.2 Eristyvyys 8
2.7.3 Kapselointi 9
2.7.4 Muokattavuus 9

3 VMWARE 10

3.1 VMware Server 10

3.2 Ohjemointirajapinnat 11

4 OHJELMISTOJEN ASENNUS 11

4.1 Server 11

4.2 Programming API:n (Vix) käyttöönotto 12
4.2.1 Clientin kääntäminen VMware Server asennettuna 12
4.2.2 Clientin kääntäminen ilman VMware Serveriä 16

4.3 VMware COM API:n käyttöönotto 16

5 HALLINTAOHJELMIEN JA –SKRIPTIEN SUUNNITTELU JA TOTEUTTAMINEN 18

5.1 Ohjelmointiympäristöt 18

5.2 Tavoitteet 19

5.3 Programmin API-skriptit 22
5.3.1 Handlet 22
5.3.2 Yhteydenotto VMware Serveriin 22
5.3.3 Virtuaalikoneen rekisteröinti 23
5.3.4 Handlen ottaminen virtuaalikoneeseen 23
5.3.5 Virtuaalikoneiden käynnistus / sammutus 24
5.3.6 Handlen lopetuksen odottaminen 25

5.4 Vix API -ohjelma 25

5.5 VmCOM API -skriptit 28
5.5.1 API:n oliot 28
5.5.2 Yhteydenotto palvelimeen 29
5.5.3 Virtuaalikoneen rekisteröinti 30
5.5.4 Yhteydenotto virtuaalikoneseen 30
5.5.5 Virtuaalikoneiden käynnistus / sammutus 30

5.6 VmCOM API stand-alone ohjelma 32
5.6.1 VmManager-luokka 32
5.6.2 MainForm-luokka 36

5.7 TCP-palvelin 41
5.7.1 Server-luokka 45
5.7.2 Handler-luokka 49
5.7.3 JobManager-luokka 52
5.7.4 VmManager-luokka 53

5.8 VmCOM API palvelinpohjainen ohjelma 54
5.8.1 MainForm-luokka 56
5.8.2 FormManager-luokka 57
5.8.3 ThreadHandler-luokka 64

5.9 ASP.NET Dynaamiset web-sivut 67
5.9.1 Login page (Default.aspx) 68
5.9.2 Admin page (AdimnView.aspx) 69
5.9.3 User Page (UserView) 75
5.9.4 ThreadHandler-luokka 81

5.10 Logger 82

6 YHTEENVETO 83

6.1 Ongelmat 83

6.2 Parannusehdotukset 84

VIITELUETTELO 86

LIITTEET

1 JOHDANTO

Virtualisoinnin yleistyttyä työpaikoilla ja kouluissa virtualisoinnin tuomiin mahdollisuuksiin on tutustuttava yhä paremmin. Työssä käsitellään 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. Näinollen 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

Tietojenkäsittelyssä virtualisoinnilla tarkoitetaan tekniikkaa, jossa tietokoneen resurssien piirteet piilotetaan muilta ulkoisilta käyttäjiltä. Näin yksi resurssi voidaan jakaa moniin suorittaviin resursseihin tai monta resurssia voidaan näyttää yhtenä resurssina. Virtualisoinnilla voidaan myös luoda resursseja, joita ei oikeasti ole olemassa.

2.1 Virtuaalikonevalvoja

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

2.2 Rautavirtuaalikoneet

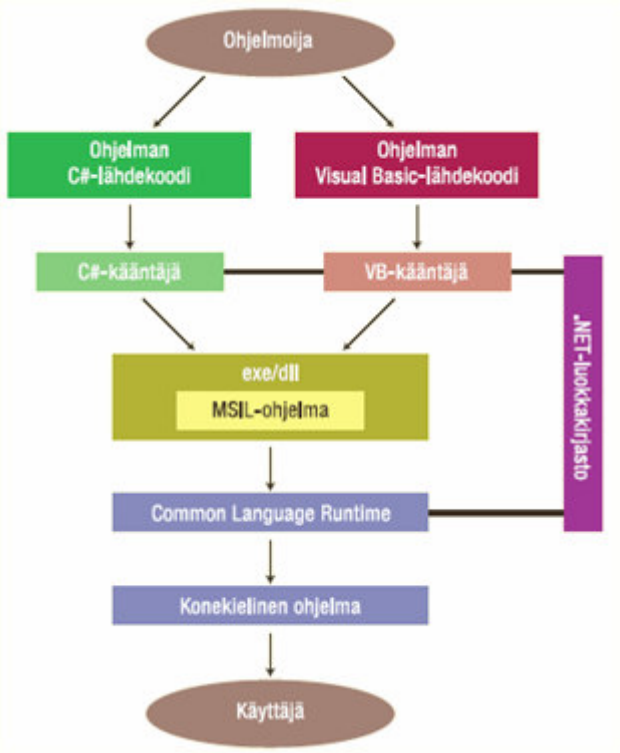
Rautavirtuaalikoneella, eng. Hardware Virtual Machine, tarkoitetaan ohjelmistosäiliötä, joka on toisen tietokoneen sisään ohjelman (virtuaalikonevalvojan) avulla luotu itsenäisesti toimiva tietokone. Ohjelmistosäiliö (kuva 1) sisältää käyttöjärjestelmän, ohjelmat, sekä ohjelmallisesti luodut prosessorit, muistit, kovelevyt ja muun raudan. Virtuaalikoneista puhuttaessa tarkoitetaan yleensä rautavirtuaalikonetta. Tietokoneesta, jonne virtuaalikonevalvoja on asennettu, käytetään nimitystä host ja virtuaalikoneesta nimitystä quest.

Kuva 1. Ohjelmistosäiliö

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

2.3 Ohjemistovirtuaalikoneet

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

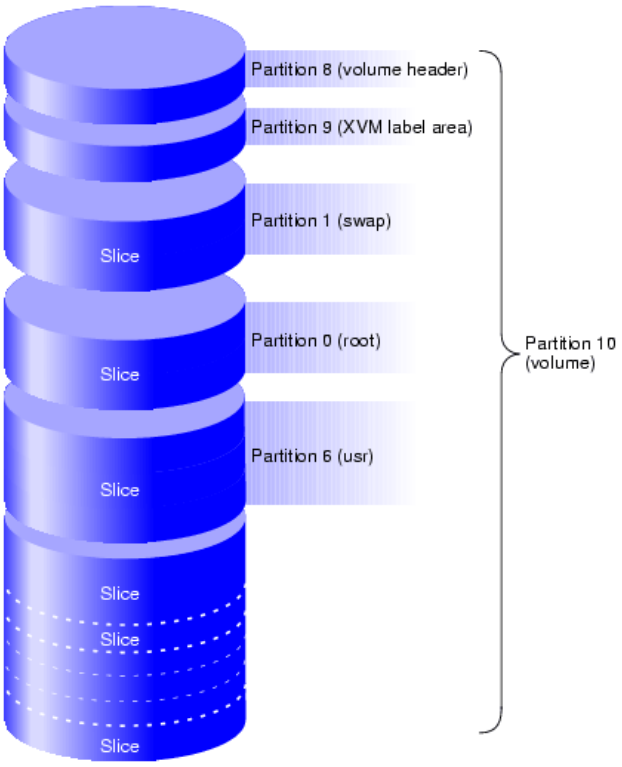


Kuva 2. .NET-sovelluksen rakenne [7.]

Ensimmäiset ohjelmistokielten virtuaalikoneet tulivat jo 70-luvulla, ja nykyisistä tunnetuimmat ovat Java Virtual Machine ja Microsoftin Common Language Runtime (CLR). Kuvassa 2 CLR:n yläpuolella on ohjelman lähdekoodiin viittaavat komponentit, jotka CLR kääntää alustalle ymmärrettäväksi konekieliseksi ohjelmaksi. [5.]

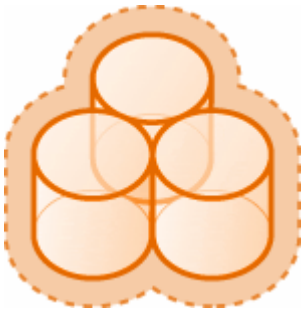
2.4 Laitteiden virtualisointi

Laitteiden virtualisoinnilla tarkoitetaan laitteen tai laitteiden resurssien jakamista tai yhdistämistä eri kokonaisuuksiksi. Osituksessa yksi resurssi jaetaan moneksi resurssiksi. Useimmille tietokoneenkäyttäjille tutuin muoto laitteiden virtualisoinnista on kovalevyn jakaminen pienempiin osiin (partitioihin) (kuva 3).



Kuva 3. Kovalevyn ositus

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.]

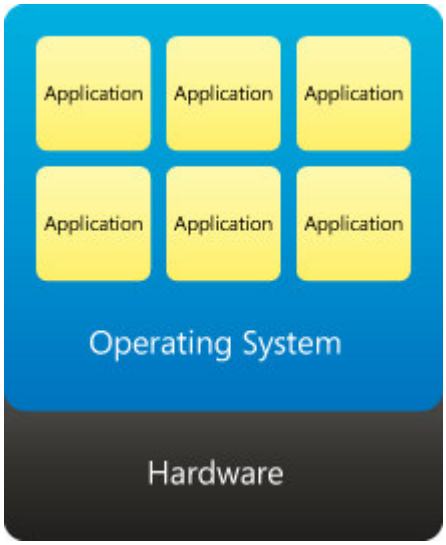


Kuva 4. Kolme tietovarastoa virtualisoituna yhdeksi

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

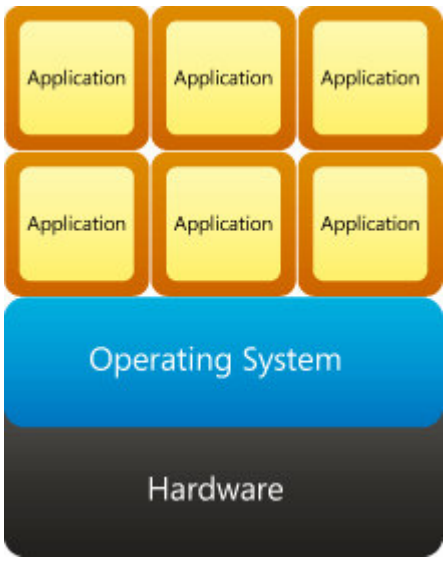
2.5 Ohjelmien virtualisointi

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

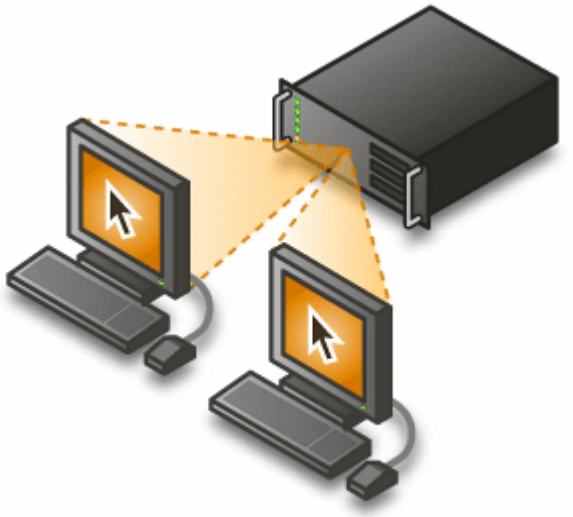
Jokainen virtuaaliohjelma luo itselleen asiakaskoneelle ohjelmakohtaisen konfiguraation ja kopioit tarvitsemistaa jaetuista resursseista. Ohjelmat ovat eristyksissä käyttöjärjestelmän lisäksi myös toisistaan (kuva 6). [11.]



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 käyttöliittymänsä etänä toiselle koneelle. Istunnoissa voi olla käynnissä yhdestä ohjelmasta koko työpöydän kattavaan ohjelmistoon.

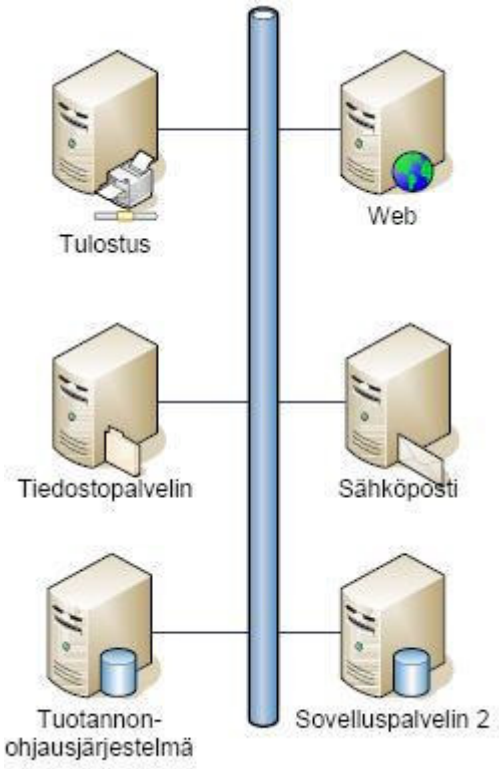
Monet istunnot voivat käyttää samanaikaisesti samoja palvelimelle asennettuja ohjelmia. [11.]

2.7 Hyödyt

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

2.7.1 Ositus

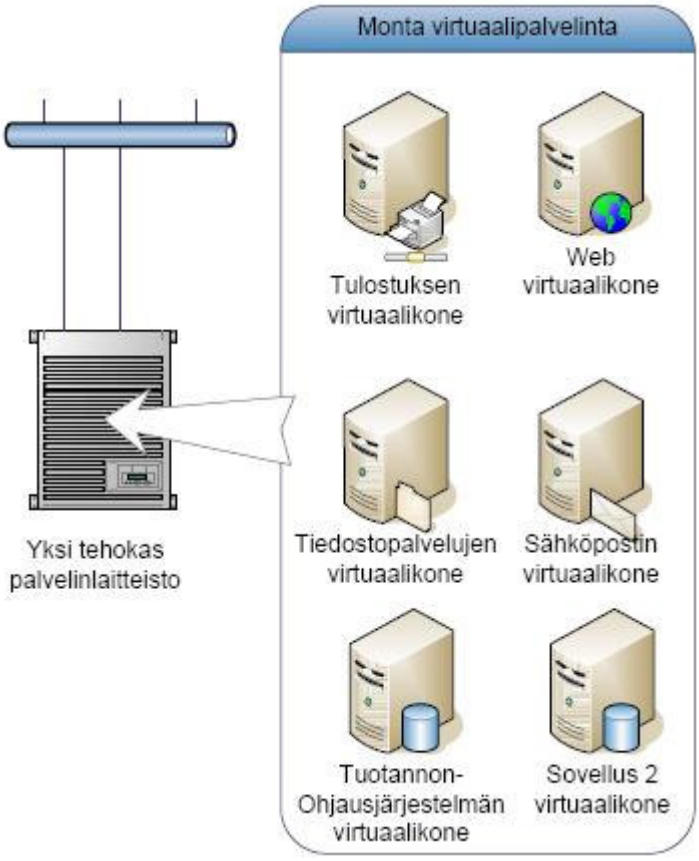
Monilla yrityksillä on lähes kaikille tehtäville oma palvelimensa (kuva 8). Tällöin palvelimet käyttävät arviolta vain noin 5-30 % tehostaan. Jokainen palvelin myös kuluttaa paljon virtaa, vaikka ne eivät toimiskaan mahdollisimman tehokkaasti.



Kuva 8. Perinteinen tapa, erillinen palvelin kaikille järjestelmille [6.]

Virtualisoinnilla voidaan palvelimet keskittää yhdelle tehokkaammalle palvelimelle, jolloin yhdestä koneesta saadaan maksimaalinen teho / käyttösuhde (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.

Osituksessa 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 Eristyvyys

Virtuaalikoneet ovat eristettynä isäntäkoneesta ja toisista virtuaalikoneista. Näin data ei pääse vuotamaan virtuaalikoneiden ja isäntäkoneen välillä. Virtuaalikoneet ja niille asennetut ohjelmat voivat keskustella ainoastaan luotuja virtuaalisia verkkoyhteyksiä pitkin.

Eristyvyydellä 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 tarvitaa 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 illuusion raudasta tai rautakonfiguraatioista joita oikeasti ei ole käytössä. Näin testitilanne saadaan luotua täysin vaaditulle konfiguraatiolle. [5.]

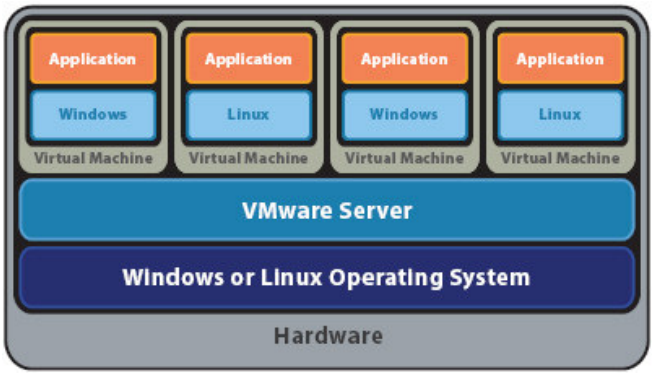
3 VMWARE

VMware on vuonna 1998 perustetty yhdysvaltalainen yritys, joka on yksi maailman johtavista virtualisointisovellusten toimittajista x86-pohjaisille palvelimin- 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 toimiakseen käyttöjärjestelmän. VMware Server toimii sekä virtuaalikonevalvojana että apuohjelmana, joka on käynnissä käyttöjärjestelmän päällä eristäen virtuaalokonevalvojan käyttöjärjestelmästä, jolloin se voi käyttää resurssejaan paremmin virtuaalikoneiden hallintaan (kuva 11). [5.]



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

3.2 Ohjemointirajapinnat

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:sta käytetään usein nimeä Vix. [2, s. 1.]

VMwaren VmCOM API -rajapinta on myös kirjoitettu C-kielellä, mutta se perustuu Component Object Modeliin (COM). COM on pääosin vain Microsoft Windows käytössä. Insinöörityön pääasiallinen koodi tullaan tekemään VmCOM API:a käyttäen, C#-ohjelmointikielellä, jolloin dynaamiset web-sivut voidaan toteuttaa ASP.NET teknologialla. [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 virtuaalikyttöjärjestelmän suorituskykyä ja toiminnallisuutta. Paketti sisältää muunmuassa hiiri- ja näytönohjainajurit käyttöjärjestelmille. [2, s. 36.]

4 OHJELMISTOJEN ASENNUS

4.1 Server

VMware Server on saatavilla VMwaren kotisivuilta. Insinöörityötä tehdessä se oli saatavilla osoitteessa http://www.vmware.com/download/server/. Käyttäjän on rekisteröidyttävä VMwaren sivuilla, jotta saa ilmaisen sarjanumeron.

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.

Mikäli VMware Server on koneeseen asennettu, muita komponentteja ei tarvitse asentaa. Mikäli haluat käyttää Programming API:a koneessa, johon ei ole serveriä asennettu, joudutaan kopioimaan muutamia tiedostoja koneesta, johon Server on asennettuna.

Esimerkissä asetukset on asetettu Visual C++ 2005 Express Editioniin. Ohjeessa oleva [asennushakemisto] viittaa hakemistoon johon VMware Server on asennettuna.

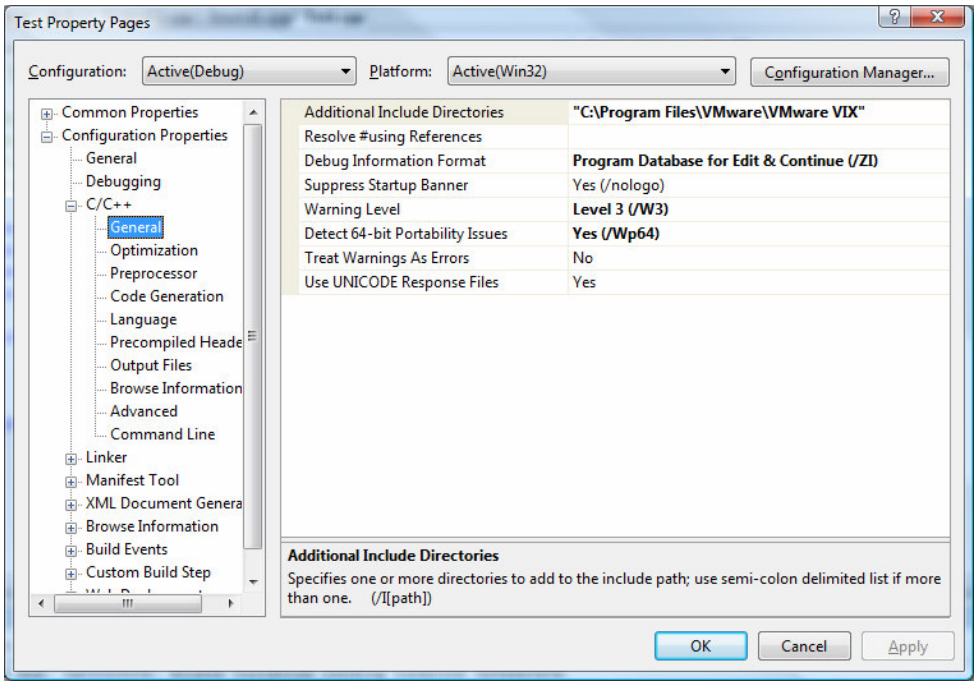
4.2.1 Clientin kääntäminen VMware Server asennettuna

1. Lisää Client–ohjelman kooditiedoston alkuun include-määritys

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

2. Aseta kehitysympäristön include pathiin [asennushakemisto]\VMware Vix\

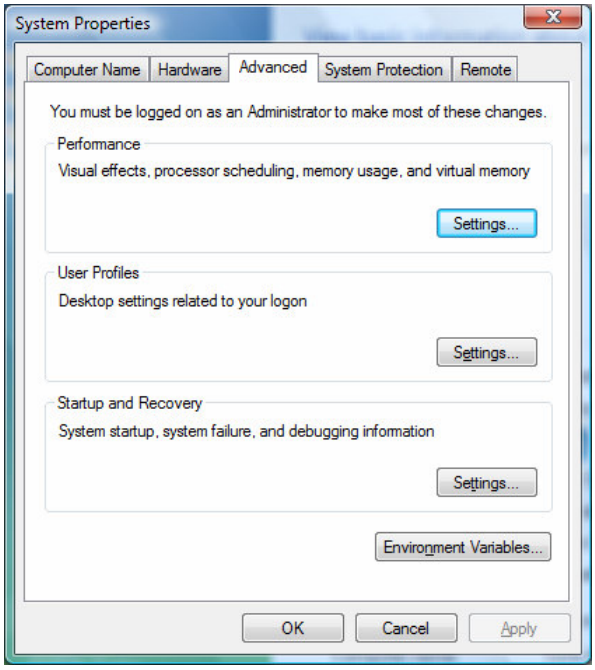
   Project -> Properties -> Configuration Properties



Kuva 12. Project Property Page

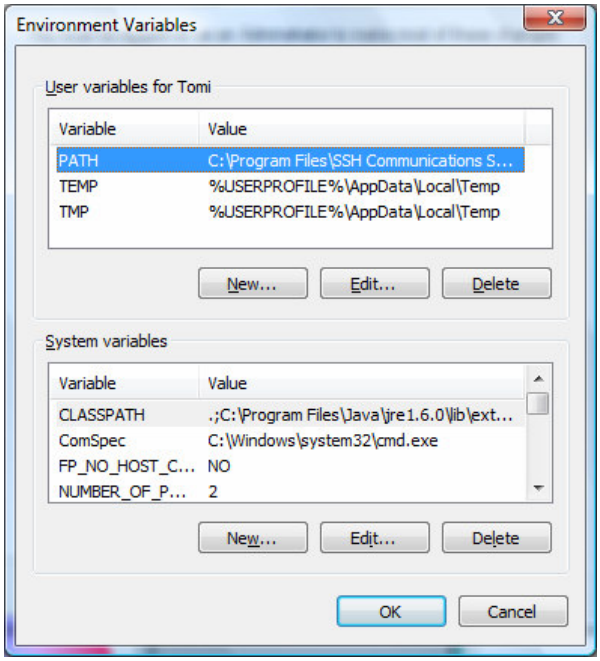
Lisää [asennushakemisto]\VMware Vix kohtaan, C/C++ -> General -> Additional Include Directories.

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



Kuva 13. System Properties

System Propertiesta valitse Enviromental Variables. System Properties löytyy My Computerin alta (My computer –> Properties).



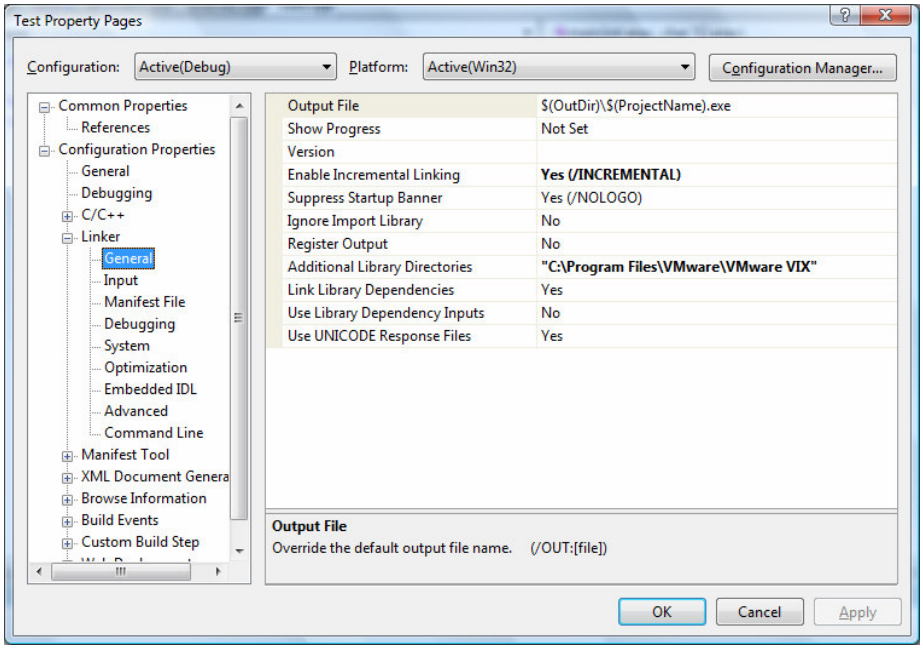
Kuva 14. Enviroment Variables

Enviromental 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.

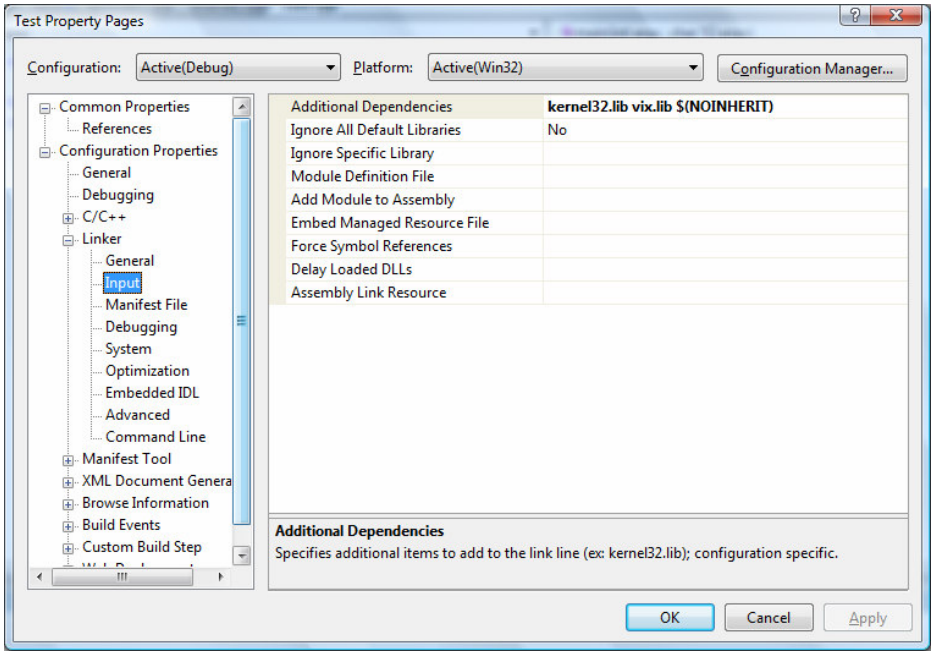
4. Aseta kääntäjä yhdistämään vix.lib-tiedostoon staattisesti

   Project -> Properties -> Configuration Properties



Kuva 15. Project Property Page

Lisää [asennushakemisto]\VMware Vix\ kohtaan, Linker -> General -> Additional Library Directories.



Kuva 16. Project Property Page

Lisää Vix.lib kohtaan, Linker -> Input -> Additional Dependecies.

4.2.2 Clientin kääntäminen ilman VMware Serveriä

Tarvittavat 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\.

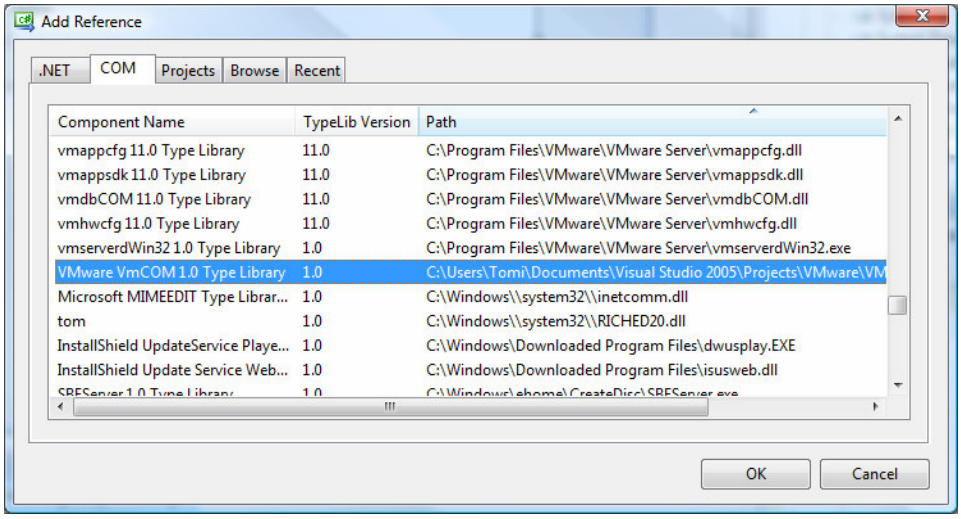
1. Asetetaan ympäristö lisäämään DLL-tiedostot [asennushakemisto]\ VMware VmCOM Scripting API\ hakemistosta tai kopioidaan tiedostot projektin debug-kansioon.

2. Rekisteröidään komentoliittymästä VmCOM.dll komennolla

   regsrv32 ”[tiedoston hakemisto]\vmcom.dll”.

3. Lisätään referenssi VMware VmCOM 1.0 Type Libraryyn

   Project -> Add Reference…



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

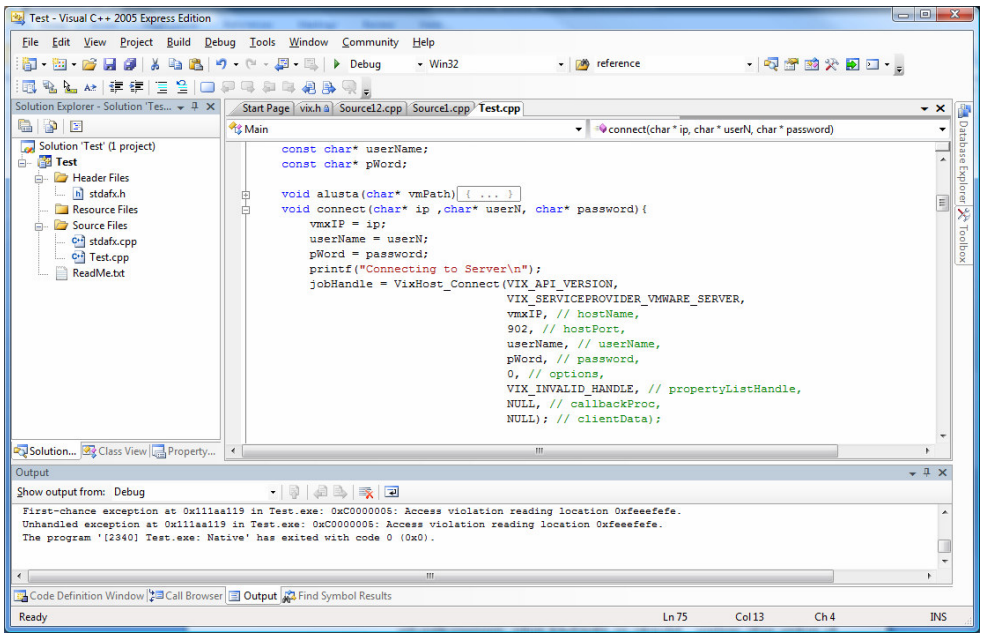
```
// Koodia
```

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

5.1 Ohjelmointiympäristöt

Kuva 18. Visual C# 2005 logo

Ohjelmointiympäristöksi valittiin Microsoftin Visual Studio 2005 Express Editionit. Express Editionit ovat riisuttuja malleja normaalista Visual Studio 2005:stä ja niitä voi koekäyttää 30 päivän ajan, jonka jälkeen käyttäjän tulee rekisteröidä ohjelma. Ohjelman rekisteröinti on ilmaista.



Kuva 19. Visual C++-ikkuna

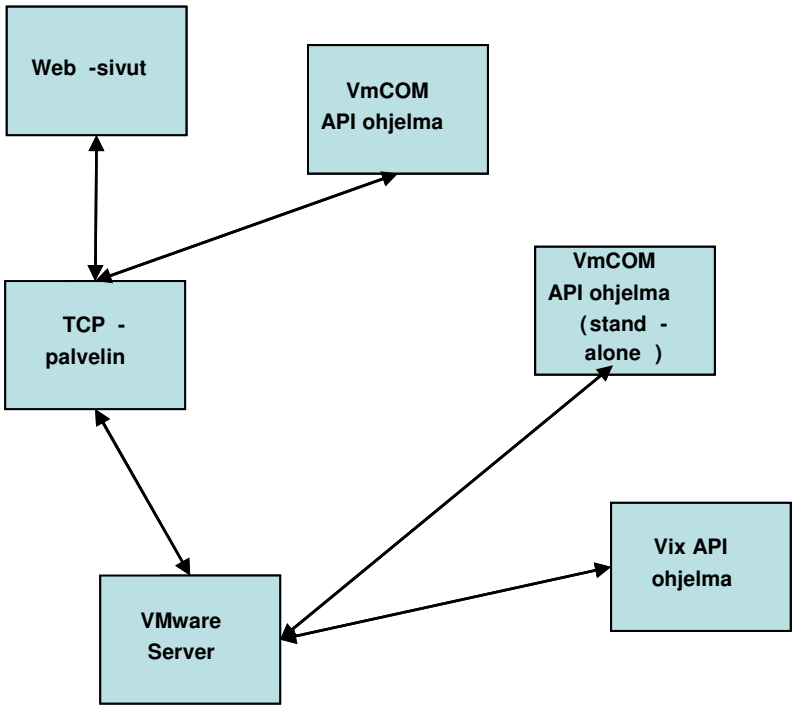
Vix–rajapintaa käyttävän ohjelman koodi luotiin Visual C++ 2005 Express Editionilla, Windows Forms –ohjelmat ja TCP-palvelimen koodit Visual C# 2005 Express Editionilla ja dynaamiset web-sivut Visual Web Developer

2005 Express Editioilla. Ohjelmat ovat saatavilla osoitteesta http://msdn.microsoft.com/vstudio/express/. Tietokanta luotiin Microsoft SQL Server 2005 –ohjelmalla.

ASP.NET sivuja varten on myös ladattava AJAX kehys (Framework) osoitteesta http://asp.net/ajax/.

5.2 Tavoitteet

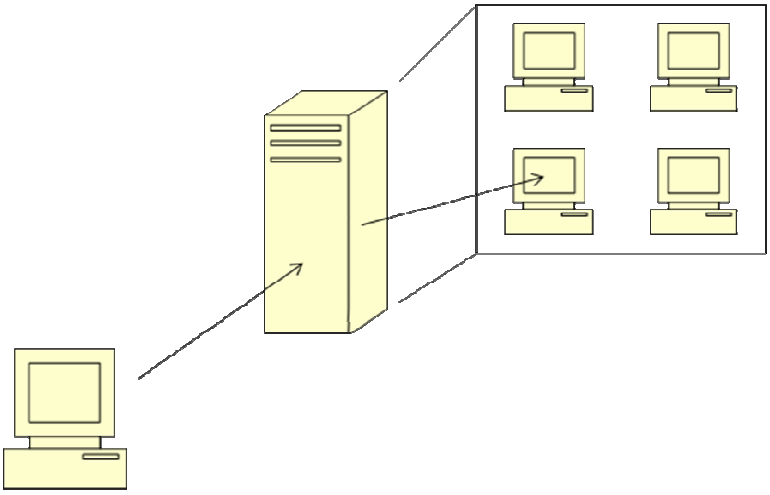
Tarkoituksena on luoda skriptejä, joilla voidaan hallita VMware Server ohjelmalla luotuja virtuaalikoneita sekä luoda normaalikäyttäjälle käyttöliittymiä, joiden avulla käyttäjä saa yhteyden virtuaalikoneisiinsa ja pystyy hallitsemaan niitä. Käyttöliittymä luodaan konsoli-ohjelmana Vix API:a käyttäen ja se kirjoitetaan C++-ohjelmointikielellä. Windows Forms-käyttöliittymät kirjoitetaan C#-ohjelmointikielellä, joista stand-alone-versio käyttää VmCOM API:a. Dynaamiset web-sivut tehdään käyttäen ASP.NET -teknologiaa. Palvelinpohjainen Windows Forms-käyttöliittymä ja web-käyttöliittymä tarvitsevat TCP-palvelinta, joka kirjoitetaan myös C#-kielellä, jolloin käytetään hyväksi VmCOM API:lla luotuja skriptejä.



Kuva 20. Ohjelmien väliset suhteet

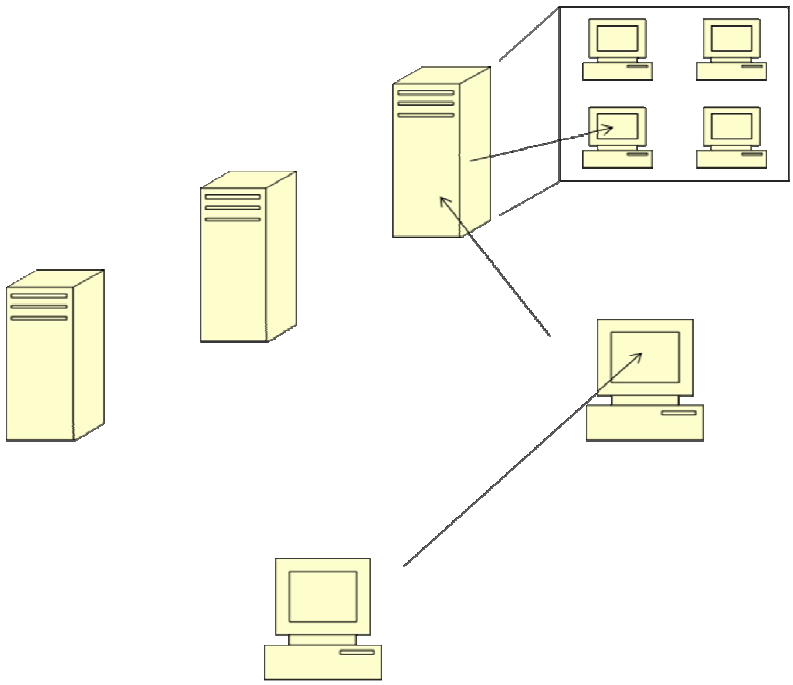
Tilanteessa jossa ohjelma on suorassa yhteydessä VMware Serveriin, käyttäjä tietää palvelinkoneen IP-osoitteen, sekä käyttäjällä on oltava

oikeudet päästä palvelinkoneen sisälle. Tiedot voidaan myös salata käyttäjältä ohjelman sisälle. Käyttäjän on myös tiedettävä omien virtuaalikoneidensa tiedot.



Kuva 21. Ilman välityspalvelinta

Tilanteessa jossa ohjelma käyttää TCP-palvelinta yhteydenhallintaan käyttäjän vastuu jää vähäisemmäksi. TCP-palvelimella on tallessa käyttäjien tiedot, virtuaalikoneet, sekä missä palvelimella käyttäjän virtuaalikoneet sijaitsevat. Käyttäjä tietää kirjautumissivun web-osoitteen ja kirjautuu sisään omilla tunnuksillaan. Dynaamiset sivut ottavat yhteyttä TCP-palvelimelle, joka hoitaa yhteydet palvelinkoneille. Windows Forms –ohjelmaa käytettäessä käyttäjän tulee tietää TCP-palavelimen IP-osoite.



Kuva 22. Palvelinkoneiden kanssa

TCP-palvelin hakee käyttäjien tiedot tietokannasta, joka on tallennettu Microsoft SQL Serverille. Tietokannassa on taulut käyttäjille (users), VMware Servereille (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:

ID	UserName	FirstName	LastName	Password	Server	Admin
1	samin	Sami	Niemelä	G4hxy	1	false
2	tiinap	Tiina	Penttilä	oi4Gjh	2	false
3	jussiv	Jussi	Vihavainen	9h7jfa	1	false
4	admin	Antti	Jussila	root		true

Servers-taulu:

ID	Name	IP	UserName	Password
1	U315/2	212.24.56.1	admin	root
2	A204/1	212.21.58.1	admin2	root2

Virtualmachines-taulu:

ID	Name	Folder	Owner

1	Ubuntu	c:\vm\ubuntu\ubuntu.vmx	3
2	FreeBSD	c:\vm\freebsd\freebsd.vmx	1
3	Vista	e:\vmachines\vista\vista.vmx	2

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

Operaatiot suoritetaan käyttämällä apuna handleja. Handleen tallentuu suoritettavan operaation tietoja. Handleja tarvitaan yleensä kolmea eri tyyppiä. JobHandle on yleinen työhandle kaikille suoritettaville operaatioille. HostHandle on VMware Serverin oma handle, jossa ovat tallessa yhteystiedot. VmHandle on virtuaalikoneen yhteyshandle.

```
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, userNamen 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()-funktiolla, 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
                    VIX_INVALID_HANDLE,
                     // propertyListHandle
                    NULL, // callbackProc
                    NULL); // clientData

// Odotetaan jobHandlen valmistumista
err = VixJob_Wait(jobHandle, // Työhandle
             VIX_PROPERTY_JOB_RESULT_HANDLE,
             &hostHandle, // Palvelinhandle
             VIX_PROPERTY_NONE);
```

5.3.3 Virtuaalikoneen rekisteröinti

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

```
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 myöhemmissä funktiokutsuissa. VixVM_Open() on myös asynkroninen funktio.

```
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.

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

VixVM_PowerOff()-funktio sammuttaa käyttöjärjestelmän. Funktio vastaa samaa, kuin tietokone sammutettaisiin virtakytkimestä. Jotkut virtuaalikoneen operaatiot, kuten virtuaalisen raudan konfiguroiminen, vaativat, että virtuaalikone on pois päältä.

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

VixVM_Suspend()-funktio vastaa samaa kuin kannettavien nukkumistila. Käyttöjärjestelmä lopettaa kaiken suorittamisen, mutta VixVM_PowerOn()-funktio palauttaa käyttöjärjestelmän samaan tilaan, missä se oli ennen VixVM_Suspend()-funktion kutsua. Jos virtuaalikone ei ole käynnissa, palautetaan virhe.

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

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

```
jobHandle = VixVM_Reset(vmHandle,
                 VIX_VMPOWEROP_NORMAL,
```

```
                NULL, // *callbackProc
                NULL); // *clientData
```

Funktiot ovat asynkronisia funktioita.

5.3.6 Handlen lopetuksen odottaminen

VixJob_Wait()-funktio palauttaa Vixin virheen (VixError). Funktiolle annetaan handle, jota odotetaan. Tässä tapauksessa se on aina työhandle. Funktiolle annetaan myös handle joka päivitetään. Päivitettävä handle on joko host-Handle tai vmHandle.

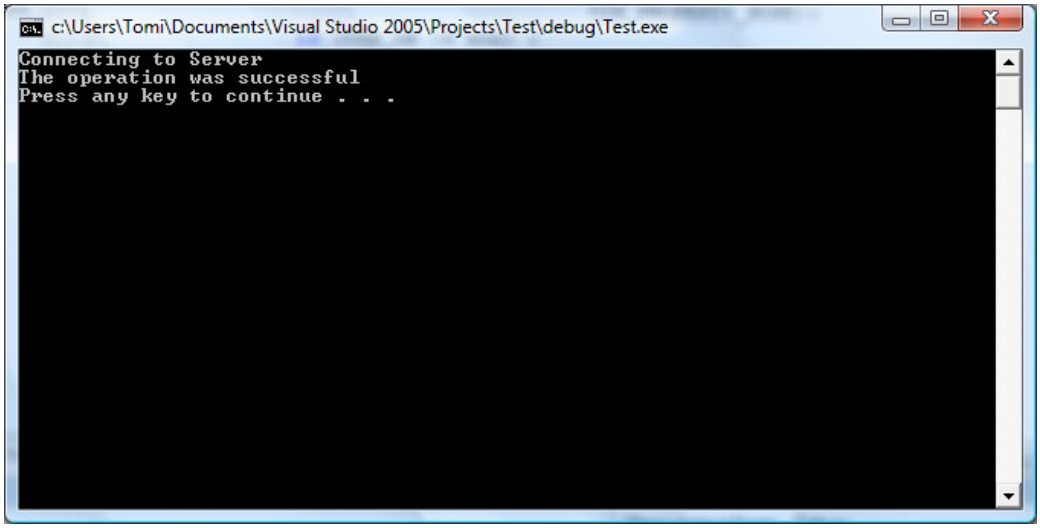
```
VixError err;

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

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

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

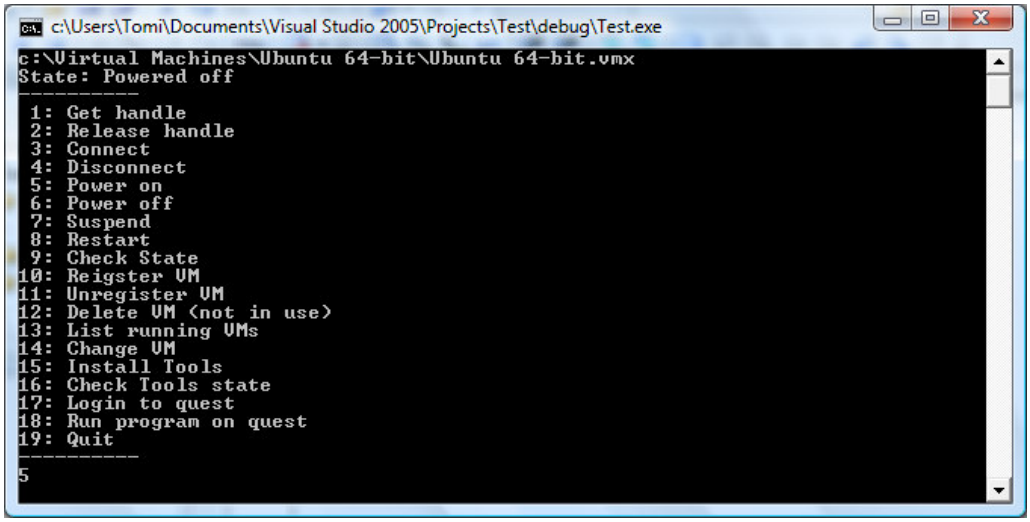
```
const char* errText;

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 suosituin 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 parametrina 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 virtuaalikoeen käynnistystä.

```
// 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)
   ```

4. Katkaisee yhteyden palvelimeen.

   ```
   void disconnect()
   ```

5. Käynnistää virtuaalikoneen.

   ```
   void powerOn()
   ```

6. Sammuttaa virtuaalikoneen.

   ```
   void powerOff()
   ```

7. Nukuttaa virtuaalikoneen.

```
void suspend()
```

8. Resetoi virtuaalikoneen.

```
void reset()
```

9. Tarkastaa virtuaalikoneen tilan.

```
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()
```

14. Vaihtaa virtuaalikonetta.

```
void alusta(char* vmPath)
```

Funktioiden suorittavat toiminnat ovat hyvin samanlaiset kuin aiemmin esitetyt Progarmming API:n skiriptit. 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

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

- VmServerCtl

Kontrolli VMware Serveriin. Sisältää kokelman Serverille rekisteröidyistä 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.

```
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{}
```

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{}
```

Mikäli toimenpide epäonnistuu, palautetaan virhe.

5.5.4 *Yhteydenotto virtuaalikoneseen*

Yhteydenotto vaatii, että virtuaalikone on rekisteröitynä 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{}
```

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) parametrina. Se määrittelee, kuinka kovasti VMware Server koittaa ajaa tahdotun metodin läpi. VmPowerOpModilla on kolme eri tilaa:

- Soft

```
VmPowerOpMode.vmPowerOpMode_Soft
```

- Hard

```
VmPowerOpMode.vmPowerOpMode_Hard
```

- TrySoft

```
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.

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

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

```
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 käynnissa, palautetaan virhe.

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

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

```
try
```

```
{
    vm.Reset(VmPowerOpMode);
}
catch{}
```

Mikäli toimenpide epäonnistuu, palautetaan virhe.

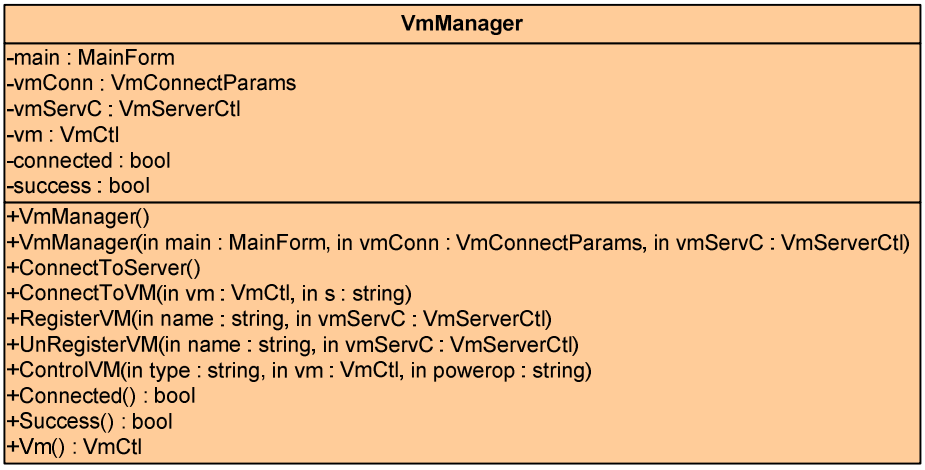
5.6 VmCOM API stand-alone ohjelma

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 lukee 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



Kuva 25. VmManagerin luokkakaavio

VmManager-luokka hoitaa kaiken yhteydenpidon ohjelman ja VMware Serverin välillä. Luokan konstruktorissa annetaan luokalle yhteysparametrit (VmConnectParams), palvelinkontrolli (VmServerCtl) sekä viittaus pääikkunaan (MainForm). Luokka sisältää metodit ConnectToServer(),

ConnectToVM(), ControlVM(), Register() ja Unregister(). Metodien suoriutuessa loppuun ne kutsuvat pääikkunan delegaatteja.

```
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.

```
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.

```
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);
}
```

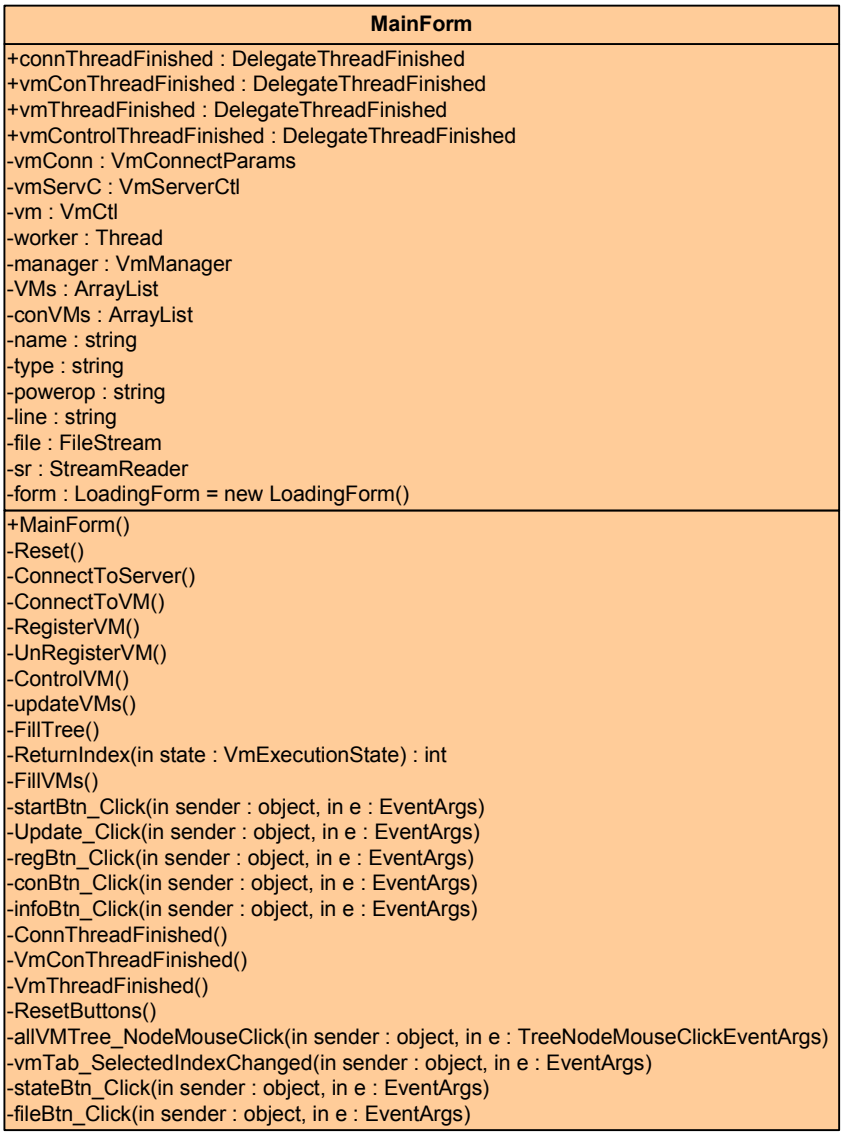
ControVM()-metodilla voidaan käynnistaa, sammuttaa, nukuttaa tai palauttaa virtuaalikone. Metodi valitsee oikean voimamoden ja suoritettavan toiminnon annettujen stringien avulla.

```
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);
}
```

VmManager-luokka sisältää myös public propertyt privaatti boolean succesille ja connectedille. Näiden avulla pääohjelma tarkistaa, onnistuiko kutsuttu metodi tehtävässään.

5.6.2 MainForm-luokka



Kuva 26. MainForm:in luokkakaavio

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

1. Luodaan uusi säie:

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

2. Säikeen sisällä kutustaan VmManager luokkaa:

```
public void ControlVM()
{
```

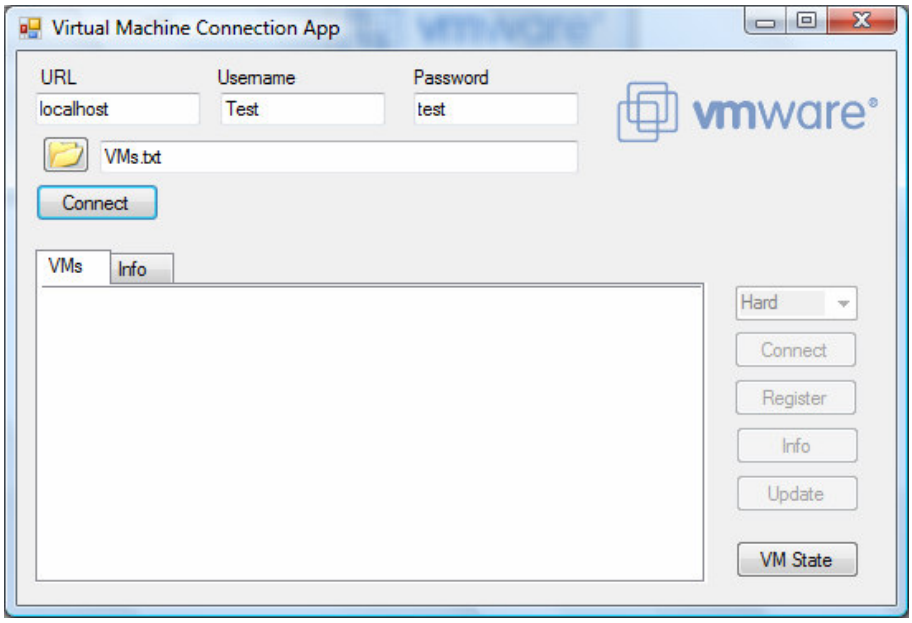
```
        manager.ControlVM(type, vm, powerop);
    }
```

3. VmManagerin metodi kutsuu MainFormin delegaattia:

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

4. Suoritetaan metodi johon delegaatti ohjaa:

```
vmThreadFinished = new
DelegateThreadFinished(this.VmThreadFinished);

private void VmThreadFinished()
{
    // koodia
    if (manager.Success)
    {
        updateVMs();
    }
    else
      MessageBox.Show("Failed");
}
```



Kuva 27. Käynnistysnäkymä

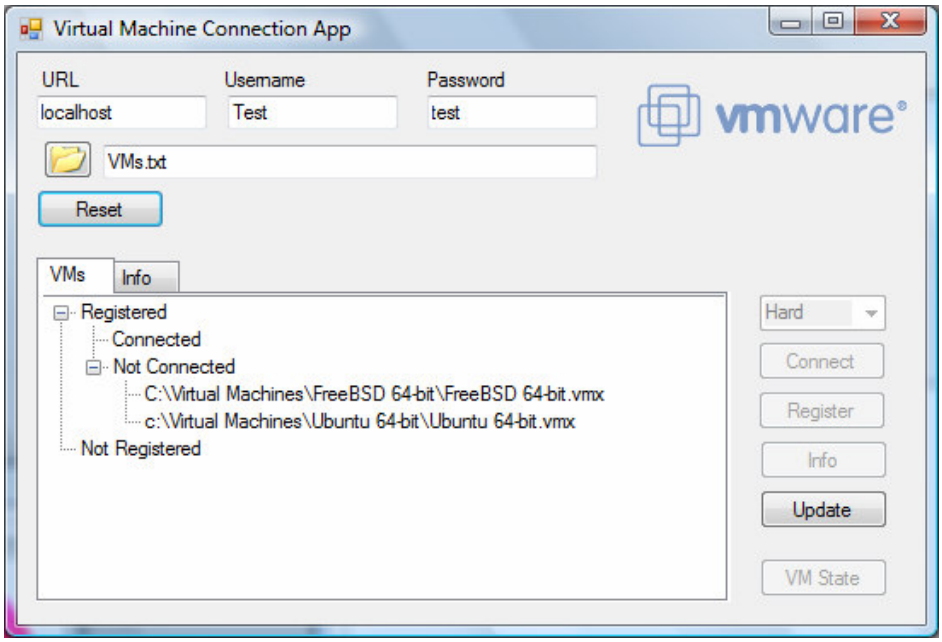
Ohjelman käynnistyessä (kuva 27) näkyvissä ei ole yhtään virtuaalikonetta. Käyttäjä syöttää teksilaatikoihin tiedot ja valitsee kansio-painikkeesta oikean tekstitiedoston. Connect-nappia painamalla ohjelma luo uuden säikeen joka kutsuu ConnectToServer()-metodia.

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

ConnectToServer()-metodi luo uuden VmManagerin ja kutsuu managerin ConnectToServer()-metodia.

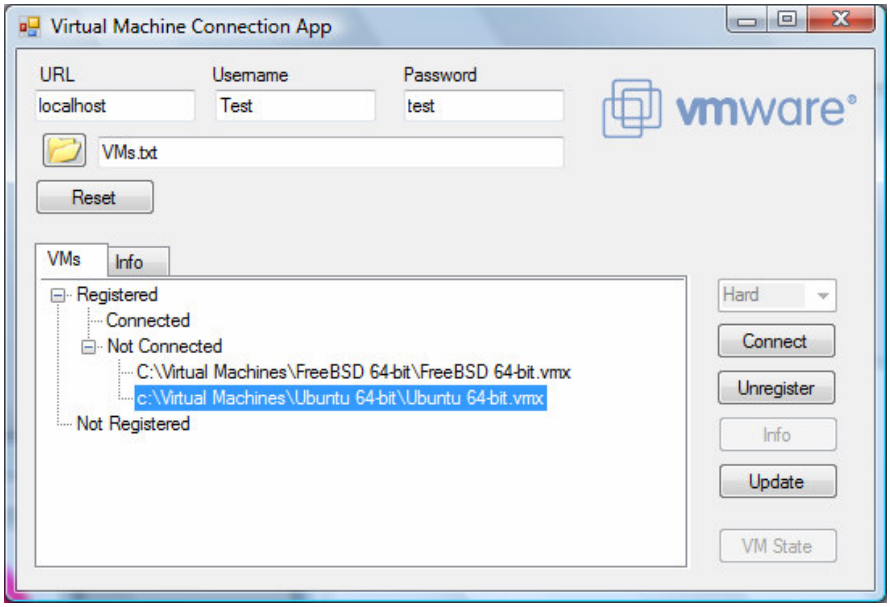
```
manager = new VmManager(this, vmConn, vmServC);
manager.ConnectToServer();
```

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



Kuva 28. Yhdistettynä VMware Serveriin

Kun käyttäjä valitsee listasta haluamansa virtuaalikoneen, ohjelma päivittää yhteysnapit vastaamaan virtuaalikoneelle olevia toimenpiteitä. Kuvassa 29 käyttäjällä on valittuna rekisteröity virtuaalikone, jolloin käyttäjä voi joko ottaa yhteyden virtuaalikoneeseen tai poistaa virtuaalikoneen hakemistosta. Update-nappi päivittää virtuaalikonelistauksen.



Kuva 29. Virtuaalikone valittuna

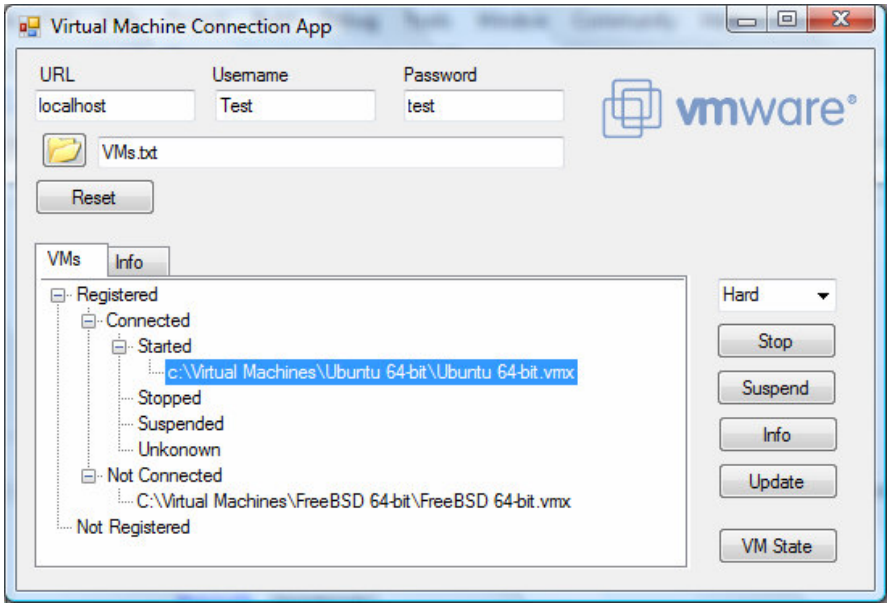
Kun käyttäjällä on virtuaalikone valittuna ja hän painaa Connect-nappia, luodaan uusi säie.

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

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

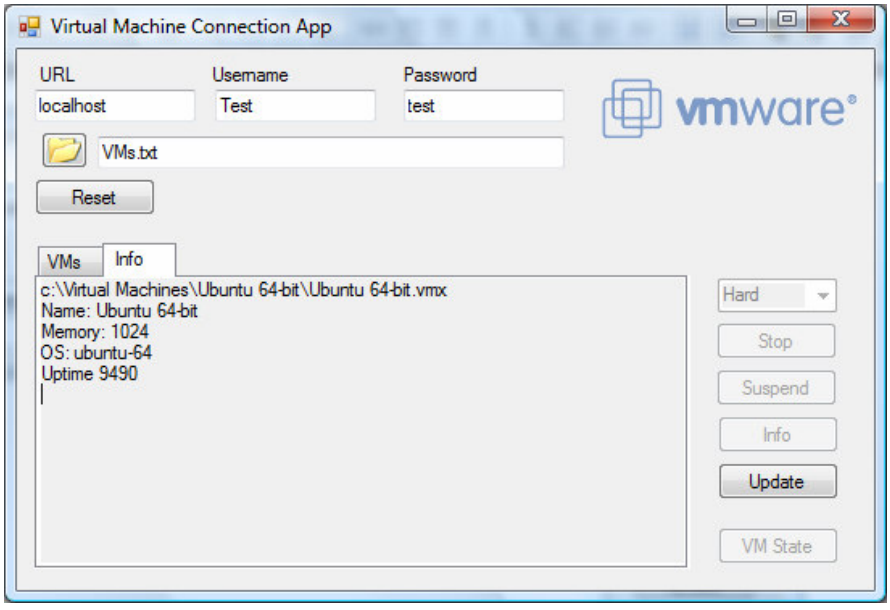
```
manager.ConnectToVM(vm, name);
```

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



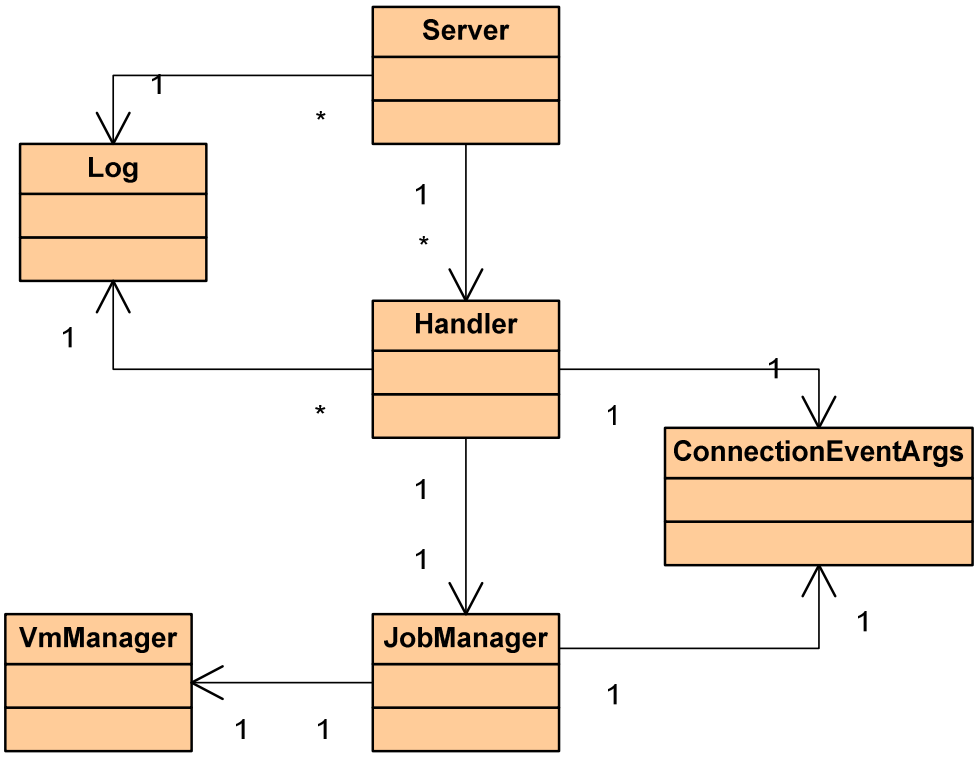
Kuva 30. Valittu virtuaalikone käynnistettynä

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



Kuva 31. Valitun virtuaalikoneen lisäinfo

5.7 TCP-palvelin



Kuva 32. TCP-palvelimen luokkakaavio

Palvelin tarjoaa yhteydanhallintapalveluita asiakasohjelmille. Palvelin ei välitä asiakasohjelman tyypistä, 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 kokelmat. Jokaiselle käyttäjälle luodaan oma Handler-luokka, joka hoitaa käyttäjän pyynnöt ja pitää huolta käyttäjän omista oliokokoelmista. Jokaisella käyttäjällä on myös omat JobManager- ja VmManager-luokat, joissa tapahtuu virtuaalikoneiden hallinta. Palvelimen toiminta on säikeistetty, jolloin se pystyy käsittelemään monien käyttäjien toimintoja ”samanaikaisesti”.

Ohjelma lukee asetuksensa Settings.xml tiedostosta, jonka täytyy sijaita samassa hakemistossa, jossa palvelimen käynnistystiedosto sijaitsee. Ohjelma luo yhteys stringin joko Trusted Connectionina tai Standard Securityna.

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

```
<settings>
  <port>5050</port>
  <connectionstring type="trusted">
    <server>IP547</server>
    <database>VMware</database>
    <trusted>true</trusted>
  </connectionstring>
<!--<connection type="standard">
  <server>ttest</server>
  <database>CPMDEMO8</database>
  <uid>sa</uid>
  <password>admin</password>
  </connection>-->
</settings>
```

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 kuvauksen sekä virtuaalikonelistauksen tai virtuaalikoneen infon XML-muodossa.

Esimerkki virtuaalikonelistasta.

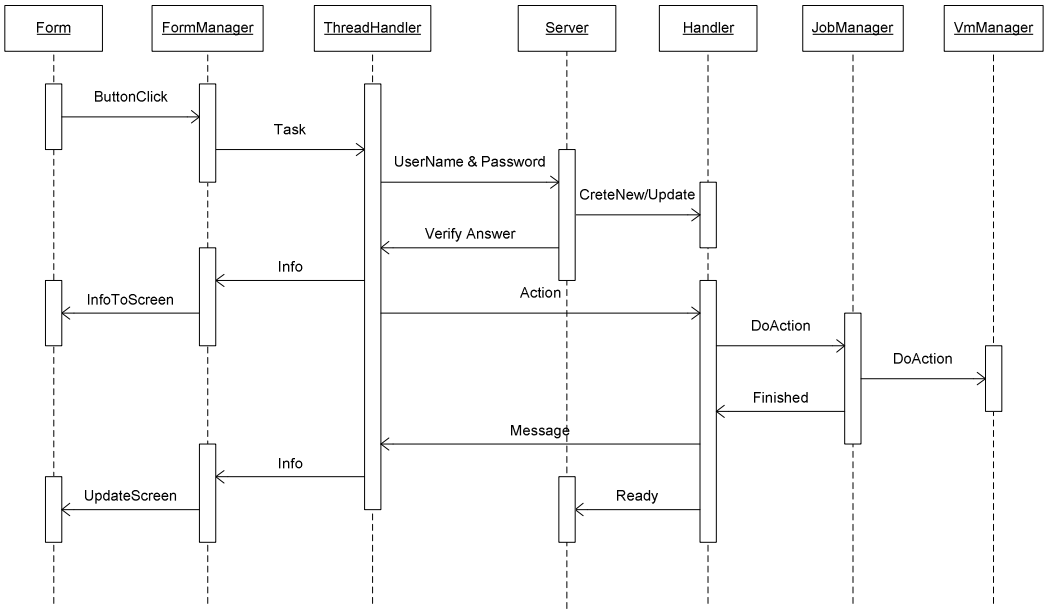
```
<?xml version="1.0"?>

<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 version="1.0"?>

<Info>
 <Name>
```

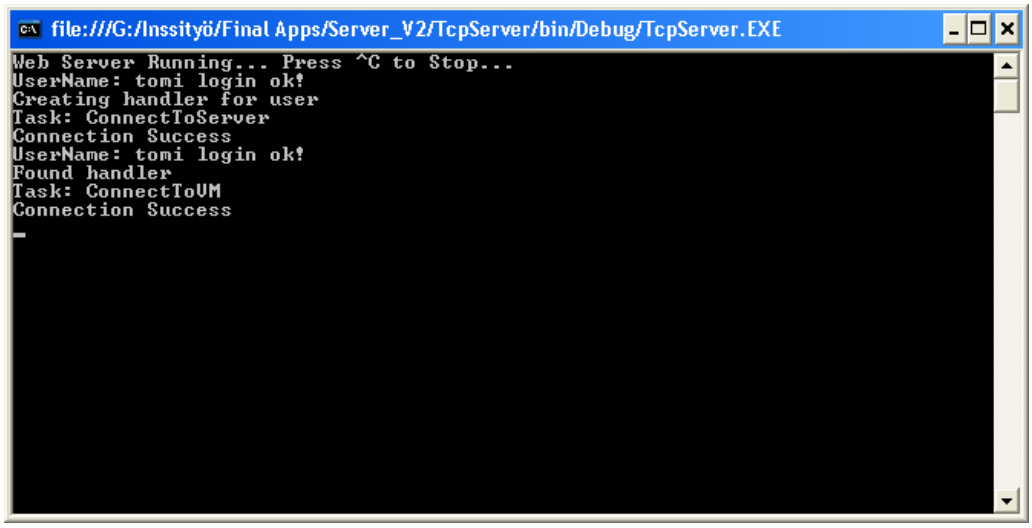
```
  FreeBSD 64-bit
 </Name>
 <Memory>
  512
 </Memory>
 <OS>
  freebsd-64
 </OS>
 <Uptime>
  0
 </Uptime>
</Info>
```

Sekvenssikaaviossa on esitetty luokkien välinen kommunikaatio client-ohjelman lähettäessä pyyntö palvelimelle.



Kuva 33. Sekvenssikaavio

Palvelin kirjoittaa toimiessaan 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.

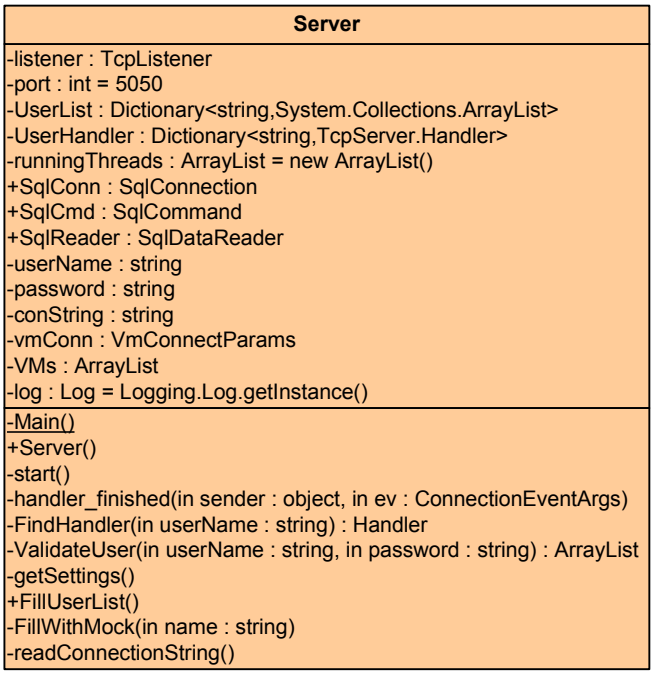


Kuva 34. Toimintojen kirjautuminen konsolin ikkunaan

Samaan aikaan lokiin 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



Kuva 35. Luokkakaavio

Käynnistyessään ohjelma lukee asetukset Settings.xml tiedostosta.

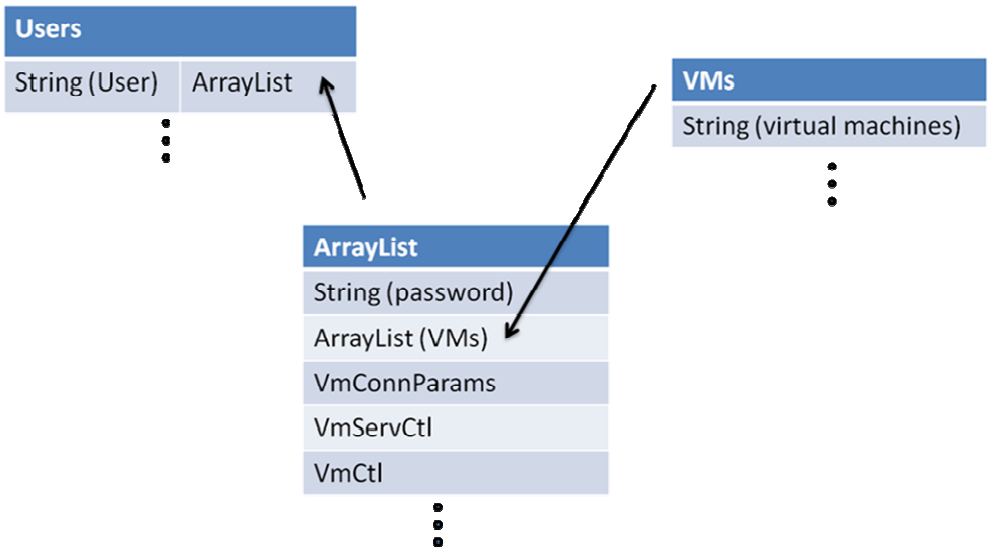
```
private void getSettings()
{
   XmlTextReader textReader = new
   XmlTextReader("Settings.xml");
   XmlDocument doc = new XmlDocument();
   doc.Load(textReader);

   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", ""))
     {
        case ("trusted"):
          conString = "Server=" +
          node.Current.SelectSingleNode("server").Value + ";";
          conString += "Database=" +
          node.Current.SelectSingleNode("database").Value +
          ";";
          conString += "Trusted_Connection=" +
          node.Current.SelectSingleNode("trusted").Value + ";";
          break;
       case ("standard"):
```

```
            // Koodia
        }
    }
}
```

Server-luokka luo kokoelman käyttäjille (kuva 36).

```
Dictionary<string, ArrayList> UserList;
```

Kokelmana käytetään Dictionarya, jossa käyttäjän käyttäjätunnus on avaimena ja arvona ArrayList-kokoelma. Kun ohjelma käynnistetään otetaan yhteys tietokantaan ja päivitetään kokoelma. Jokaisen käyttäjän salasana talletetaan kokoelmaan, luodaan VMs-kokoelma tietokannassa oleville virtuaalikoneille, talletetaan yhteysparametrit palvelimelle VmConnectParams-luokkaan, sekä talletetaan palvelinkontrolli VmServCtl. Virtuaalikonekontrolleja (VmCtl) lisätään kokoelmaan sitämukaan, kun käyttäjä ottaa yhteyksiä virtuaalikoneisiin.



Kuva 36. Käyttäjäkokoelman rakenne

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

```
string cmd = "Select * FROM Users";
SqlCmd = new SqlCommand(cmd, SqlConn);

SqlReader = SqlCmd.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.

```
foreach (string s in UserList.Keys)
{
     SqlCmd.CommandText = "Select * FROM VirtualMachines,
     Users WHERE VirtualMachines.Owner = Users.ID AND
     USers.UserName = '" + s + "'";

     SqlReader = SqlCmd.ExecuteReader();
     VMs = new ArrayList();

     while (SqlReader.Read())
     {
          string name = SqlReader["Folder"].ToString();
          VMs.Add(name);
     }

     UserList[s].Add(VMs);
     SqlReader.Close();
}
```

3. Käydään foreach-silmukalla jokainen käyttäjä läpi ja päivitetään yhteysparametrit.

```
foreach (string s in UserList.Keys)
{
     vmConn = new VmConnectParams();

     SqlCmd.CommandText = "Select * FROM Servers, Users
     WHERE Servers.ID = USers.Server AND Users.UserName='" +
     s + "'";

     SqlReader = SqlCmd.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());
     SqlReader.Close();
}
```

Virtuaalikoineta 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.

```
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 asettuu kuuntelemaan sockettia ja luo tietovirralle lukijan ja kirjoittajan.

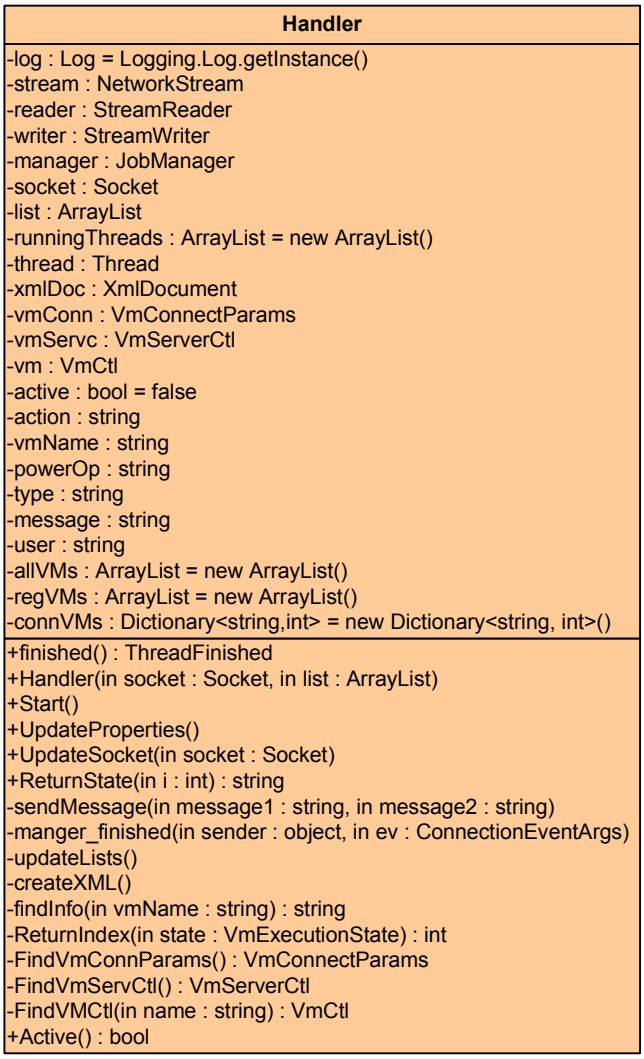
```
Socket socket = listener.AcceptSocket();

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

Ohjelma kuuntelee käyttäjän käyttäjätunnuksen ja salasanan ja vertaa niitä kokoelmassa oleviin tunnuksiin. Mikäli vastaavat löytyvät, ohjelma katsoo löytyykö käyttäjälle valmiiksi Handler-luokka, mikäli ei, luodaan uusi. Jos Handler-luokka löytyy, päivitetään Handler-luokan socket vastaamaan uutta sockettia. Lopuksi Handler-luokkan Start()-metodi käynnistetään omassa säikeessa ja TCP-palvelin palaa takaisin kuuntelemaan uutta sockettia.

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

Säikeet talletetaan 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



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 yhteysparametrit (VmConnectParams) ja palvelinkontrolli (VmServerCtl), jotka välitetään JobManagerille sen konstruktorissa.

```
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);
}
```

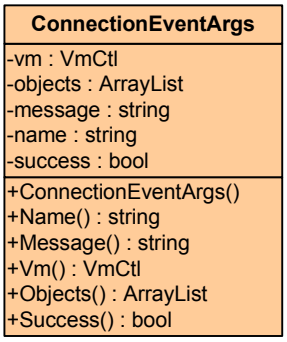
Tietovirrat päivitetään aina, kun käyttäjä ottaa uudestaan yhteyden palvelimeen.

```
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 ConnectionEventArg-luokan sisään.

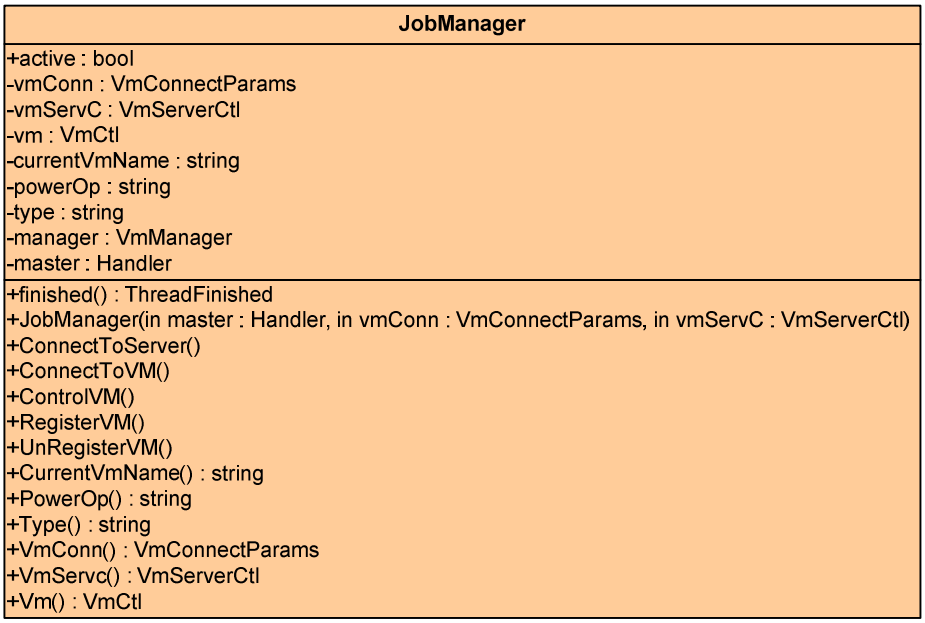


Kuva 38. ConnectionEventArgs- luokkakaavio

Delegaatti ohjautuu manager_finished()-metodiin, jolloin poistetaan säie runningThreads-kokelmasta ja päivitetään luokan omat kokoelmat. Ohjelma käy läpi käyttäjän kaikki virtuaalikoneet, vertaa niitä VMware Serverillä oleviin rekisteröityihin koneisiin ja valitsee niistä vain käyttäjän omistamat virtuaalikoneet, sillä yhdellä VMware Serverillä saattaa olla monen käyttäjän virtuaalikoneita. Handler luo omista luokistaan virtuaalikonelistan XML-muotoon ja lähettää sen client-ohjelmalle.

```
private void manger_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



Kuva 39. Luokkakaavio

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

```
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);
}
```

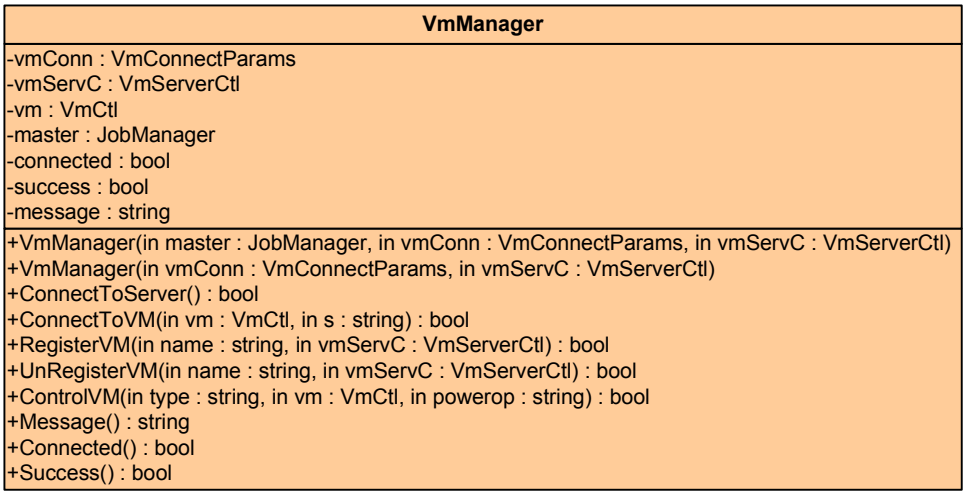
JobManagerin yhteydenhallintametodit luovat yhteydenhallinta-argumentin, jossa välitetään booleanina onnistuminen, yhteydenhallintaviesti sekä mahdollisesti myös uuden virtuaalikoneen kontrolliolio, mikäli kyseessä on uusi yhteys virtuaalikoneeseen.

```
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 pyyntöjä samanaikaisesti VMware Serverille.

5.7.4 VmManager-luokka



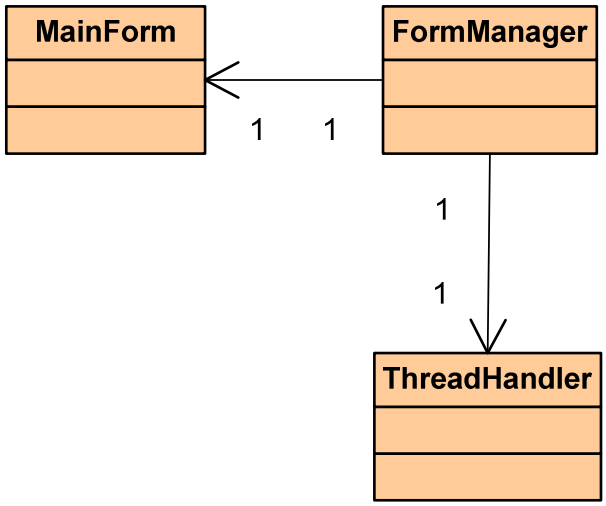
Kuva 40. Luokkakaavio

VmManager-luokka toimii hiukan erilailla kuin stand-alone-ohjelman VmManager-luokka johtuen ohjelmien erilaisesta luonteesta. Valimistuessaan metodit eivät kutsu delegaattia Invoke()-metodilla, vaan palauttavat booleanin arvon. Täten metodit eivät ole muotoa void, vaan bool. Luokaan on myös lisätty property Message, johon talletetaan viimeisimmän toimenpiteen viesti. Alla esimerkkinä ConnectToServer()-metodi.

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

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

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

        return success;
    }
```

5.8 VmCOM API palvelinpohjainen ohjelma



Kuva 41. Ohjelman luokkakaavio

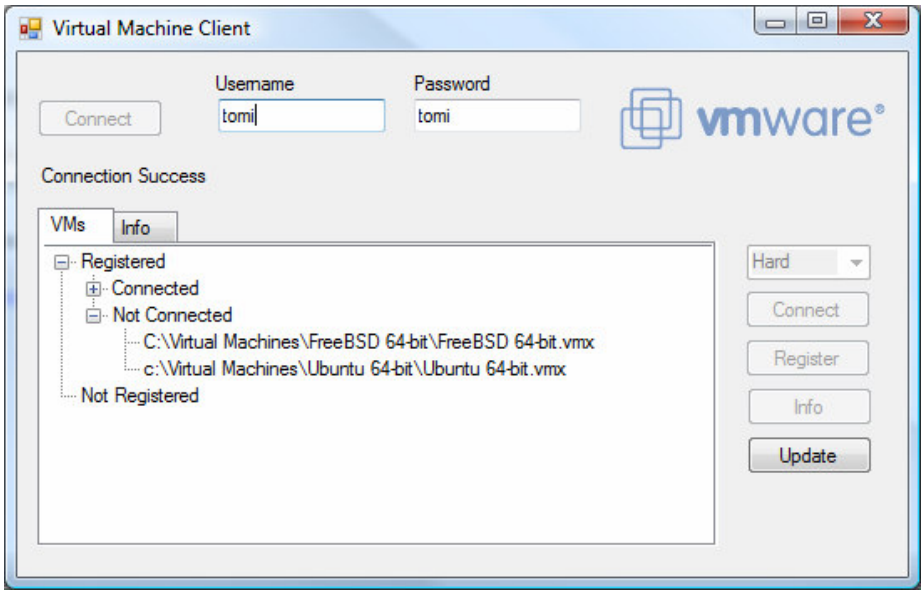
Client-ohjelmalla ei ole enää yhteyttä VMware Serveriin, vaan client lähettää pyynnöt TCP-palvelimelle, joka on yhteydessä VMware Serveriin. Client-ohjelmalla ei ole enää hallussa kontrolli- ja virtuaalikoneolioita. Se toimii vain näyttönä, joka näyttää käyttäjälle tämänhetkisen tilanteen.

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 version="1.0" encoding="utf-8" ?>

<settings>
 <ip>192.168.181.1</ip>
 <port>5050</port>
</settings>
```

TCP-palvelimella on tietokannassa käyttäjälle määrätyn VMware Serverin IP-osoite, salasana sekä käyttäjän virtuaalikoneet, joten käyttäjän ei tarvitse itse huolehtia kyseisistä tiedoista.

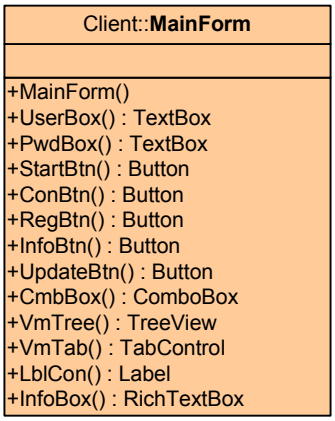
Ulkoisesti käyttöliittymä (kuva 42) on stand-alone -ohjelman kanssa lähes identtinen. Vain virtuaalikonetiedoston haku puuttuu.



Kuva 42. Clientin käyttöliittymä

Ohjelma on toteutettu MVC (Model-View-Controller) -arkkitehtuuria mukaillen, 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

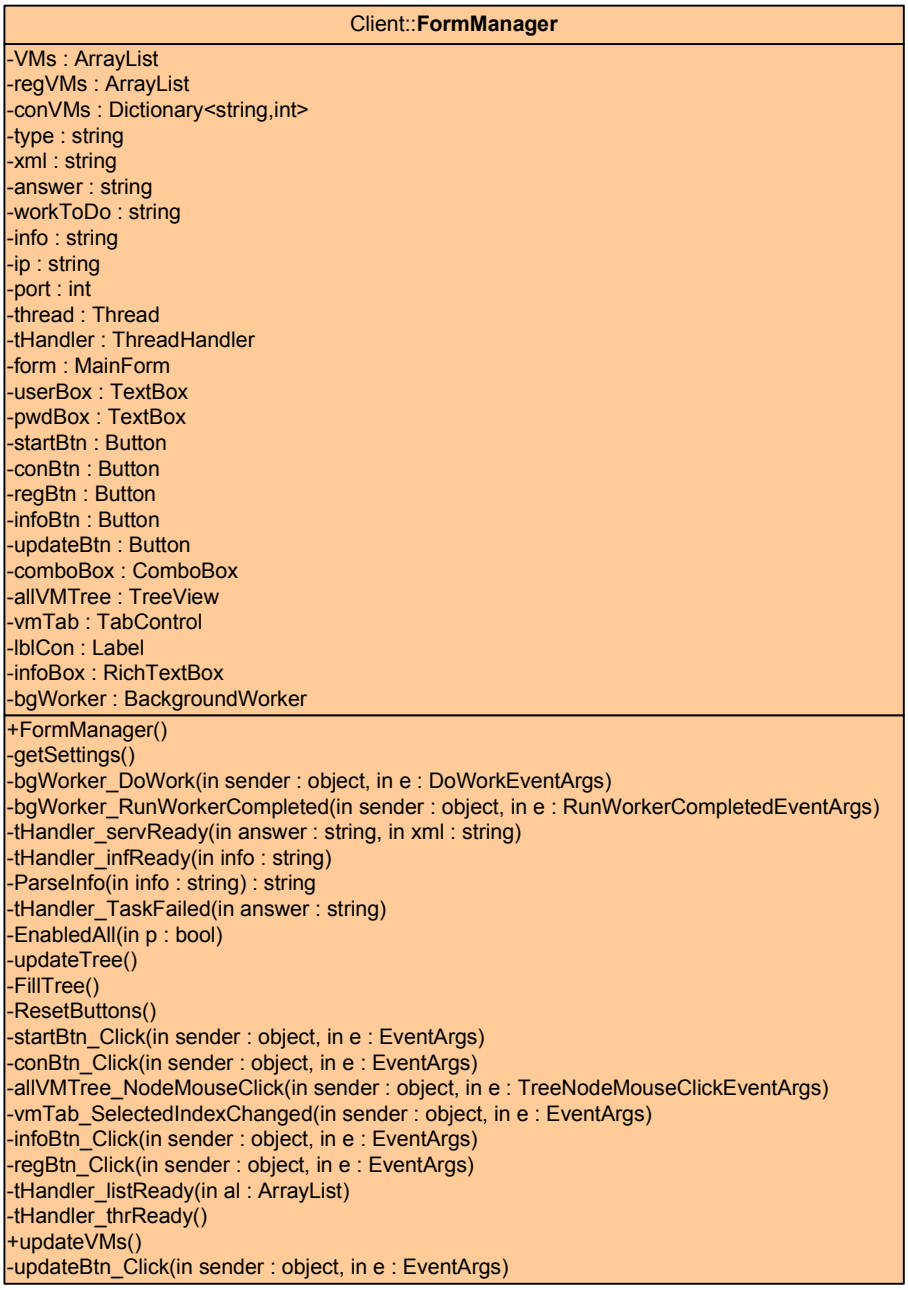


Kuva 43. Luokkakavio

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

```
public Button StartBtn
{
   get { return startBtn; }
   set { startBtn = value; }
}
```

5.8.2 FormManager-luokka



Kuva 44. Luokkakaavio

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

```
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();
}
```

Konstruktorissa luodaan myös Background Worker, joka hallitsee .Net -Frameworkissa säikeistyksen. Konstruktorissa haetaan myös annetut asetukset 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 ip
    node = nav.Select("settings/ip");
    while (node.MoveNext())
    { ip = node.Current.Value; }
}
```

Ohjelma ottaa yhteyden TCP-palvelimeen, kun painetaan Connect-nappia. Samalla ohjelma luo uuden ThreadHandlerin ja asettuu kuuntelemaan sen delegaatteja. ConnectButton-, RegisterButton- ja InfoButton-kontrollien enabled-tilaksi asetetaan false EnabledAll() -metodilla. StartButton-kontrollin enabled-tilaksi asetetaan erikseen false, sillä sen tilaa ei tulla enää ohjelman suorituksen aikana muuttamaan. Yhteydenotto aloitetaan käynnistämällä Background Worker ja antamalla sille käskyksi ConnectToServer.

```
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 käskyään vastaavan metodin 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;
    }
}
```

Säie 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.

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

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

```
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 perussolmut ja käy läpi virtuaalikone XML-tiedoston. Tiedostosta ohjelma hakee virtuaalikoneen nimen, sekä tilan ja tilan mukaan asettaa virtuaalikoneen TreeView-kontrollin oikeaan solmuun.

```
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);

   nav = 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].Expand();
            allVMTree.Nodes[0].Nodes[0].Nodes[0].Expand();
            break;
          case ("Stopped"):
            // Koodia
          default:
            break;
      }
   }
}
```

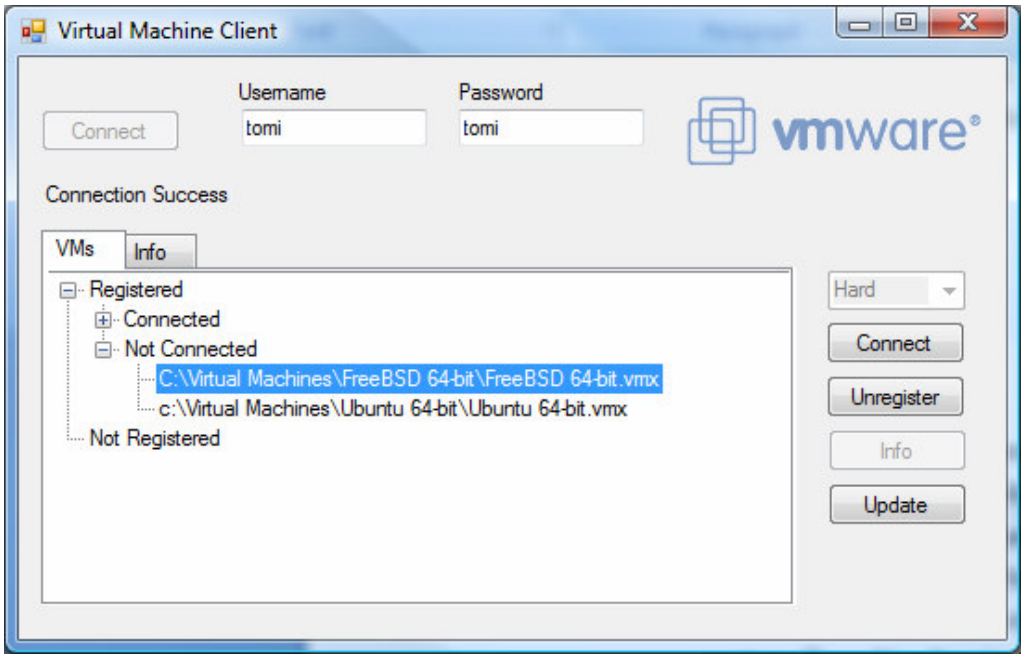
ResetButtons()-metodi asettaa kaikkien näytön Button-kontrollien enable-tilaksi falsen.

```
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

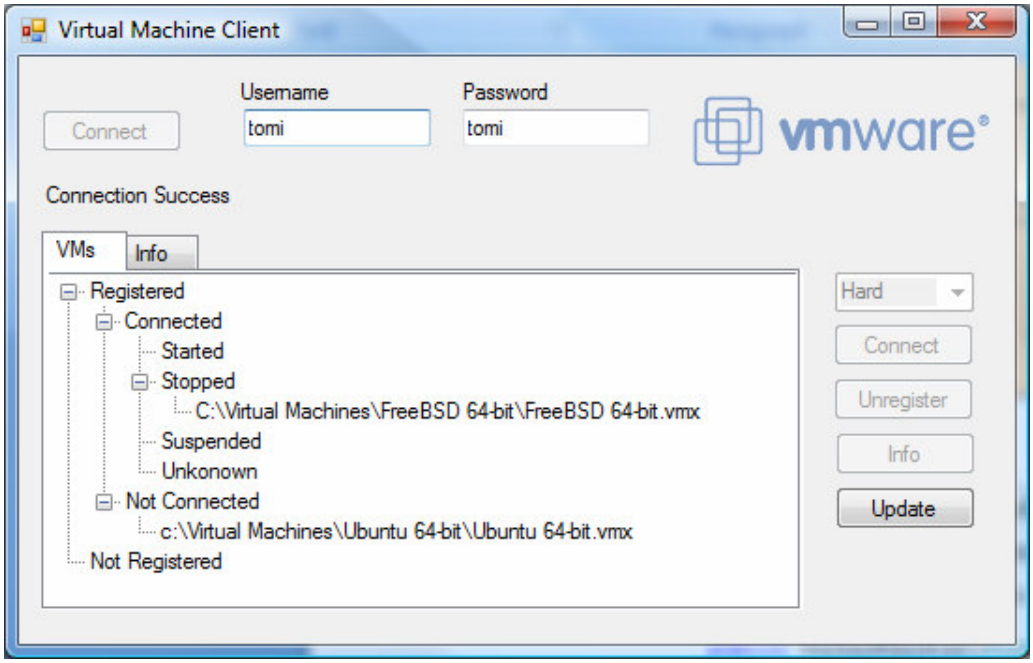
Jokaisen näppäimen Tag-attribuuttin on tallennettu valitun virtuaalikoneen tämänhetkinen tila, koska yhdellä Button-kontrollilla voi olla eri tehtävä riippuen valitun virtuaalikoneen tilasta. Eri tilat on tallennettu States-enumeratoriin. Enumraatiotyyppien hyöty on nopeampi käsittely verrattuna normaalin string-tietotyypin vertailuun keskenään, lisäksi koodista tulee yleensä enemmän itseään dokumentoiva enumeraatiotyyppien avulla.

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

```
    NotRegistered
}
```

Button–kontrollia painettaessa vaihdetaan kontrollien enabled-tiloiksi false, jotta ohjelman hakiessa tietoa käyttäjä ei voi käskea ohjelmaa hakemaan uusia tietoja. Ohjelma tarkastaa Button–kontrolliin talletetun State enumeroidun tyypin ja suorittaa oikeat toimenpiteet. Alla on esitetty Connect-buttonin painallus. Mikäli virutaalikoneen tila on NotConnected, ohjelma asettaa ThreadHandler-luokalle käsiteltäväksi virtuaalikoneeksi valitun solmun virtuaalikoneen ja käynnistää Background Workerin ConnectToVM-käskyllä. Mikäli virtuaalikoneen tila ei ole NotConnected, metodi käy läpi muut mahdolliset tilat ja asettaa ThreadHandlerin tarvitsemat parametrit ja käynnistää background workerin ControlVM-käskyllä.

```
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);
    }
}
```



Kuva 46. FreeBSD-virtuaalikone yhdistettynä

5.8.3 ThreadHandler-luokka

Client::**ThreadHandler**
-client : TcpClient -stream : NetworkStream -write : StreamWriter -reader : StreamReader -answer : string -xml : string -userName : string -password : string -ip : string -port : int -VMs : ArrayList -regVMs : ArrayList -conVMs : Dictionary<string,int> -main : FormManager +vmName : string +powerOp : string +type : string
+TaskReady() : TaskReadyHandler +TaskFailed() : TaskFailedHandler +InfoReady() : InfoReadyHandler +ListReady() : DelegateList +ThreadReady() : DelegateThreadFinished +ThreadHandler(in main : FormManager, in userName : string, in password : string, in ip : string, in port : int) -Connect() +ConnectToServer() +ConnectToVM() +ControlVM() +Info() +Register() +UnRegister() +ListVMs()

Kuva 47. Luokkakaavio

ThreadHandler hoitaa yhteydet TCP-palvelimeen. Handlerin konstruktorissa annetaan viittaus FormManageriin sekä annetaan ohjelman tarvitsevat tiedoit, joilla saadaan yhteys TCP-palvelimeen.

```
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;
}
```

ThreadHandlerin käyttää ThreadReady-, ThreadFailed- ja InfoReady-eventteja. ThreadReady-eventilla välitetaan vastaus, sekä virtuaalikone XML-lista, ThreadFailed-eventilla vain vastaus ja InfoReady-eventilla vain xml.

```
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.

```
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();
}
```

Esimerkissä otetaan yhteyttä palvelimeen. Kun TCP-palvelimelle on lähetetty käyttäjätunnus ja salasana, ohjelma odottaa vastausta. Mikäli vastaus on ”ok”, ohjelma lähettää yhteyspyynnön, tässä ”ConnectToServer”, ja jää odottamaan vastauksia. Mikäli yhteyttä palvelimeen ei saada, laukaistaan TaskFailed-event, mutta mikäli yhteys on onnistunut, laukaistaan TaskReady-event. TaskReady-event laukaistaan myös, vaikka palvelin ei

olisi hyväksynyt sisäänkirjautumista, mutta vastuksena näkyy silloin virheilmoitus.

```
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 ConnecToServer()-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).

```
public void ControlVM()
{
   try
   {
     Connect();

     string ok = reader.ReadLine();
     if (ok == "ok")
     {
       write.WriteLine("ControlVM");
       write.Flush();
       write.WriteLine(vmName);
       write.Flush();
       write.WriteLine(powerOp);
       write.Flush();
       write.WriteLine(type);
       write.Flush();
       answer = reader.ReadLine();
```

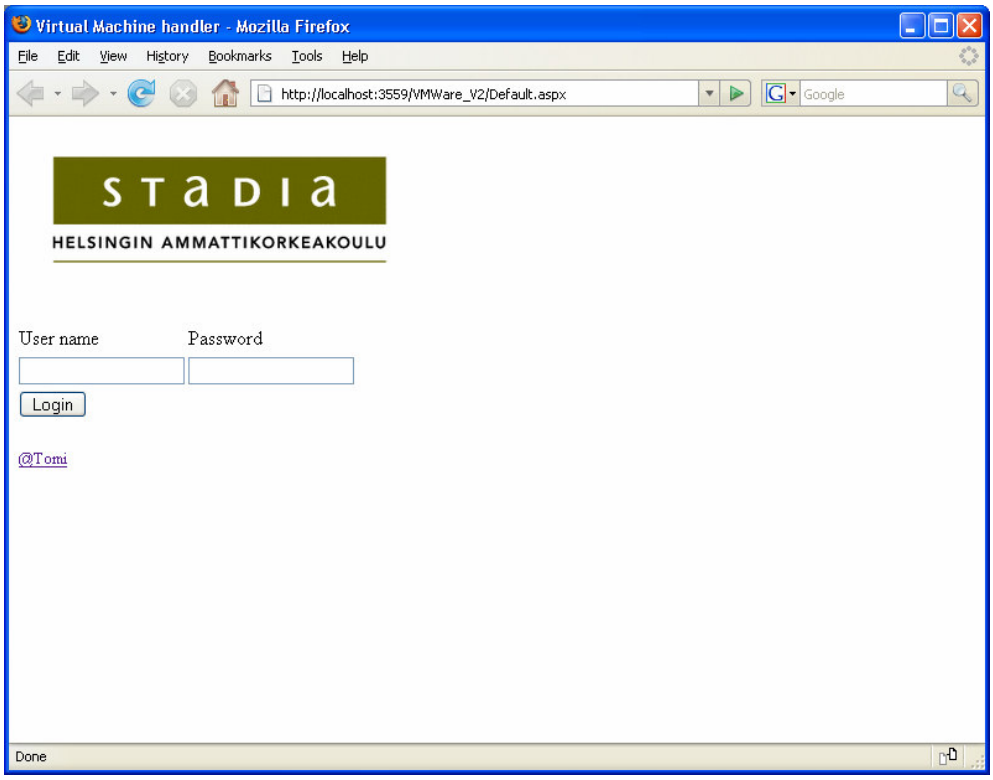
```
            xml = reader.ReadLine();
        }
        // Koodia
}
```

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

5.9 ASP.NET Dynaamiset web-sivut

ASP.NET-sivut käyttävät täysin samaa TCP-palvelinta kuin Windows Forms-pohjainen sovellus. TCP-palvelin ei huomaa eroa asiakkaiden välillä. Web-sivut on toteutettu Code-Behind Modelilla, jolloin näkymä on erotettu ohjelman koodista. Tällöin näkymä on xxx.aspx-tiedostossa ja sivuun liittyvä koodi on xxx.aspx.cs-tiedostossa. Web-sivuihin tehtiin kaksi näkymää, käyttäjä- ja admin-näkymä. Käyttäjän näkymästä koitettiin tehdä mahdollisimman samankaltainen kuin Forms-pohjainen sovellus. Admin-näkymän päätarkoitus on helpottaa tietokannanhallintaa.

5.9.1 Login page (Default.aspx)



Kuva 48. Kirjautumissivu

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.

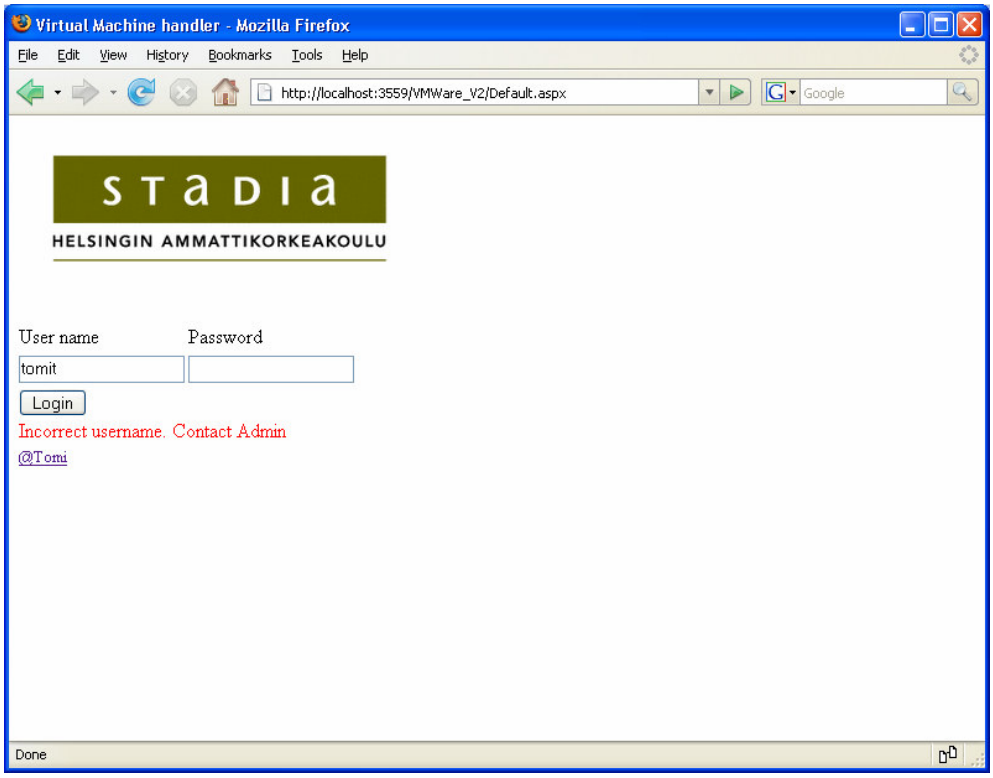
```
SqlConn = new SqlConnection(conString);
SqlConn.Open();

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

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

Saatua salasanaa verrataan käyttäjän syöttämään salasanaan, sekä talletetaan käyttäjän oikeus admin-booleaniin.

```
string password = SqlReader["Password"].ToString().Trim();

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

```
    return true;
}
```

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.



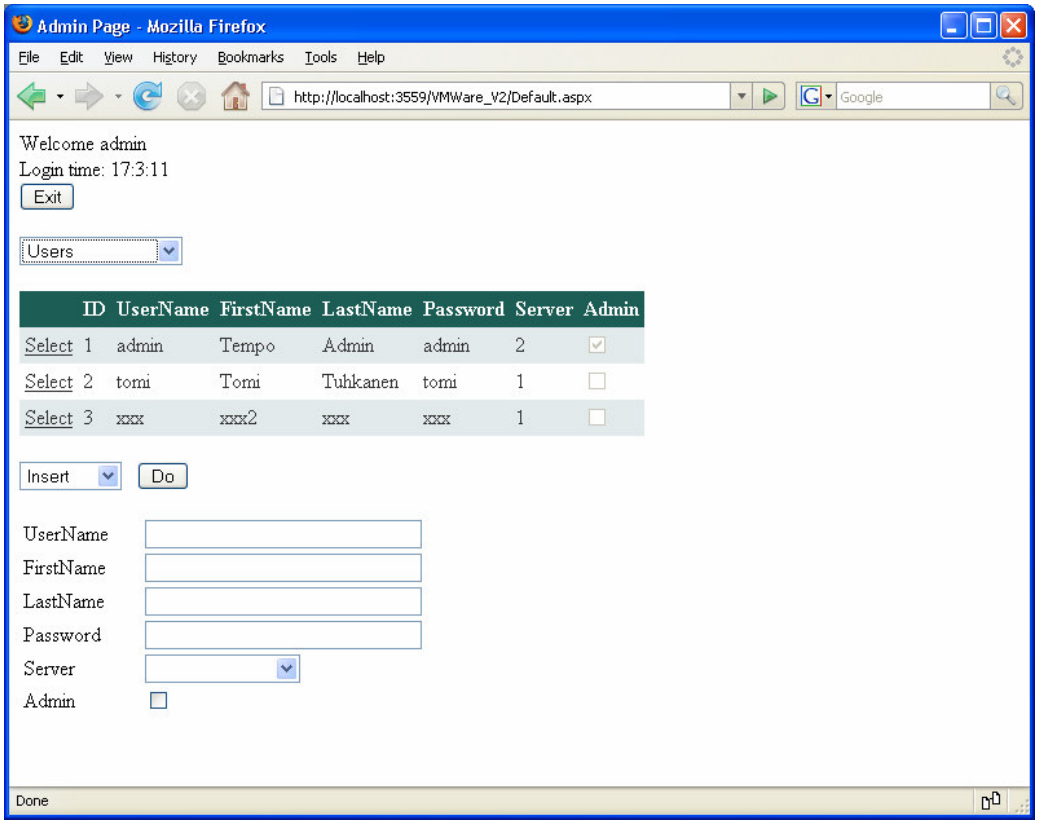
Kuva 49. Virheilmoitus

Lopuksi käyttäjä ohjataan käyttöoikuksia 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 (AdimnView.aspx)*

Sivulla on valittavat taulut DropDown –listassa. Taulu näytetään GridViewissä ja jokaisen taulun hallintaan liittyvät kontrollit on talletettu MultiViewin alle omiin View:hin. Käyttäjällä on insert-, update- ja delete-toiminnot DropDown-listassa (kuva 50).



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.

```
string cmd = "SELECT name FROM dbo.sysobjects WHERE xtype =
'U'";

SqlCmd = new SqlCommand(cmd, SqlConn);
dataA = new SqlDataAdapter(SqlCmd);
tables = new DataSet("Tables");
dataA.Fill(tables);

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

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

```
DataSet set = GetSet("SELECT * FROM " + ddList.SelectedValue);

private DataSet GetSet(string command)
{
  SqlConn = new SqlConnection(conString);
```

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

    return set;
}

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

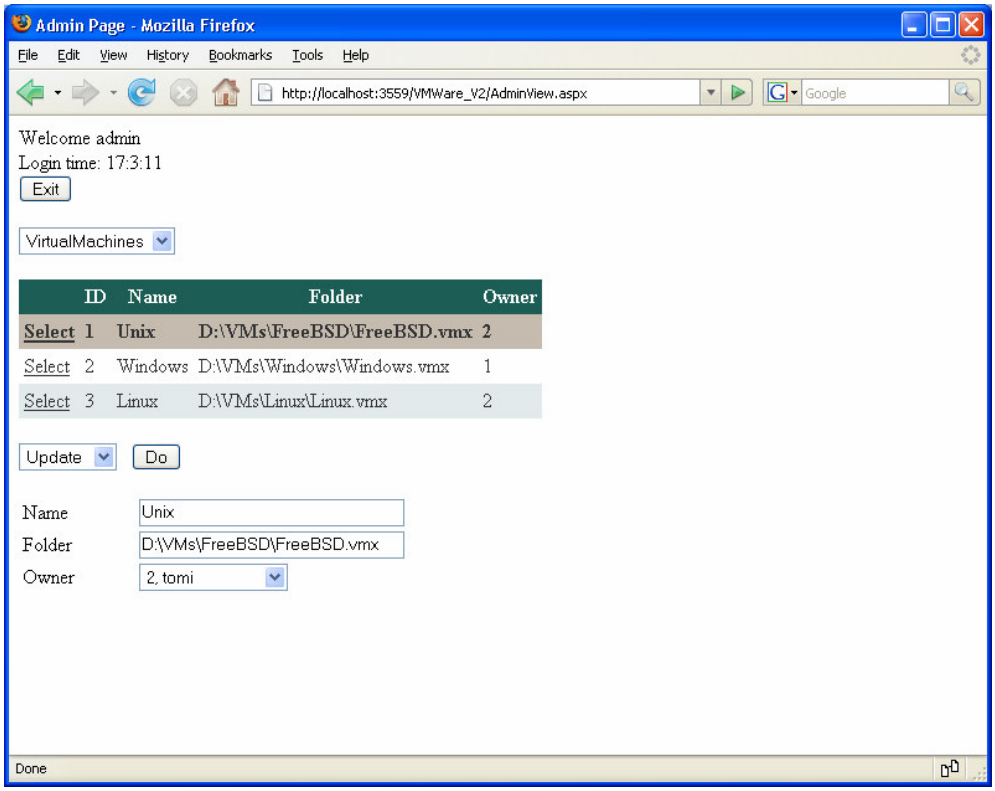
Aktiivinen näkymä valitaan automaattisesti listasta valitun taulun mukaan. Metodi käy läpi kaikki MultiViewin kontrollit ja vertaa jokaisen kontrollin ID:tä DropDown-listasta valitun taulun nimeen.

```
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;
      }
    }
}
```

Näkymän kontrolleihin haetaan tiedot GridViewin valitun rivin mukaan. Allaolevassa esimerkissä tiedot haetaan käyttäjätaulusta. Metodi asettaa TextBoxien teksteiksi valitun rivin vastaavat solut. Palvelin ComboBoxin tiedot haetaan SQL-palvelimelta. Tekstiksi asetetaan palvelimen ID, sekä nimi. Arvoksi asetetaan pelkästään ID, sillä vain arvo talletetaan tietokannan tauluun.

```
private void GetUserData()
{
    txtUserName.Text =
    GridView4.SelectedRow.Cells[2].Text.Trim();
    txtFirstName.Text =
    GridView4.SelectedRow.Cells[3].Text.Trim();
    txtLastName.Text =
    GridView4.SelectedRow.Cells[4].Text.Trim();
    txtUserPassword.Text =
    GridView4.SelectedRow.Cells[5].Text.Trim();

    fillList(ddServer);

    ddServer.SelectedValue =
    GridView4.SelectedRow.Cells[6].Text.Trim();
```

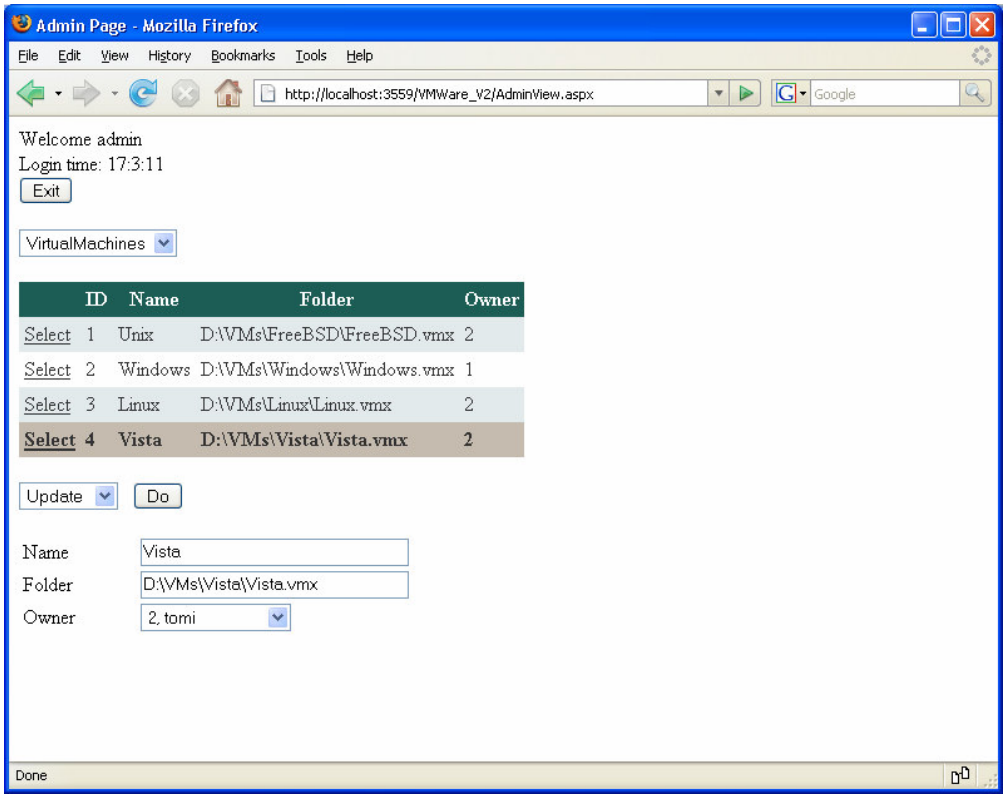
```
    chkBoxAdmin.Checked =
    ((CheckBox)GridView4.SelectedRow.Cells[7].Controls[0]).Chec
    ked;
}

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.

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

switch (table)
{
   case "VirtualMachines":
     command += "SET Name = '" + txtVmName.Text + "', ";
     command += "Folder = '" + txtVmFolder.Text + "', ";
     command += "Owner = '" + ddOwner.SelectedValue + "' ";
     command += "WHERE ID = " +
     GridView4.SelectedRow.Cells[1].Text;
     break;
   case "Users":
   // Koodia
}

SqlConn = new SqlConnection(conString);
SqlConn.Open();
SqlCmd = new SqlCommand(command, SqlConn);
SqlCmd.ExecuteNonQuery();
```

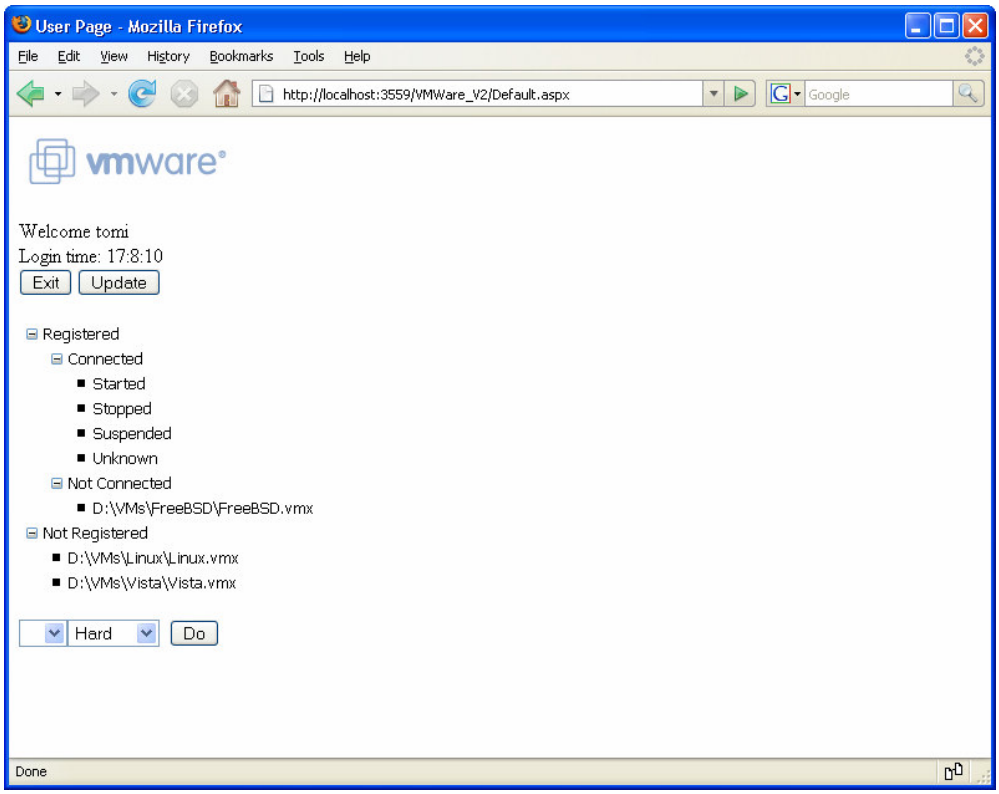


Kuva 52. Virtuaalikone valittuna päivitystä varten

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;
     }
   }
}
```

5.9.3 User Page (UserView)



Kuva 53. Käyttäjän aloitusnäkymä

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

```
private int CreateHashCode()
{
   // Create Hashcode using ip and logon name
   return Request.UserHostAddress.GetHashCode() +
   Request.LogonUserIdentity.Name.GetHashCode();
}
```

Ensimmäistä kertaa ladatessaan sivun käyttäjälle haetaan käyttäjätiedot sekä asetukset Default.aspx –sivulta. Nämä tiedot välitetään ThreadHandler-luokalle sen luonnin yhteydessä.

```
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;
```

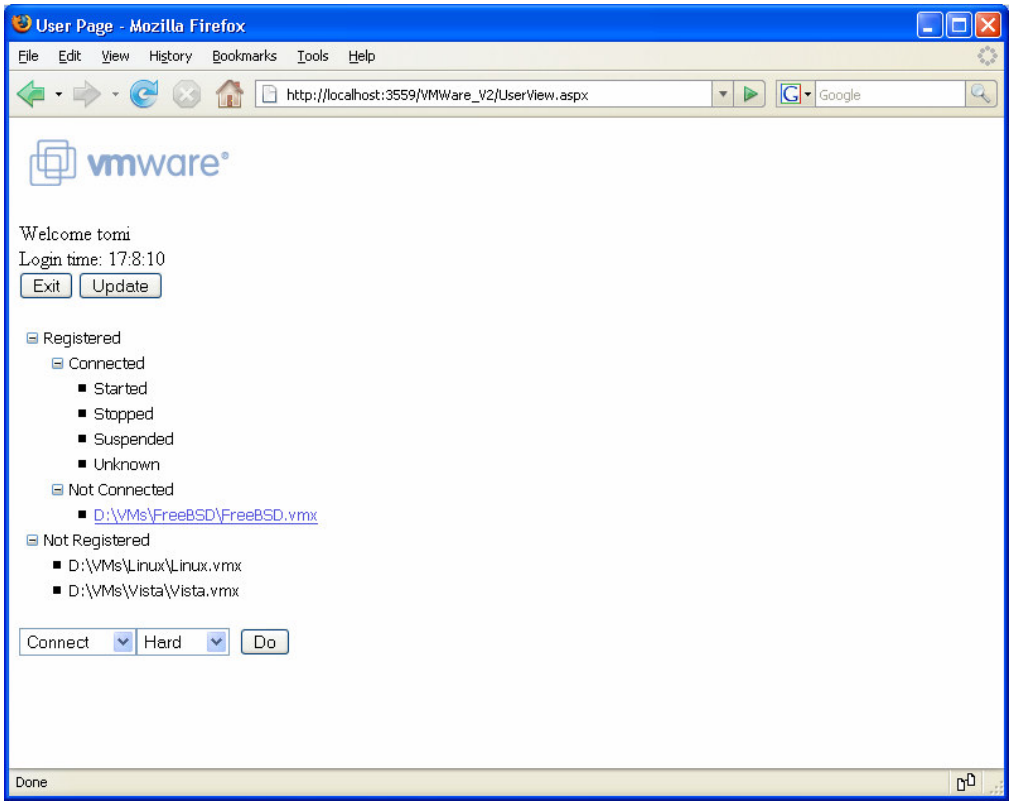
```
    tHandler = new ThreadHandler(userName, password, ip, port);
    tHandler.ready += new
    ThreadHandler.threadHandler(tHandler_ready);

    lblWelcome.Text = "Welcome " + userName;

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

    ConnectToServer();
}
```

Yhteys palvelimeen otetaan käynnistämällä uudessä 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 käyttäen käyttäjälle luotua avainta. Lisäksi asetetaan virtuaalikone näkymän datalähteeksi ThreadHandlerin XML-data.

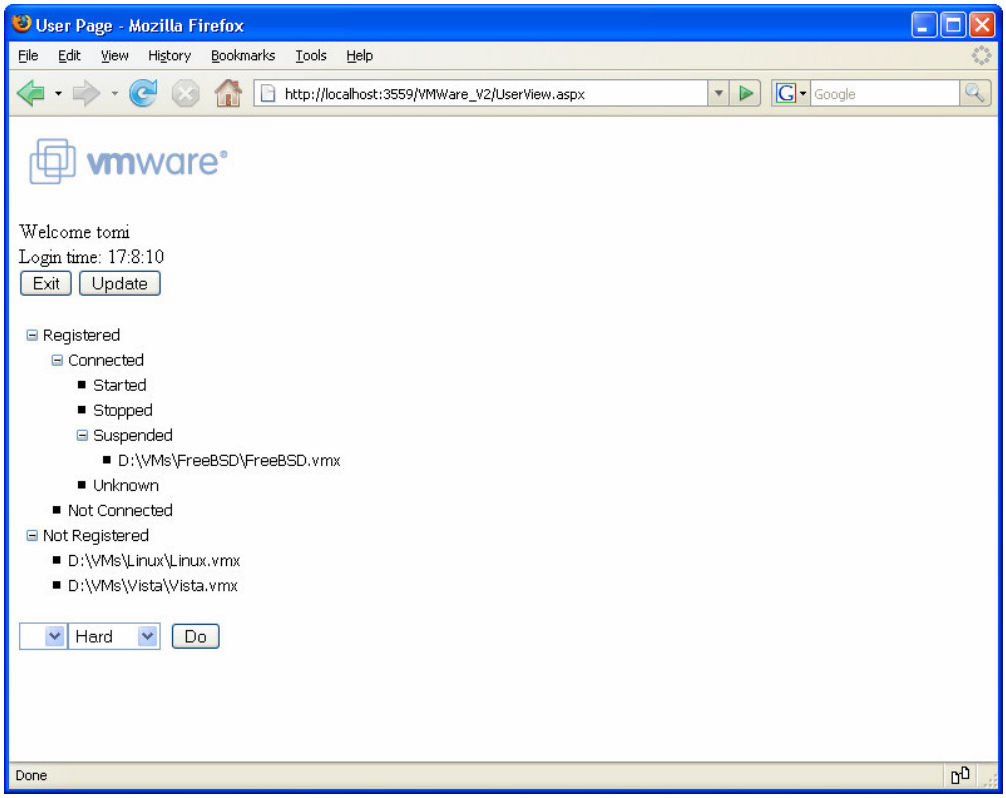
```
// Get handler from cache
Handler = (ThreadHandler)Cache[hash.ToString() + ":Handler"];
xml = tHandler.Xml;

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.

```
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
   }
}
```



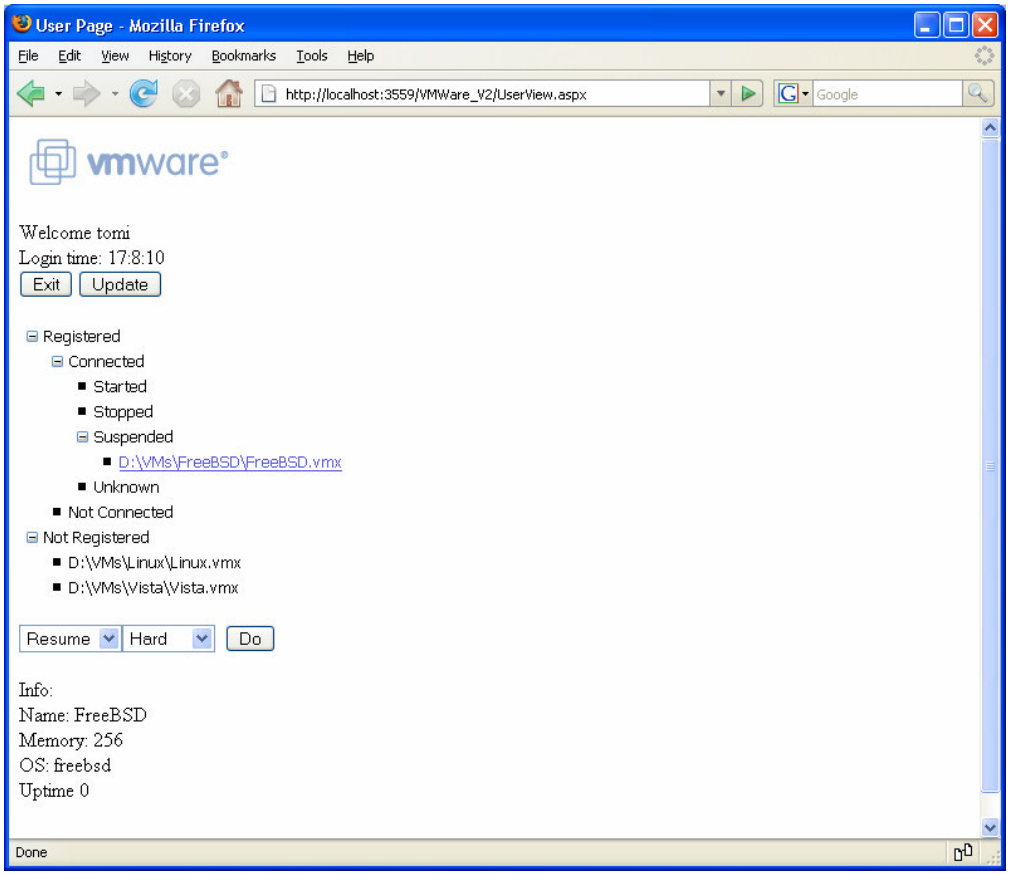
Kuva 55. Yhteys luotuna virtuaalikoneeseen

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

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

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

```
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 hallintaparametri samaan tapaan kuin Forms-pohjaisessa sovelluksessa. Toimenpide käynnistetään samalla tavalla kuin aiemmin esitetty 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":
```

```
            tHandler.Type = "Start";
            ControlVM();
            break;
          // Koodia
      }

      VmTree.DataBind();
    }
}
```

TreeView-kontrollin DataBind()-metodi käskee kontrollin sitoa annettuun datanlähteen kontrolliin. Normaalissa tapauksessa data olisi jo valmiiksi päivitettynä, mutta VMwaren Serverilla virtuaalikoneiden vaativat toimenpiteet kestävät pitkään, joten joudutaan odottamaan että ThreadHandler saa suoritettua toimenpiteen loppuun. Tämä varmistetaan katsomalla, onko säie, joka käynnisti ThreadHandlerin metodin, vielä elossa. Kun säie on ”kuollut”, päivitetään XML-tiedosto. Metodi asettaa puuhun vakiosolmut FillTree()-metodilla. Vakiosolmuja ovat kaikki solmut, jotka eivät ole virtuaalikonesolmuja. Tämän jälkeen käsitellään XML-tiedosto läpi ja asetetaan virtuaalikoneet oikeiden vakiosolmujen alle.

```
protected void VmTree_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"):
```

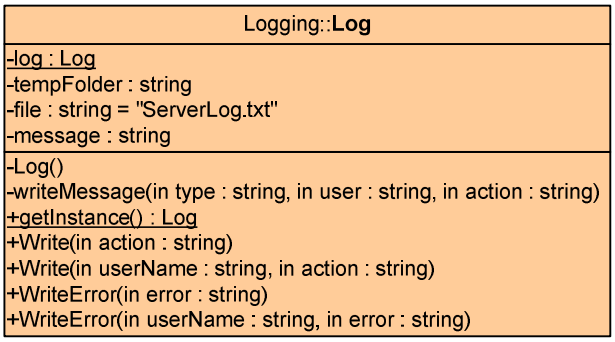
```
            // Koodia
        }
    }
```

5.9.4 *ThreadHandler-luokka*

ThreadHandler on toiminnaltaa hyvin samanlainen kuin VmCOM API Client –ohjelman ThreadHandler-luokka. Eroavaisuutena luokka ei kutsu User Pagea delegaatin kautta saatuaan tehtävän valmiiksi, vaan User Page odottaa, että ThreadHandler-luokka saa suoritettua tehtävänsä loppuun, kuten edellisessä kappaleessa selitettiin. Esimerkkinä ConnectToServer()-metodi, jota käytettiin myös VmCOM API -palvelipohjainen ohjelma esimerkkinä.

```
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



Kuva 57. Luokkakaavio

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

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

Viestin Type-attribuutin arvo kertoo, onko viesti normaali tapahtuma, Action, vai virhe, Error. Time-attribuuttiin tallennetaan tapahtuman aika ja User-attribuuttin käyttäjä. Käyttäjän ollessa System, viestin on aiheuttanut logger –ohjelmaa käyttävä ohjelma, eli käyttäjää ei ole tarkemmin määritetty.

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 konstruktosissa haetaan vain käyttäjän temp-hakemisto.

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

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

```
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();
  }
```

```
    return log;
}
```

Lock-statement lukitsee sisällään olevan koodin väin yhden säikeen käyttöön. Tässä tapauksessa lukituksella estetään tilanne, jossa kaksi oliota koittavat luoda instanssin samaan aikaan. Mikäli instanssia ei ole vielä luotu, ohjelma luo uuden instanssin ennen sen palauttamista.

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 pääkäyttäjille. Samalla saatiin myös eristettyä VMware Serverin yhteys varsinaisesta client-ohjelmasta.

Tiedonvälitys päätettiin 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 kehitysympäristön toimintakuntoon laittamisesta oli erityisin vaikea löytää dokumentaatiota. Suureksi avuksi olivat internetin monet keskustelupalstat. Tästä syystä kappaleessa 4 käsitellään 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 opiskeluaikanani oli varsin vähäistä.

TCP-palvelimen kanssa oli monia ongelmia, joista suurin oli socket-yhteyksien katoaminen. Lopulta päädyttiin ratkaisuun, jolloin käyttäjän jokainen toimenpide ottaa uuden yhteyden palvelimeen. Tämä kuluttaa kuitenkin turhaa kaistaa, mutta näin myös vältytään toimenpiteeltä, 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 sarakkeelle 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 näkymä 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 käyttäjäkokoelmansa läpi ja päivittäisi muuttuneet tiedot. Nykyisessä versiossa TCP-palvelin on käynnistettävä uudestaan, jotta käyttäjäkokoelmat saadaan päivitettyä.

Jokaisella käyttäjällä voi olla vain yhdellä VMware Serverillä virtuaalikoneita. Tämä tulisi muuttaa siten, että käyttäjälle ei olisi määrättynä omaa VMware Serveriä, 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 hyvin pelkistetyn virheilmoituksen, jolloin käyttäjän on vaikea tietää, mikä on vialla.

VIITELUETTELO

[1] Programming API Programming Guide. 30.6.2006 [viitattu 4.3.2007]. Saatavissa: http://www.vmware.com/pdf/Prog_API_Prog_Guide.pdf.

[2] Programming API Reference Guide. 11.8.2006 [viitattu 4.3.2007]. Saatavissa: http://www.vmware.com/pdf/Prog_API_Ref_Guide.pdf.

[3] VMware Server Data Sheet. 2006 [viitattu 4.3.2007]. Saatavissa: http://www.vmware.com/pdf/server_datasheet.pdf.

[4] VMware Virtualization Technology [web-dokumentti]. 2007 [viitattu 4.3.2007]. Saatavissa: http://www.vmware.com/virtualization/.

[5] An Introduction to Virtualization [web-dokumentti]. 2007 [viitattu 5.6.2007]. Saatavissa: http://www.kernelthread.com/publications/virtualization/.

[6] Palvelinten virtualisointi. 30.5.2006 [viitattu 3.3.2007] Saatavissa: http://www.profimill.fi/files/397_Palvelimienvirtualisointi.pdf.

[7] Heikniemi, Jouni, Microsoft .NET yhdistää. Mikrobitti 11/2004, s. 78 – 80.

[8] Faizullin, Marat, How To Write a Computer Emulator [web-dokumentti]. 18.7.2007 [viitattu 2.8.2007] Saatavissa: http://fms.komkon.org/EMUL8/HOWTO.html

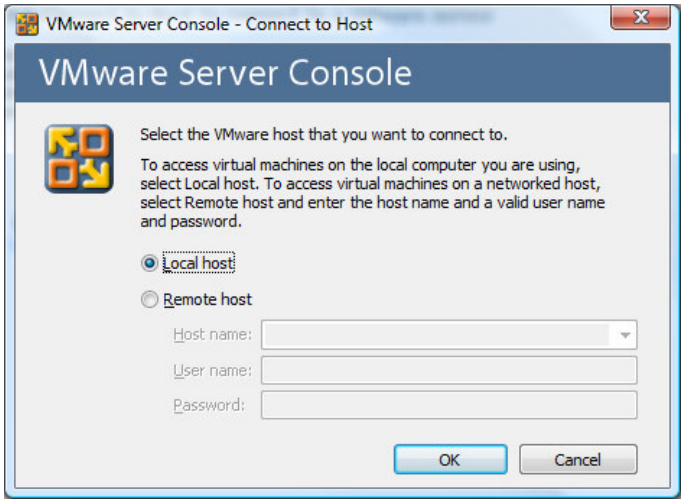
[9] VMware Scripting API. 2007 [viitattu 10.10.2007] Saatavissa: http://www.vmware.com/pdf/Scripting_API_23.pdf

[10] 10 reasons for Linux virtualization. [viitattu 16.10.2007] Saatavissa: http://h50043.www5.hp.com/hpservices/ap_features/Mar06/4001.htm

[11] Microsoft, Virtualization [web-dokumentti]. 2007 [viitattu 15.10.2007] Saatavissa: http://www.microsoft.com/virtualization/default.mspx

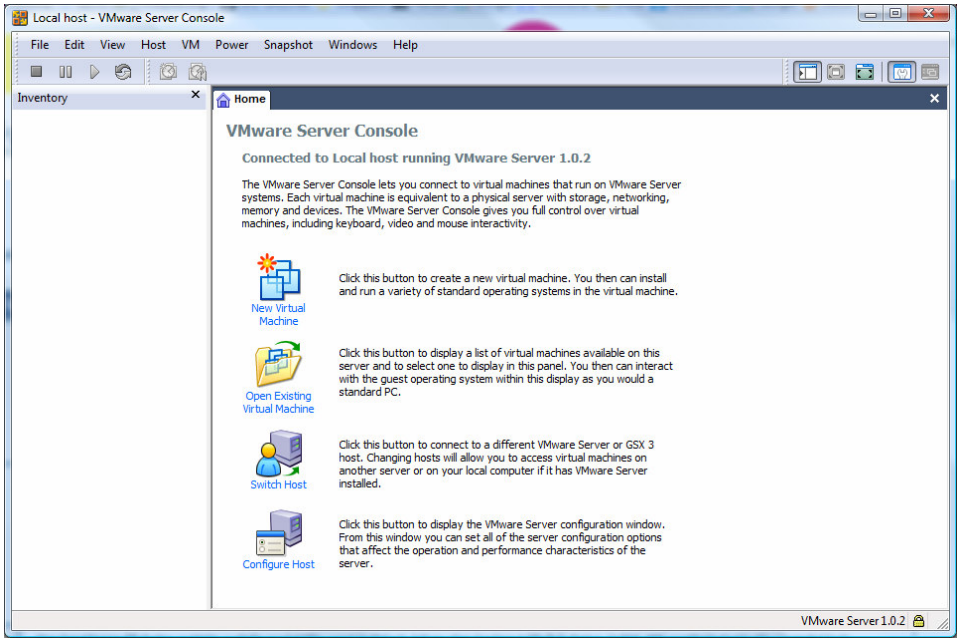
VIRTUAALIKONEIDEN ASENNUS

Aluksi luodaan uusi virtuaalikone, jonka jälkeen asennetaan siihen käyttöjärjestelmäksi Ubuntu 6.10 64-bittinen käyttöjärjestelmä. Käyttöjärjestelmä on saatavilla osoitteesta http://www.ubuntu.com/.



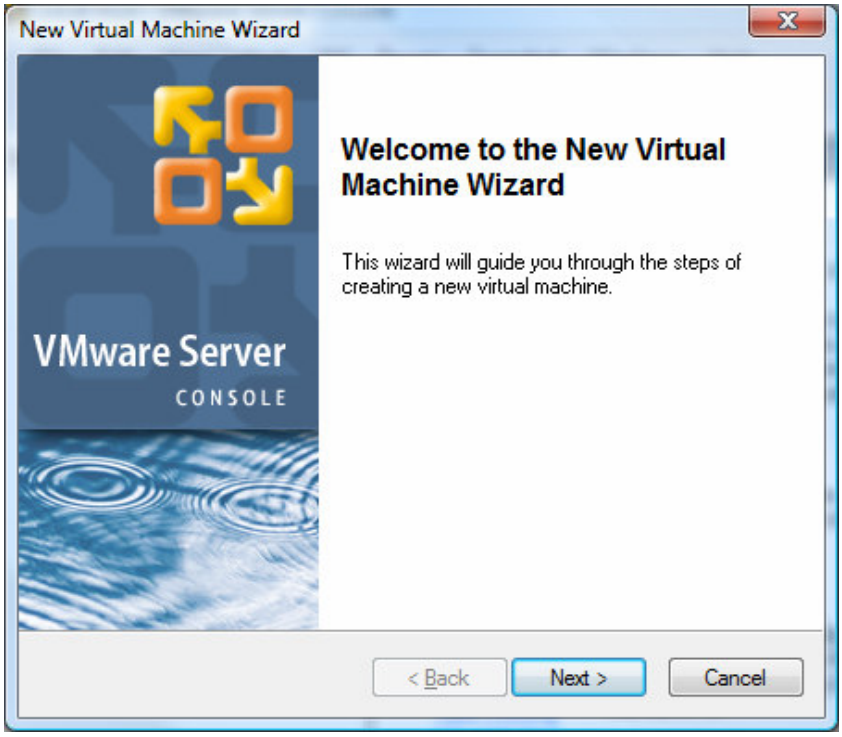
Kuva 1. Connect to Host

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



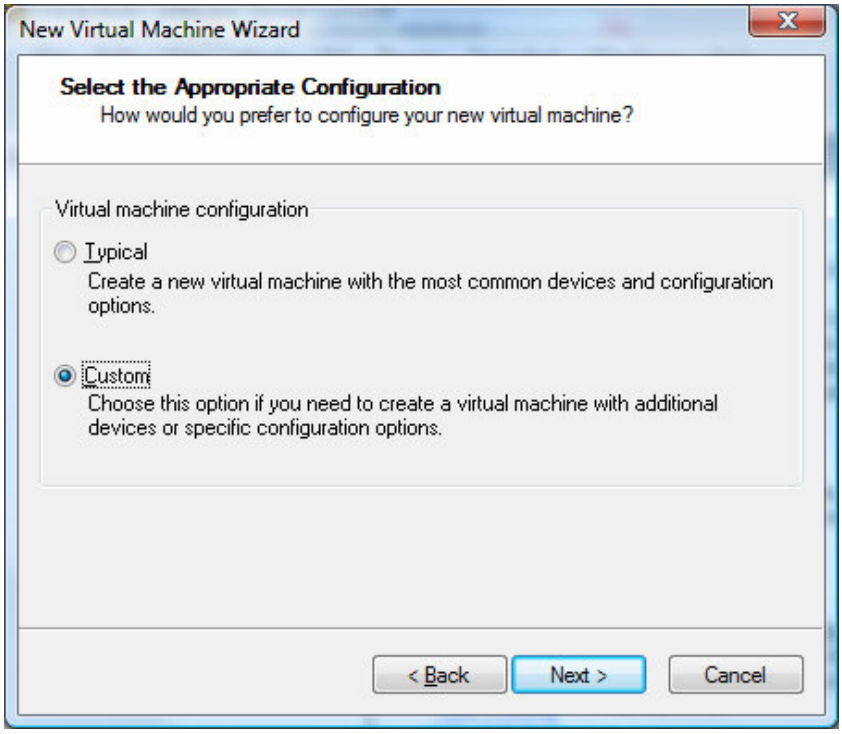
Kuva 2. Päänäkymä

2. Valitaan New Virtual Machine, jotta aloitetaan uuden virtuaalikoneen asennus.



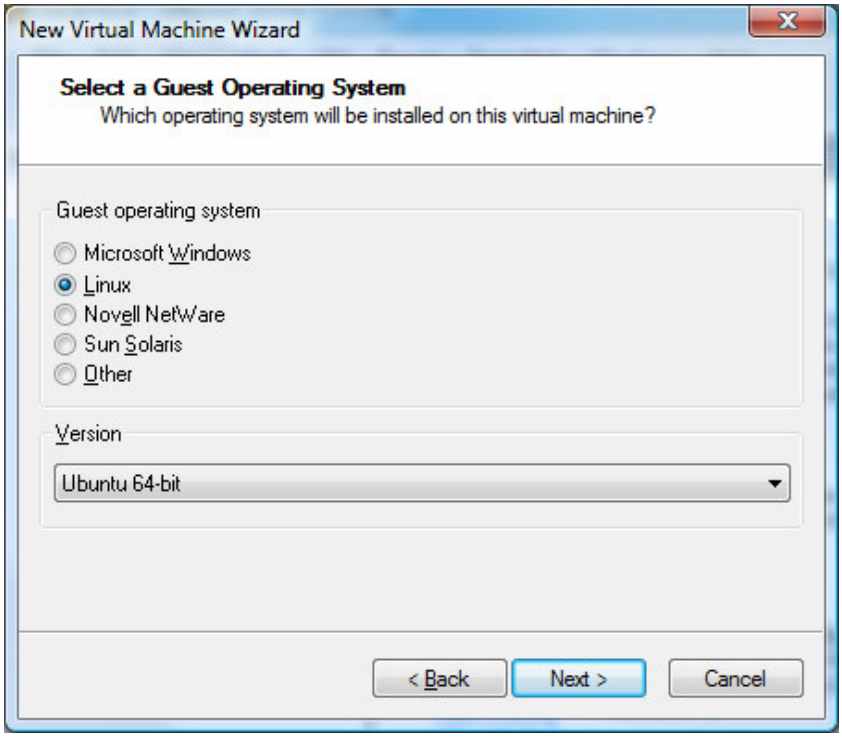
Kuva 3. Welcome to the New Virtual Machine Wizard

3. Jatketaan painamalla Next-nappia.



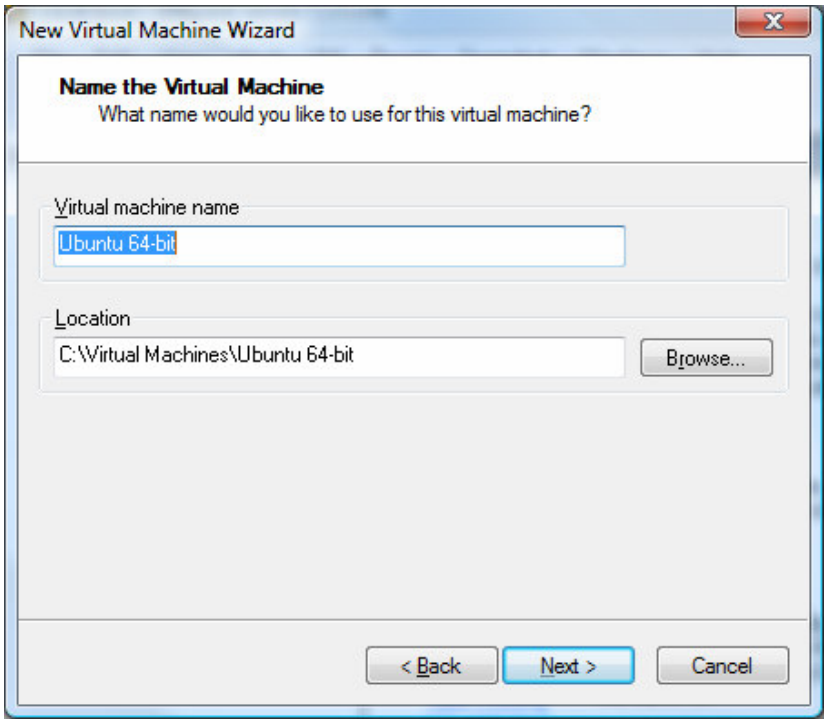
Kuva 4. Select the Appropriate Configuration

4. Valitaan Custom konfiguraatio, jolloin voidaan antaa tarkempia määrityksiä asennuksen suhteen.



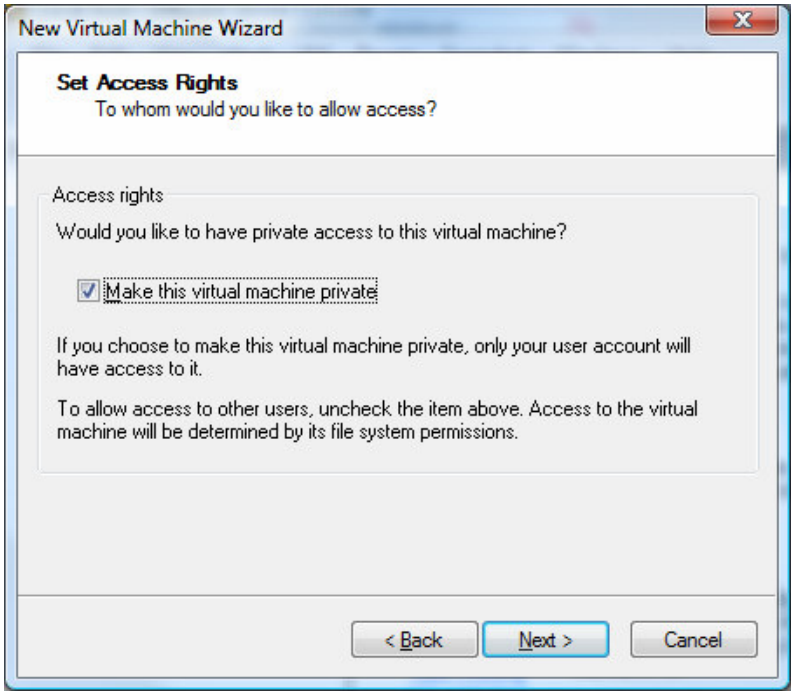
Kuva 5. Select a Guest operationg System

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



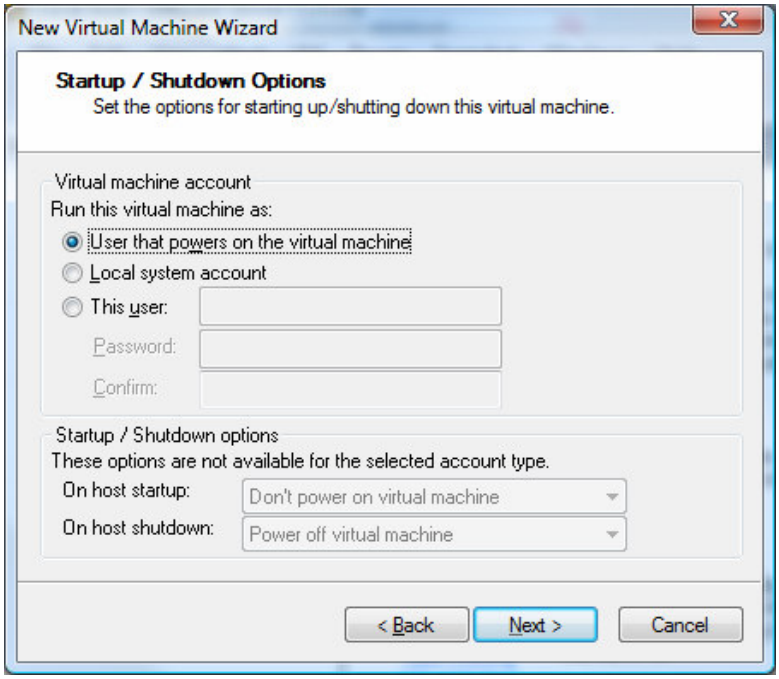
Kuva 6. Name the Virtual Machine

6. Virtuaalikoneen voi nimetä oman valintansa mukaan. Hakemistoksi määritetään hakemisto, jonne käyttäjä haluaa virutaalikoneen tulevan asennetuksi.



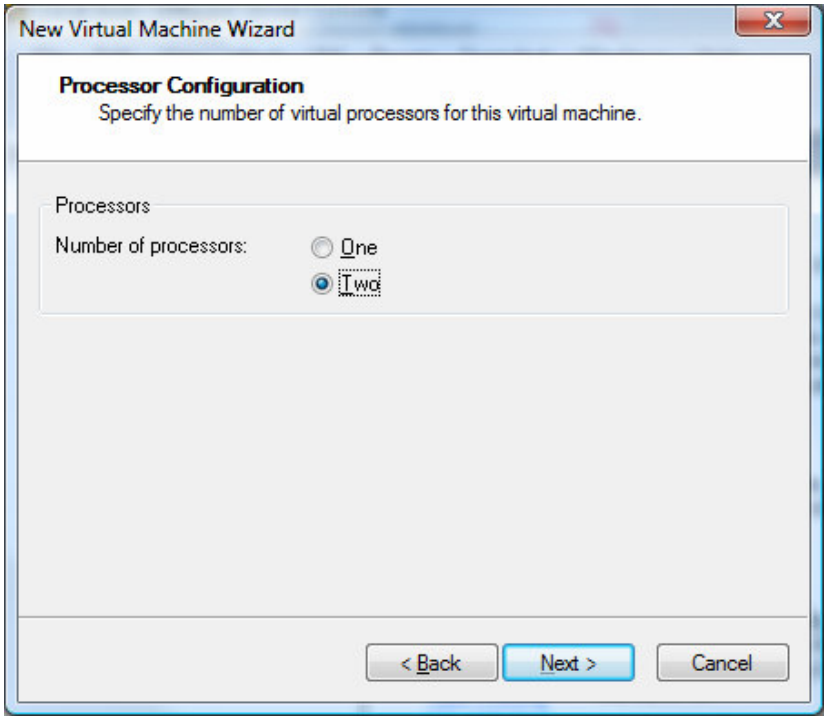
Kuva 7. Set Access Rights

7. Laitetaan virtuaalikone vain oman käyttäjätunnuksen käyttöön.



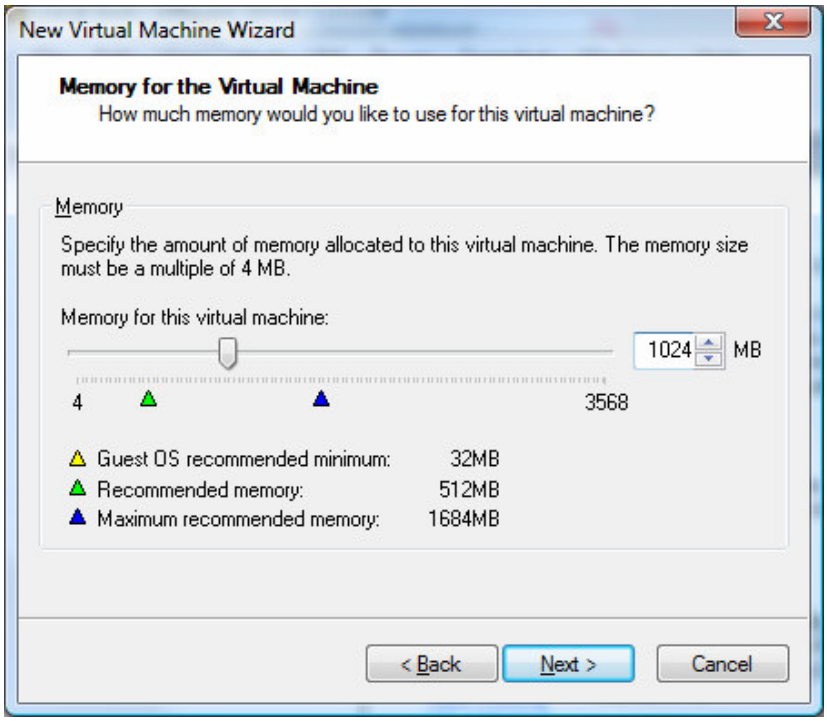
Kuva 8. Startup / Shutdown Options

8. Valitaan virtuaalikoneen accountiksi käyttäjä joka käynnistää virtuaalikoneen. Näinollen ei tarvitse syöttää käyttäjätunnusta ja salasanaa virtuaalikoneen käynnistyksen yhteydessä.



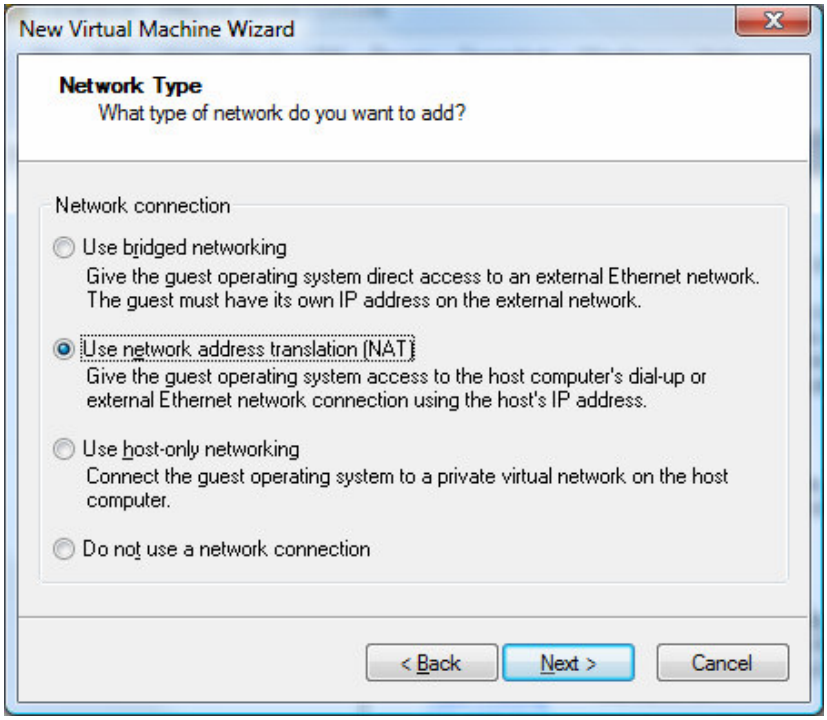
Kuva 9. Processor Configuration

9. Luodaan virtuaalikoneelle kaksi virtuaalista prosessoria.



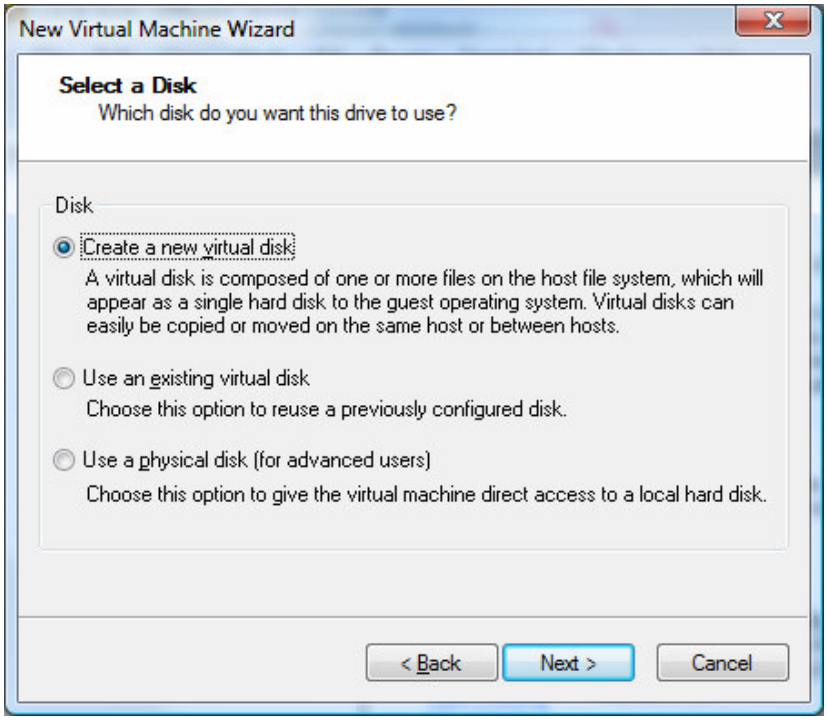
Kuva 10. Memory for the Virtual Machine

10. Annetaan virtuaalikoneelle 1024MB muistia käyttöön. Käytettävissä olevan muistin maksimimäärä määräytyy host-koneen muisin mukaan.



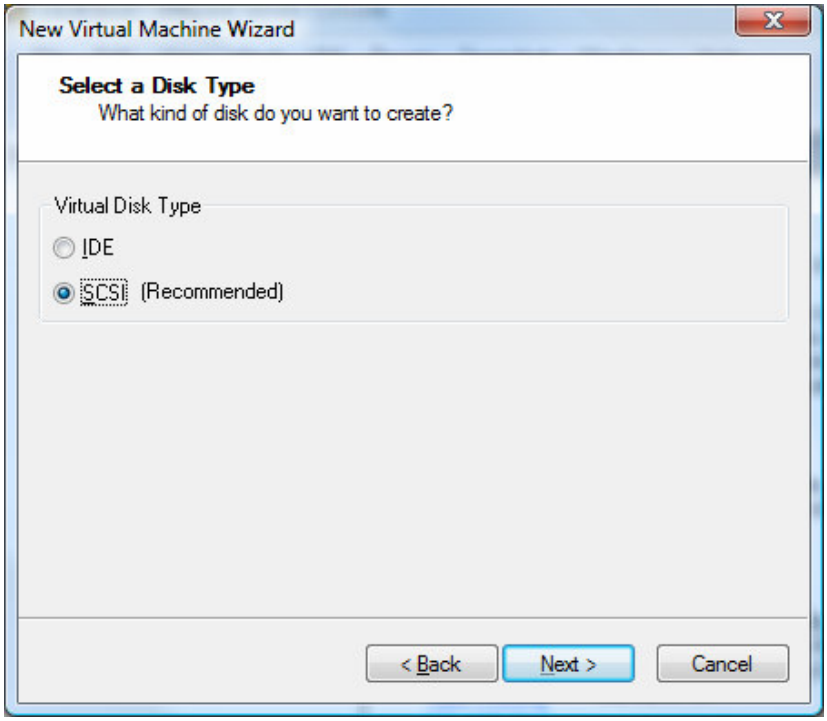
Kuva 11. Network Type

11. Asetetaan Network Address Translator käyttöön.



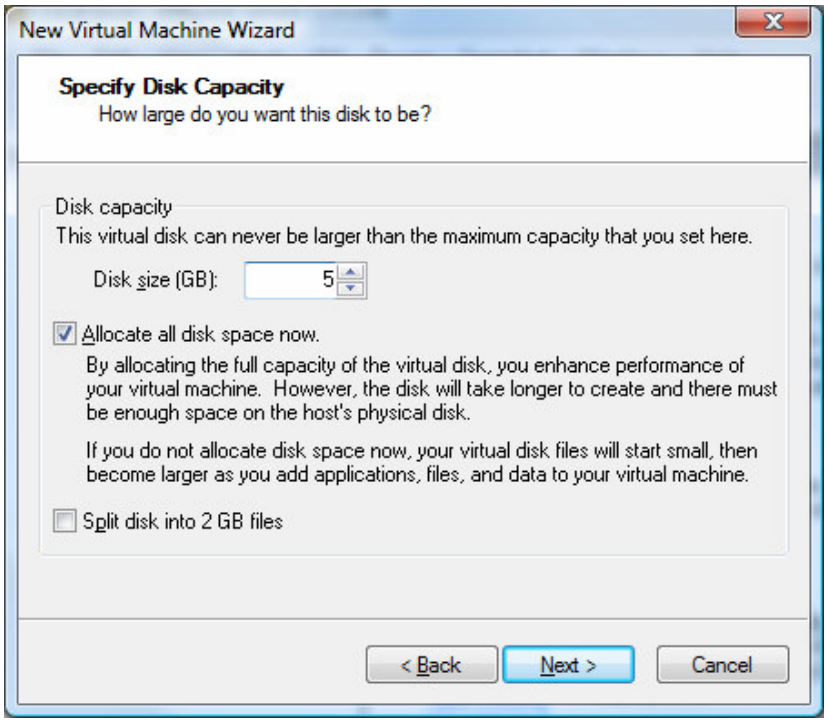
Kuva 12. Select a Disk

12. Luodaan uusi virtuaalinen kovalevy.



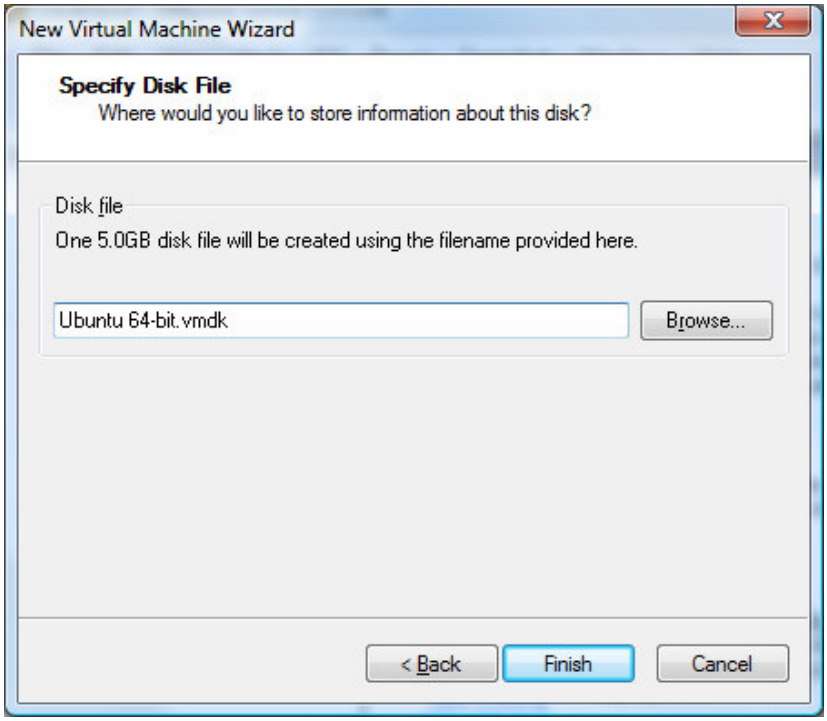
Kuva 13. Select a Disk Type

13. Asetetaan kovalevyn tyypiksi SCSI.



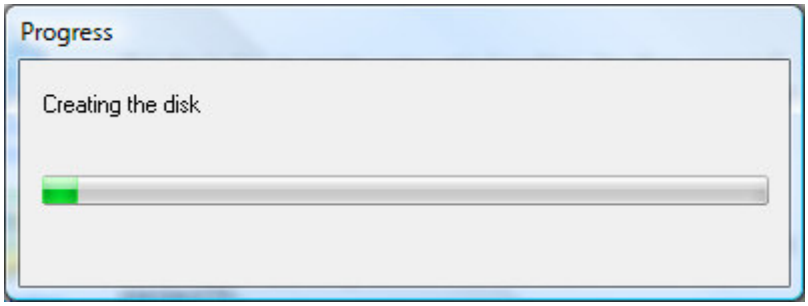
Kuva 14. Specifiy Disk Capacity

14. Asetetaan kovalevyn kooksi 5GB.



Kuva 15. Specify Disk File

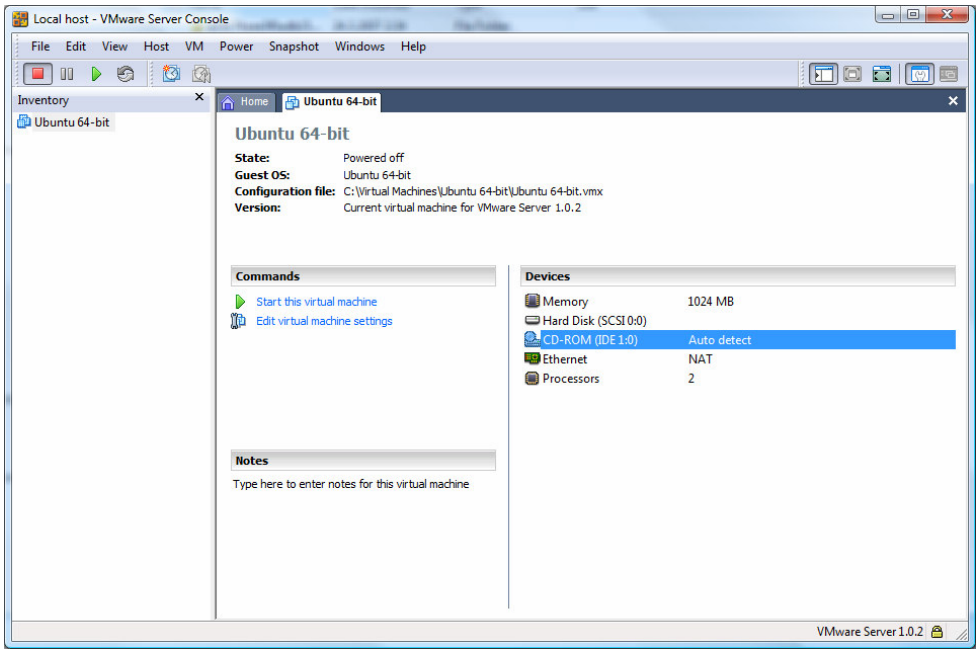
15. Annetaan virtuaalisen kovalevyn nimeksi Ubuntu 64-bit.vmdk. Tiedosto tallentuu automaattisesti asetettuun tallennushakemistoon.



Kuva 16. Creating Disk Progress

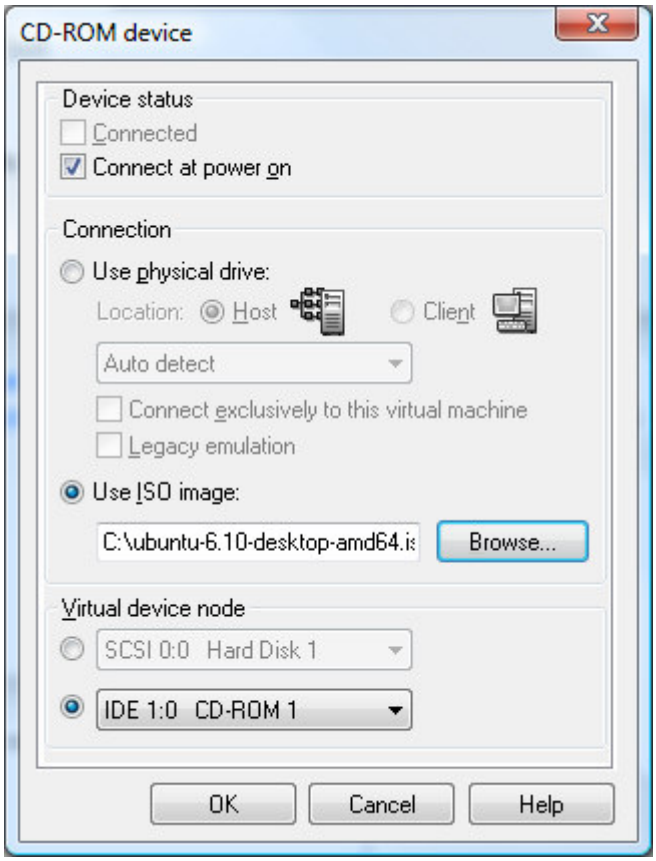
16. Kovalevyn luonti kestää koosta riippuen melko pitkään.

Kun virtuaalikone on luotu asennetaan virtuaalikoneeseen käyttöjärjestelmä. Käyttöjärjestelmän asennus tapahtuu aivan samaan tapaan kuin normaalikoneeseen tapahtuva asennus. Tässä asennuksessa cd-asemaksi virtualisoidaan Ubuntun cd:n image tiedosto.



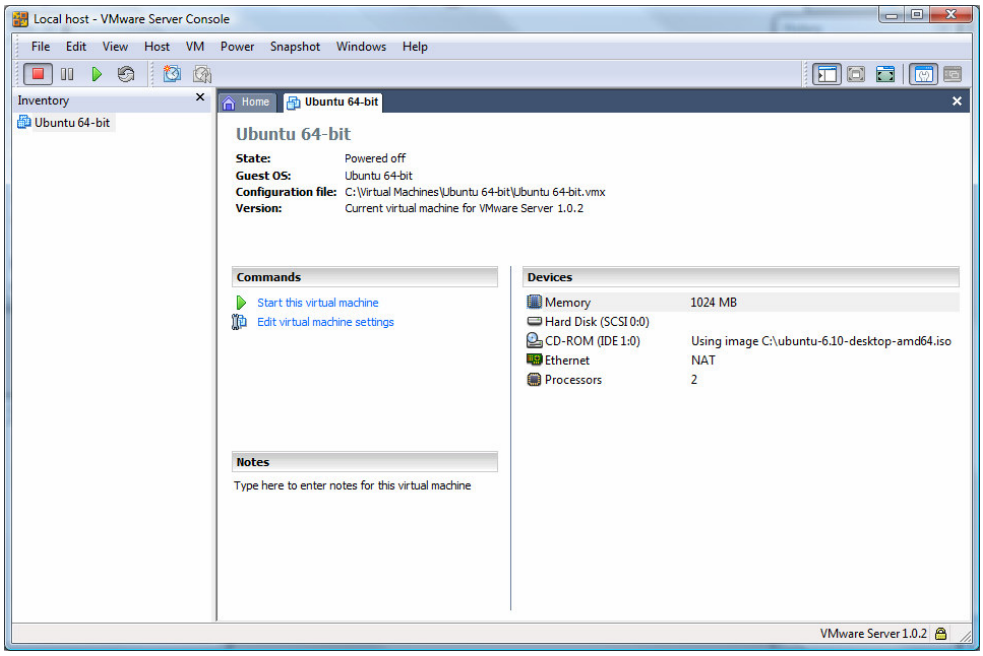
Kuva 17. Virtuaalikoneen pääikkuna

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



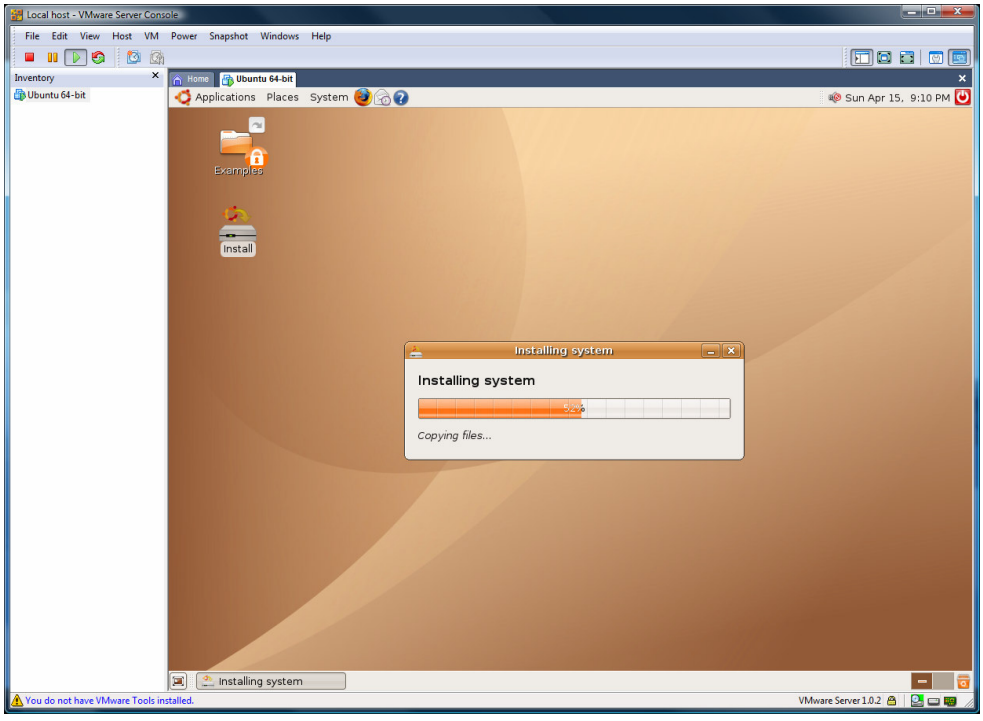
Kuva 18. CD-ROM drive settings window

2. Valitaan Use ISO image ja valitaan käyttöjärjestelmän image-tiedosto.



Kuva 19. Virtuaalikoneen pääikkuna

3. Klikataan Start the virtual machinea.



Kuva 20. Käynnissä olevan käyttöjärjestelmän pääikkuna

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