TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Opinnäytetyö

Jaakko Juutila

# TYÖKONESIMULAATTORIN ETÄHALLINTAKOMPONENTTI

Työn ohjaaja Lehtori Jari Mikkolainen
Työn teettäjä Creanex Oy, valvojana tuotekehityspäällikkö Markku Pusenius
Tampere 2009

TAMPEREEN AMMATTIKORKEAKOULU OPINNÄYTETYÖ
Tietotekniikan koulutusohjelma, ohjelmistotekniikka
Jaakko Juutila

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Juutila Jaakko Työkonesimulaattorin etähallintakomponentti
Opinnäyteyö 26 sivua
Työn ohjaaja Lehtori Jari Mikkolainen
Työn teettäjä Creanex Oy, valvojana tuotekehityspäällikkö Markku Pusenius
Tammikuu 2009
Hakusanat Etähallinta, Remoting, simulaattori

# TIIVISTELMÄ

Tutkintotyön tarkoituksena oli suunnitella ja toteuttaa Creanex Oy:n SimDriller -työkonesimulaattoriin etähallintakomponentti, jonka avulla verkon yli voidaan työkonesimulaattorissa käynnistää harjoituksia ja tallentaa harjoituksista luotavia harjoitusraportteja erillisellä simulaattorikoulutuksessa käytettävällä apuohjelmistolla.

Etähallintakomponentin kehityksessä oli ohjelmointikielinä C++/CLI ja C#. Etäyhteyden muodostamisessa käytettiin Microsoftin .NET -ohjelmistokomponettikirjaston tarjoamaa .NET Remoting -rajapintaa.

Työn tuloksena oli toimiva etähallintakomponentti, joka täytti sille asetetut vaatimukset. Jatkossa etähallintakomponenttiin tullaan kehittämään yksityiskohtaisempia työkonesimulaattorin hallintaominaisuuksia, kuten simulaattoriajon aikana ohjausjärjestelmään tehtävät muutokset etäkoneelta.

TAMK UNIVERSITY OF APPLIED SCIENCES
Information Technology
Software Engineering
Juutila Jaakko

Remote Control Component for Work Machinery Simulator

Engineering thesis
26 pages
Thesis supervisor
Jari Mikkolainen
Commissioning company
Creanex Oy, R&D Manager Markku Pusenius
January 2009
Keywords
Remote controlling, Remoting, simulator

# ABSTRACT

The purpose of this engineering thesis was to design and implement a remote control component for work machinery simulator to start simulator exercises and save training reports to a utility program.

C++/CLI and C# programming languages were used in the development of the component. Microsoft .NET Framework provides .NET Remoting application programming interface that was used for the remote connection. The result of this work was a component that fulfilled the requirements that were set.

# KÄYTETYT TERMIT JA LYHENTEET

| | |
|---|---|
| Appdomain | *Application Domain* on prosessia vastaava mekanismi CLR:ssä. |
| C# | *C#* on yksi .NET Frameworkin tukemista ohjelmointikielistä. |
| C++ | *C++* on yleiskäyttöinen ohjelmointikieli. |
| CIL | *Common Intermediate Library* on tavukoodia, jota suoritetaan virtuaalikoneessa. |
| CLR | *Common Language Runtime* on virtuaalikone Microsoftin .NET Framework:lle. |
| DLL | *Dynamic-link library* eli dynaamisesti linkitettävä kirjasto on Microsoftin toteutus jaetusta kirjastosta. |
| IANA | *Internet Assigned Numbers Authority* toimii tahona, joka hallitsee Internetin IP-osoitteita ja porttinumeroita. |
| IPC | *Inter-process communication* tarkoittaa kahden prosessin tai säikeen välistä kommunikaatiota. |
| Käännös | Käännöksellä tarkoitetaan CIL-koodiksi käännettyä tiedostoa, englanniksi assembly. |
| Marshallointi | Marshalloimisella tarkoitetaan olion muistiesityksen muuntamista dataformaattiin, joka soveltuu tallennukseen tai lähettämiseen. |
| TCP | *Transmission Control Protocol* on yhteydellinen tietoliikenneprotokolla. |
| URL | *Uniform Resource Locator* on merkkijono, jolla kerrotaan resurssin sijainti ja protokolla sen hakemiseen. |
| WCF | *Windows Communication Foundation* on sovelluskehys, jota käytetään keskenään kommunikoivien sovellusten kehittämiseen. |

# 1 Johdanto

Creanex Oy on liikkuvien työkoneiden suunnitteluun erikoistunut konesuunnittelun ammattilaisten perustama yhtiö, joka on erikoistunut uusien tuotteiden ja palvelujen konseptisuunnitteluun. Creanex Oy:n tuotteisiin kuuluu työkonesimulaattori, jonka opetuskäytön yhteydessä on noussut esiin kasvava tarve saada työkonesimulaattorin harjoitusajoissa tuottama data talletettua ja esitettyä ihmiselle helposti ymmärrettävässä muodossa. Tämän toteuttamiseksi päätettiin kytkeä työkonesimulaattori työkonekoulutuksen apuna olevaan koulutuksenhallintasovellukseen.

Työn tavoitteena oli tutustua .NET Framework -ohjelmistokomponenttikirjaston tarjoamaan Remoting-rajapintaan ja toteuttaa sillä etähallintakomponentti Creanex Oy:n natiivilla C++:lla toteutetun työkonesimulaattorin ja C#:lla toteutetun koulutuksenhallintasovelluksen välille. Työssä oli tarkoituksena myös tutustua C++/CLI-ohjelmointikieleen, jonka avulla pystyy helposti yhdistämään natiivia C++-ohjelmakoodia ja .NET Framework -luokkakirjaston tarjoamia palveluita. Etähallintakomponentin pääasiallisena käyttötarkoituksena on kyetä käynnistämään simulaattoriharjoitus työkonesimulaattorissa ja siirtämään työkonesimulaattorin tuottama harjoitusraportti koulutuksenhallintasovelluksen käsiteltäväksi. Työkonesimulaattorissa tapahtuva harjoitustietojen kerääminen ei sisälly tämän työn piiriin.

# 2 Työkonesimulaattorit

Creanex Oy on toteuttanut työkonesimulaattoreita usealle eri teollisuusalalle, esimerkiksi Harvester Head Simulator on simulaattori metsäkoneen hakkuupään ohjelmiston testaukseen ja kouluttamiseen, SimDriller on taas simulaattori porakonelaitteen kouluttamiseen ja testaamiseen. SimDriller-työkonesimulaattori on esillä kuvassa 1.

**Kuva 1. SimDriller-työkonesimulaattori.**

Työkonesimulaattoreita voidaan käyttää kokonaisintegrointitestaamiseen ennen kenttäkokeita. Työkonesimulaattorit mahdollistavat laitteen testaamisen työpisteen ääressä, jolloin ei tarvitse siirtyä mahdollisesti pitkänkin matkan päässä sijaitsevan työkoneen luo. Koulutuskäytössä työkonesimulaattorit madaltavat koulutettavan kynnystä oikean laitteen käyttöön siirtymiseen ja mahdollistavat luokkahuoneympäristössä tapahtuvan koulutuksen, jolloin on helppo tarkastella eteen tulevia kysymyksiä. Työkonesimulaattorin käyttäminen koulutuksessa ei varaa oikeita työkoneita, jolloin ne ovat käytettävissä työmaalla.

# 3 Koulutuksenhallintasovellus

Creanex Oy on kehittänyt työkonekoulutuksen aputyökaluksi koulutuksenhallintasovelluksen, jonka tarkoituksena on helpottaa koulutusta pitävien henkilöiden työkuormaa koulutusten suunnittelussa ja koulutusten pitämisessä. Koulutushallintasovelluksen ominaisuuksiin kuuluu kurssien koostaminen pienemmistä osakokonaisuuksista, koulutusmateriaalien ylläpito ja oppilaiden koulutustietojen tallentaminen.

Tässä työssä kehitettävä etähallintakomponentti mahdollistaa simulaattoriharjoituksista tallennettavien raporttien siirtämisen automaattisesti koulutuksenhallintasovelluksen säilöttäväksi, jolloin niitä voidaan tarkastella ja eri suorituksia vertailla keskenään.

# 4 .NET Framework

Microsoftin kehittämä .NET Framework on alusta sovellusten kehittämiseen ja ajamiseen. .NET Framework koostuu kahdesta pääkomponentista: .NET Framework Class Library (FCL) -luokkakirjastosta ja Common Language Runtime (CLR) -ajoympäristöstä. /3, s. 5/. Luokkakirjasto koostuu yli 7000 tyypistä, eli luokista, tietueista, rajapinnoista, enumeraatioista ja delegaateista, jotka on jaettu hierarkisiin nimiavaruuksiin. /3, s. 17, 18/.
Kehitysympäristö esikääntää .NET-yhteensopivan ohjelmakoodin Common Language Infrastructure (CLI) -määrityksen mukaiseksi tavukoodiksi, jota

kutsutaan nimellä Common Intermediate Language (CIL). Ajoympäristö lukee ja suorittaa esikäännetyn ohjelmakoodin, jolloin se käännetään konekielelle, jota käyttöjärjestelmä voi lukea ja suorittaa, just-in-time (JIT) -käännöksenä. /3, s. 6/. Ajoympäristö tarjoaa myös muistinhallinnan ja .NET-koodia kutsutaankin usein hallituksi (managed) koodiksi. Natiivia .NET:n ulkopuolista koodia kutsutaan ei-hallituksi (unmanaged). .NET:n tyypit ja oliot ovat myös hallittuja ja muistikekoa, johon .NET:n oliot luodaan, kutsutaan hallituksi keoksi. Kaikissa näissä tapauksissa termi hallittu tarkoittaa, että sitä hallitsee CLR-ajoympäristö. /1, s. 2/.

# 5 .NET Remoting

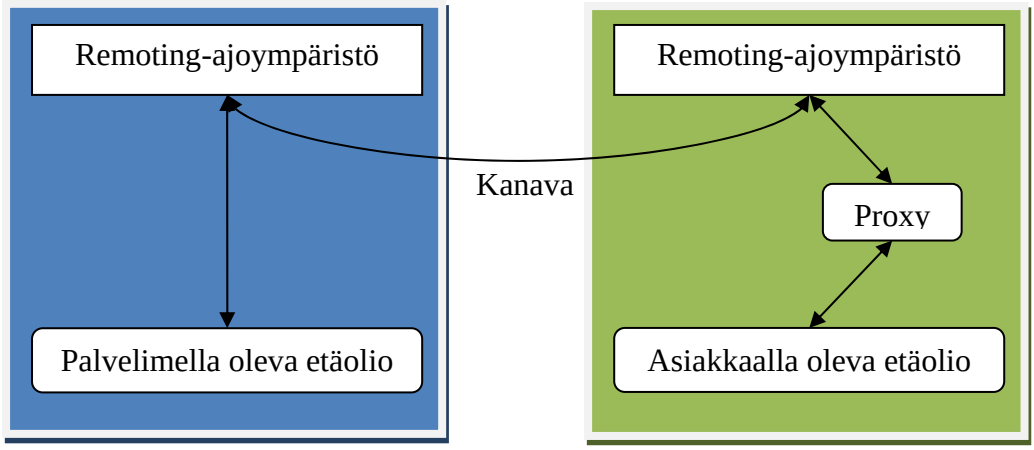
.NET Remoting on Microsoftin tekemä ohjelmistorajapinta prosessien väliseen kommunikaatioon (IPC), joka julkaistiin .NET Framework-ohjelmistokomponettikirjaston versiossa 1.0 vuonna 2002. Jatkossa tässä työssä käytetään kielellisistä syistä .NET Remotingsta Remoting-nimeä.

## *5.1 Yleiskatsaus*

Remotingin toiminta perustuu etäolioihin, jotka ovat käytettävissä eri sovellusalueilla (appdomain), prosesseissa ja tietokoneissa, joilla on verkkoyhteys toisiinsa. Remoting-ajoympäristö isännöi kuuntelijaa palvelinsovelluksen sovellusalueella olevan etäolion kutsuja varten. Asiakassovelluksen puolella kaikki kutsut etäolioon välitetään Remoting-ajoympäristön toimesta kanavan avulla, joka muodostaa putken etäolion ja asiakkaan väliin. Kanavat myös koteloivat datan todellisen kuljetustavan. Kuljetustapoina voidaan käyttää TCP-virtaa, HTTP-virtaa tai nimettyjä putkia. /3, s. 710–711/.

Remoting-ajoympäristö luo viittauksen etäolioon asiakassovellukselle, joka sitten käyttää etäoliota, kuin se olisi tavallinen paikallinen olio. Kuitenkin varsinainen koodin suorittaminen tapahtuu palvelinpuolella. Etäolio yksilöidään aktivointi-URL:n perusteella ja alustetaan yhdistämällä tähän samaan URL:ään. Remoting-ajoympäristö luo kuuntelijan etäoliolle, kun palvelin rekisteröi kanavan, jota käytetään etäolioon yhdistämiseen. Asiakaspuolella Remoting-ajoympäristö luo

proksin, joka esittää pseudoesiintymää etäoliosta. Se ei toteuta etäolion toiminnallisuutta, vaan tarjoaa samanlaisen rajapinnan. Tästä syystä täytyy etäolion julkinen rajapinta olla Remoting-infrastruktuurin tiedossa etukäteen.
Kaikki metodikutsut asiakassovelluksesta etäoliolle, mukaan lukien metodin nimi ja parametrit, serialisoidaan tavuvirtaan ja siirretään protokollariippuvaisen *Channel*-luokan olion avulla vastaanottavalle palvelinsovellukselle. Kuvassa 2 on esillä Remoting-infrastruktuurin muodostaman prosessin yleiskuvaus. /3, s. 710–713/.




**Kuva 2. .NET Remoting -infrastruktuurin muodostama prosessi.**

## *5.2 Arkkitehtuuri*

Käytettäessä etäolioita asiakkaalla ja palvelimella täytyy olla tiedossa samat rajapintamäärittelyt ja marshalloitavat arvotyyppiset luokat. Tämä johtaa sellaiseen yleiseen vaatimukseen, että Remoting-projektissa tarvitaan vähintään kolme käännöstä (assembly): jaettu käännös, joka sisältää marshalloitavat luokat ja rajapinnat; palvelinkäännös, joka toteuttaa MarshalByRefObjects-luokasta periytyvän olion; ja asiakaskäännös, joka käyttää niitä. /4, s. 14/.

## *5.3 Päätelmät*

Remoting-tekniikalla on helppo luoda keskenään kommunikoivia sovelluksia, koska sitä käyttämällä ei tarvitse luoda omia rajapintoja eri ohjelmointikielellä tai valita tarkalleen käytettävää kuljetuskanavaa etukäteen. /4, s. 9/. Tämän tekniikan

käyttäminen natiivilla C++-kielellä toteutetun simulaattorin kanssa vaatisi, että simulaattoriin tehtäisiin C++/CLI-kieltä käyttävä komponentti. C++/CLI-kieltä käsitellään tarkemmin luvussa 6.

# 6 C++/CLI-ohjelmointikieli

C++/CLI on joukko laajennuksia C++-ohjelmointikieleen, joiden avulla voidaan hyödyntää CLI:n tarjoamia palveluita. Näiden laajennuksien avulla ohjelmoija voi käyttää .NET Frameworkin rakenteita olemassa olevan C++-koodin seassa. C++-kieli sisältyy C++/CLI-kieleen, mikä tarkoittaa sitä, että validi C++-ohjelma on myös validi C++/CLI-ohjelma. Tämän seurauksena olemassa oleva koodipohja ei katoa vaan sitä voi jatkaa .NET Framework -lisäyksillä. Lisättäessä C++/CLI-koodia olemassa olevaan natiivista C++-koodista muodostuvaan projektiin, täytyy CLR-ajoympäristö olla asennettuna kohdekoneilla. /1, s. 3, 143/.

Yhteentoimivuus C++/CLI:n ja natiivin C++:n välillä voidaan tiivistää kahteen toimintoon:

- Olemassa oleva C++-koodi voidaan kääntää hallituksi koodiksi (lähdekoodiyhteensopivuus).
- Natiivi koodi ja hallittu koodi voidaan yhdistää sekakoodikäännökseksi (objektitiedostoyhteensopivuus).

/1, s. 8/.

## *6.1 C++/CLI:n käyttäminen*

C++/CLI-ohjelmakoodi muistuttaa hyvin pitkälti tavallista C++-koodia, mutta poikkeuksiakin löytyy. Ominaisuudet esimerkiksi moniperintä, private- ja protected-tyyppinen perintä, vakiojäsenmuuttujat, union-avainsana, argumenttien oletusarvot ja ystäväfunktiot eivät ole tuettuna C++/CLI:ssä. /1, s. 73/.

## 6.2 Muistinhallinta

Olioiden luominen muistikekoon tapahtuu C++:n `new`-operaattorista poikkeavalla tavalla. C++/CLI:ssä käytetään `gcnew`-operaattoria, jolla luotavalle oliolle varataan tila hallitusta muistikeosta. Koska natiivi muistiosoitin ei ole riittävä osoittamaan sijaintia hallitusta muistikeosta, käytetään siihen tarkoitukseen erilaista muuttujaa: seurantakahvaa (tracking handle), jossa tähden (*) tilalla on kohdistinmerkki (^). /1, s. 15, 16/.
Koodiesimerkki `gcnew`-operaattorin käytöstä:

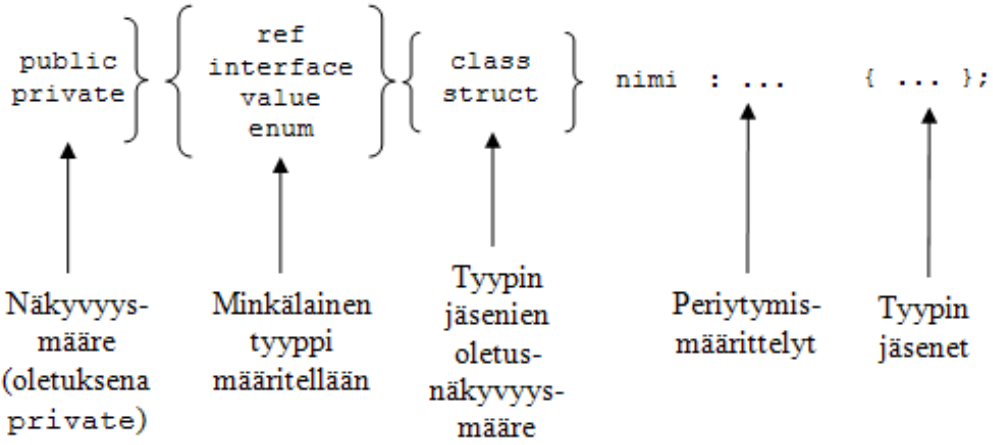
```
int^ i = gcnew int(5);
```

hallitussa muistikeossa olevan muistivarauksen elinikä ei rajoitu metodikutsuun tai metodikutsun näkyvyysalueeseen kuten C++:n `new`-operaattorilla varattu muistialue. Poikkeuksena on kuitenkin se, että hallitussa muistikeossa olevaa muistinvarausta ei tarvitse ohjelmoijan vapauttaa kuten C++:ssa. Hallitussa muistikeossa olevien muistivarausten elinikää ohjaa roskienkeruu (garbage collector, GC), joka ei ole suoraan ohjelmoijan hallittavissa. C++/CLI:ssä `delete`-operaattorin käyttäminen seurantakahvaan vastaa metodikutsua `IDisposable`-rajapinnan `Dispose`-metodiin. Tämä ei suoraan vapauta oliolle varattua muistia hallitusta muistikeosta, vaan se vapauttaa ei-hallittuja resursseja, kuten tiedostoja, virtoja ja kahvoja. /1, s. 15, 124, 126/.

## 6.3 Hallittujen tyyppien luominen

Kuvassa 3 on esillä hallittujen tyyppien määrittelymalli. Avainsanat `class` ja `struct` ovat saaneet etuliitteen, joka kertoo, minkälaista hallittua tyyppiä ollaan määrittämässä. /1, s. 73/.

Näistä tarkemmin käsitellään avainsanat `ref` eli viittaustyyppi ja `value` eli arvotyyppi. Avainsana `interface` tarkoittaa, että määriteltävä tyyppi on rajapinta, ja `enum` tarkoittaa, että määriteltävä tyyppi on enumeraatio. /1, s. 73/.




**Kuva 3. Hallittujen tyyppien määrittelymalli.**

### 6.3.1 Ref-tyyppimäärittely

Avainsana `ref` määrittelee, että tyyppi on viittaustyyppinen. /1, s. 73/. Viittaustyypit säilövät muistiosoitteen ja ne allokoidaan muistikeosta. Viittaustyypit voivat olla luokkia, taulukoita, osoittimia tai rajapintoja. /2/. Seuraavassa esimerkissä luodaan julkinen viittaustyyppinen luokka `FooBar`:

```
public ref class FooBar
{ }
```

### 6.3.2 Value-tyyppimäärittely

Avainsana `value` määrittelee, että tyyppi on arvotyyppi. Useimmissa tapauksissa on suositeltavampaa luoda viittaustyyppi arvotyypin sijaan, sillä arvotyypeissä on useita rajoituksia. Arvotyypit eivät tue periytymistä, eli ne eivät voi periytyä eikä niistä voi periyttää toista tyyppiä. Erikoisjäsenmuuttujat, kuten oletusrakentaja, kopiorakentajat, kopio-operaattorit ja purkajat eivät ole tuettuja arvotyypeissä. Arvotyyppien käyttäminen voi olla hyödyllistä, kun monia instansseja tarvitaan samaan aikaan. Arvotyypeistä voidaan luoda useita instansseja samaan aikaan luomalla hallitun taulukon. *A*:n ollessa arvotyyppi lauseke

```
gcnew array<A>(100)
```

varaa muistia sadalle *A*:n instanssille yhdessä palassa. Viittaustyypille `V` lauseke

```
gcnew array(V^)(100)
```

varaisi muistia sadalle seurantakahvalle ja kaikki instanssit täytyisi luoda erikseen. Arvotyyppien käyttämien tässä tapauksessa parantaa myös roskienkeruun suorituskykyä. Käyttämällä arvotyyppitaulukkoa roskienkeruun täytyy siivota vain yksi hallittu olio. Käyttämällä viittaustyyppitaulukkoa, jossa on n-kappaletta elementtejä ja niistä jokainen viittaa uuteen viittaustyypin instanssiin, roskienkeruun täytyy siivota jokainen instanssi erikseen taulukon lisäksi. /1, s. 76, 77/.

## *6.4 Kääntäjäasetukset*

Visual C++ -kääntäjä tarjoaa neljä erilaista käännösmallia, joilla voidaan ottaa objektitiedostoyhteensopivuus tai molemmat C++/CLI:n yhteentoimivuusominaisuudet pois käytöstä. Ne ovat /clr, /clr:pure /clr:safe ja natiivi käännös. /1, s. 8/.

Jos käytössä ei ole yksikään /clr-kääntäjäargumenteista, suoritetaan silloin natiivi käännös. Käytettäessä /clr-kääntäjäargumenttia, on käytössä molemmat yhteentoimivuusominaisuudet. Kääntäjäargumentti /clr:pure mahdollistaa olemassa olevan C++-koodin kääntämisen hallituksi koodiksi, mutta sitä käyttämällä ei voi tuottaa sekakoodikäännöksiä, jotka vaativat objektitiedostoyhteensopivuuden. Kääntäjäargumentilla /clr:safe voidaan tuottaa verifioitavaa koodia, joka tarkoittaa sitä, että lähdekoodissa voi olla vain .NET:n tyyppejä eli natiiveja C++-tyyppejä ei voida käyttää. Verifioitava koodi on vaatimuksena .NET:n turvamallille nimeltään Code Access Security (CAS). /1, s. 9/.

Koska objektitiedostoyhteensopivuus ei ole tuettu /clr:pure-käännöksessä, on /clr-käännösmalli järkevin valinta, kun halutaan laajentaa olemassa olevaa projektia hallitulla koodilla. /1, s. 146/.
Kääntämällä hallituksi koodiksi vain ne tiedostot, jotka hyötyvät hallitusta suorittamisesta, voidaan erittäin tehokkaasti pienentää hallitun suorittamisen luomaa yleisrasitetta. Tällä yleisrasitteella on monia muotoja, kuten se että

metatietoa lisätään jokaiseen hallittuun funktioon, jota kutsutaan natiivista koodista, ja jokaiseen natiiviin tyyppiin, jota käytetään hallitussa koodissa. Kaikki tämä metatieto kasvattaa luotavan käännöksen kokoa, tarvittavan muistin määrää ja latausaikaa. /1, s. 146/.

# 7 Etähallintakomponentin toteutus

Kehitysympäristönä etähallintakomponentin toteuttamisessa käytettiin Microsoft Visual Studio 2008 –ohjelmistoa. Tässä työssä kehitetty etähallintakomponentti käyttää .NET Frameworkin versiota 2.0.

## *7.1 Määrittely*

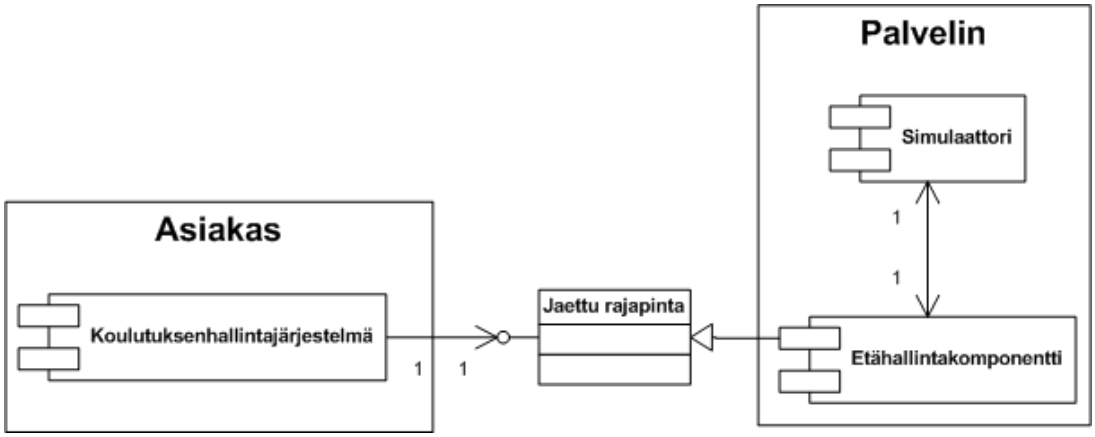
Työn alussa ja toteutusvaiheessa asetettiin etähallintakomponentille seuraavanlaisia toiminnallisia määrityksiä:

- Harjoitustiedostoja täytyy pystyä lähettämään asiakkaalta palvelimelle, jotta voidaan käyttää muitakin harjoituksia kuin niitä, jotka ovat jo simulaattorissa.
- Simulaattorin tuottamia harjoitusraportteja täytyy pystyä lataamaan, jotta ne saadaan tallennettua koulutuksenhallintasovelluksella.
- Etähallintakomponenttia täytyy pystyä käyttämään verkkoyhteyden avulla sekä paikallisesti. Yleensä etähallintakomponentilla ohjataan simulaattoria verkkoyhteyden avulla, mutta testaamisessa ja joissain erikoistapauksissa täytyy etähallintakomponentilla pystyä ohjaamaan samalla tietokoneella olevaa simulaattoria.
- Etäyhteyden yli tapahtuvat metodikutsut eivät ole aikakriittisiä, koska käynnistettävien toimintojen suorittaminen kestää sovelluksen näkökulmasta pitkän ajan.
- Palvelimeen samaan aikaan yhteydessä olevien asiakkaiden määrä on rajoitettu yhteen, koska simulaattorissa voi olla käynnissä vain yksi harjoitus kerrallaan.

## 7.2 Suunnittelu

Remoting-tekniikan käyttämiseen päädyttiin lyhyen esitutkimuksen perusteella. Esimerkiksi DCOM-tekniikan käyttäminen ei vaikuttanut houkuttelevalta ja haluttiin ottaa käyttöön uutta tekniikkaa, joka on hyvin yhteensopiva .NET Frameworkin kanssa. Kuitenkaan Remotingin korvaavaa uutta WCF-tekniikkaa ei haluttu ottaa vielä käyttöön, koska se olisi vaatinut .NET Framework 3.0:n käyttöönottamista, jota ei kuitenkaan haluttu vielä tehdä.

Edellä kuvattuun skenaarioon liittyy Remoting-arkkitehtuurin mukaiset kolme osaa: asiakas, jaettu käännös ja palvelin. Asiakas on koulutuksenhallintasovellus, josta suoritetaan kutsuja palvelimelle etähallintakomponentin välityksellä. Palvelin on työkonesimulaattori, jolla ajetaan asiakkaan käynnistämiä simulaattoriharjoituksia. Jaettu käännös määrittelee toiminnot, joita voidaan suorittaa palvelimella. Koska kutsuja on tarkoitus tehdä vain asiakkaalta palvelimelle, ei palvelinsovelluksen tarvitse tietää asiakkaasta mitään. Tämä selkeyttää etähallintakomponentin rakennetta. Kuvassa 4 on esillä etähallintakomponenttiin liittyvät komponentit.




**Kuva 4. Etähallintakomponenttiin liittyvät komponentit.**

## 7.3 Jaettu käännös

Jaettu käännös käännetään omaan DLL-tiedostoonsa, jossa on vain yksi C#-kielellä toteutettu rajapintaluokka eli `IRemoteObject`.

```
public interface IRemoteObject
{
```

```
    bool DoTraining(String trainingName);
    void AbortCurrentTraining();
    bool WasReportGenerated();
    byte[] ReadTrainingReport();
    void SetLocalTrainingDirectory(String directory);
    bool SendTrainingFile(System.IO.FileStream stream);
    bool DoesTrainingFileExist(String fileName);
}
```

Tämä rajapinta määrittää ne toiminnot, jotka voidaan suorittaa palvelimella.

## *7.4 Palvelin*

Palvelinsovellusta varten luodaan DLL-luokkakirjasto, jossa on etäolioluokka, etäolion Remoting-ajoympäristöön rekisteröimiseen käytettävä luokka sekä natiivina C++-rajapintana toimivat julkiset funktiot. Luokkakirjastossa toteutetaan C++/CLI-kielellä etäoliona toimiva luokka `CRigSimRemoteObject`. Se periytetään .NET Frameworkin `System.MarshalByRefObject`-luokasta, jotta sitä voidaan käyttää Remoting-ajoympäristön avulla eri sovellusalueilla. Se myös toteuttaa jaetun käännöksen `IRemoteObject`-rajapinnan, jotta sitä voidaan käyttää asiakassovelluksesta.

`CRigSimRemoteObject`-luokan olio rekisteröidään palvelinsovelluksen käynnistyessä Remoting-ajoympäristölle ja poistetaan sieltä, kun sovellus lopetetaan, käyttäen viittaustyyppisessä `CRemoteObjectFactory`-luokassa toteutettuja staattisia metodeja `Open(CTrainingKeeper*, int)` ja `Close()`. Rekisteröimiseen ja poistamiseen käytetään staattisia metodeja, jotta näitä metodeja voidaan kutsua natiivista C++-koodista. Näiden metodien kutsumista varten on DLL:stä ulospäin näkyvät rajapintafunktiot `OpenRemoteConnection(CTrainingKeeper* pKeeper, int nPort)` ja `CloseRemoteConnection()`.

Metodissa `CRemoteObjectFactory::Open(CTrainingKeeper*, int)` otetaan käyttöön TCP-kanava (`System.Runtime.Remoting.Channels.Tcp.TcpChannel`) `int`-argumentin

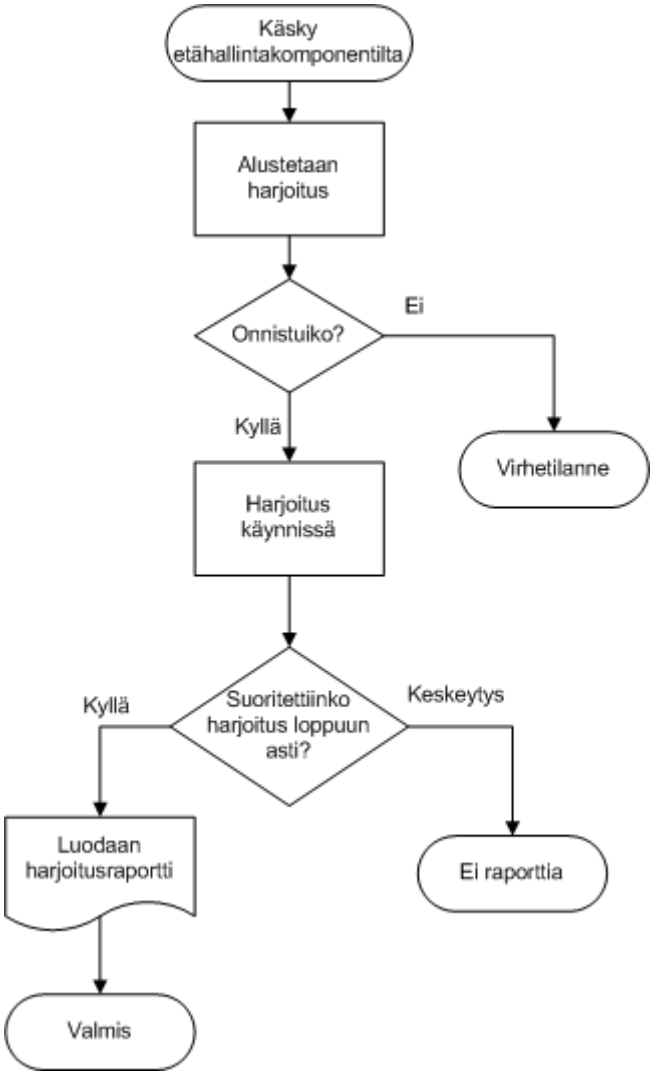
osoittamassa portissa. Käytettäväksi porttialueeksi valittiin IANA:n määrittelemä dynaaminen porttialue eli portit 49152- 65535.

Jotta kaikki .NET Frameworkin luokat voidaan automaattisesti marshalloida kanavaan, täytyy `TcpChannel`-luokan alustuksessa antaa `BinaryServerFormatterSinkProvider`-luokan olio, jonka `TypeFilterLevel`-ominaisuudeksi on asetettu `TypeFilterLevel.Full`. Tämä luokka kuuluu `System.Runtime.Remoting.Channels`-nimiavaruuteen. Luotu kanavaolio rekisteröidään Remoting-ajoympäristöön `System.Runtime.Remoting.Channels.ChannelServices`-luokan `RegisterChannel(IChannel, bool)`-metodilla.

Kanavan rekisteröimisen jälkeen luodaan `CRigSimRemoteObject`-luokan olio, jolle asetetaan osoitin `CTrainingKeeper`-olioon ja polku hakemistoon, johon tallennetaan asiakkaalta lähetettävät harjoitustiedostot. `CTrainingKeeper`-luokka on se osa simulaattorisovelluksesta, joka toteuttaa tilakoneen, jonka tilasiirtymät määräytyvät simulaattoriharjoituksen ajon perusteella. Tilojen muuttuminen on kuvattu kuvassa 5 vuokaaviona.

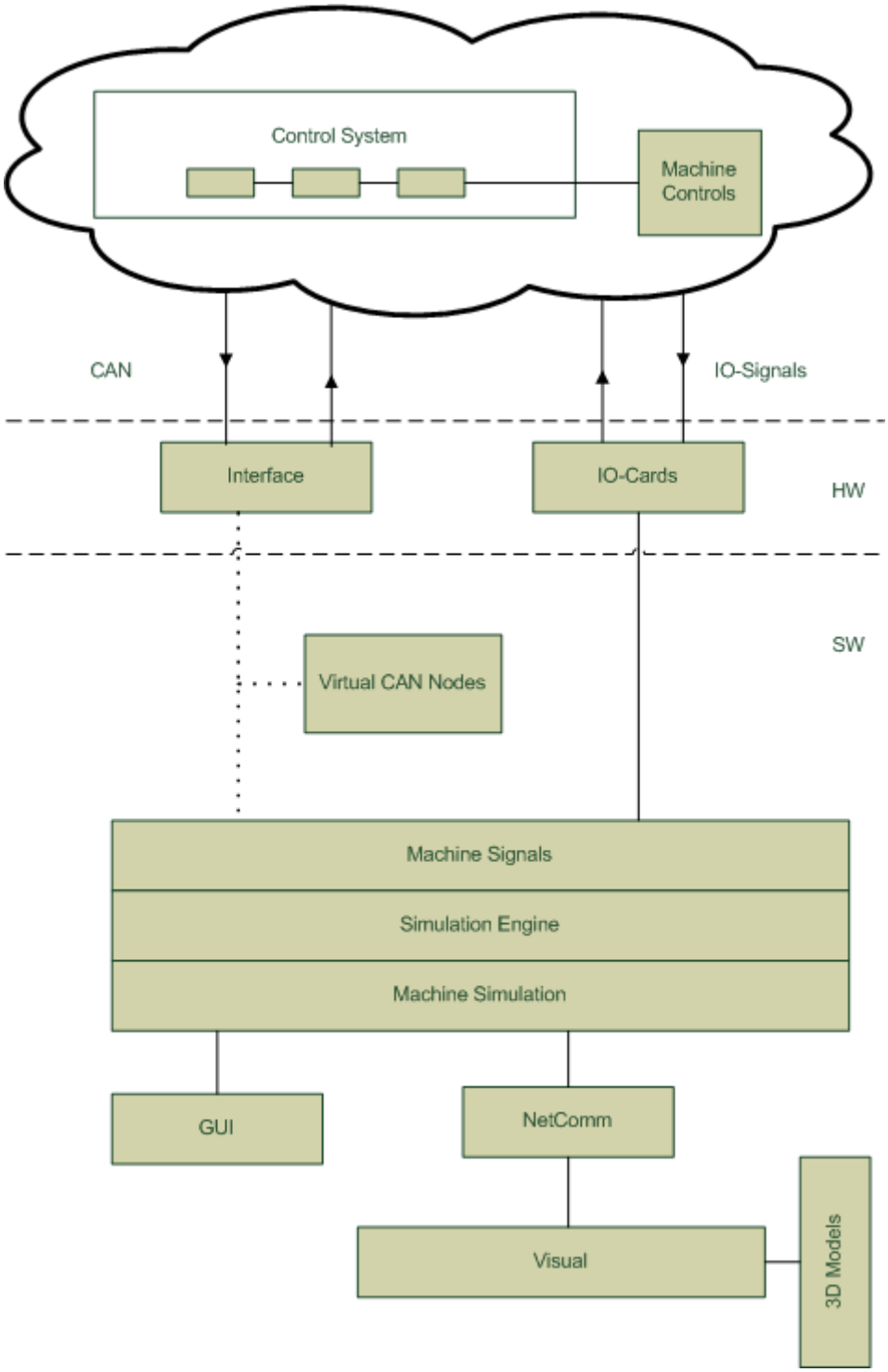
Seuraavana tämä olio rekisteröidään etäolioksi marshalloimalla se `System.Runtime.Remoting.RemotingServices`-luokan `Marshal(MarshalByRefObject, String)`-metodilla. Koska `CRigSimRemoteObject`-luokka periytyy `MarshalByRefObject`-luokasta, voidaan se antaa ensimmäisenä argumenttina ja `String`-argumentissa annetaan etäolion tunnisteena toimiva URL. Tämän metodikutsun jälkeen palvelinsovellukseen voidaan ottaa Remoting-yhteys.

`CRemoteObjectFactory` -luokan `Close()`-metodin kutsuminen lopettaa Remoting-metodikutsujen tulemisen etäoliolle ja vapauttaa varatun TCP-kanavan. Tätä metodia kutsutaan, kun palvelinsovellus suljetaan.




**Kuva 5. Vuokaavio simulaattoriharjoituksen ajosta.**

Etähallintakomponentin vaikutus simulaattorissa kohdistuu Machine Simulation -lohkoon, joka on esillä simulaattorin lohkokaaviokuvassa 6. Kuvan yläosassa pilven sisällä olevat komponentit ovat kuvassa 1 esillä olevat fyysiset ohjauslaitteet ja simuloitavan koneen oma ohjausjärjestelmä. Ohjausjärjestelmä on kytketty simulaattoriin CAN-väylällä ja IO-signaaleilla. Machine Signals, Simulation Engine ja Machine Simulation -lohkot muodostavat varsinaisen simulaattoriytimen. Näiden alapuolella on simulaattorin graafinen käyttöliittymä sekä Visual-järjestelmä, jolla luodaan koneen kuljettajan näkymä.




**Kuva 6. Simulaattorin lohkokaavio.**

## *7.5 Palvelinkoodin kääntäminen*

Palvelinsovelluksen lähdekoodista käännetään sekakoodikäännökseksi, mikä tarkoittaa sitä, että hallituksi koodiksi käännettäville lähdekooditiedostoille täytyy asettaa muista projektiin kuuluvista lähdekooditiedostoista poikkeavia asetuksia. Tämä tapahtuu Visual Studiossa avaamalla lähdekooditiedoston ominaisuudet ja muuttamalla seuraavia kohtia:

- C/C++ -> General -> Compile with CLR Support: Common Language Runtime Support /clr
- C/C++ -> Code Generation -> Basic Runtime Checks: Default
- C/C++ -> Code Generation -> Enable Minimal Rebuild: No
- C/C++ -> Code Generation -> Enable C++ Exceptions: Yes with SEH exceptions /EHa

Koska projektissa käytetään esikäännettyä otsikkotiedostoa, täytyy hallituille kooditiedostoille olla oma esikäännetty otsikkotiedosto, joka nimettiin stdafx_clr.cpp:ksi. Tälle tiedostolle asetettiin edellä mainittujen asetusten lisäksi esikäännettyyn otsikkotiedostoon liittyvät asetukset:

- C/C++ -> Precompiled Headers -> Create/Use precompiled headers: Create Precompiled header /Yc
- C/C++ -> Precompiled Headers -> Precompiled header file: $(IntDir)\$(TargetName)_clr.pch
- C/C++ -> Precompiled Headers -> Create/Use PCH Through file: stdafx_clr.h

Lähdekooditiedostoille asetetaan samat esikäännettyyn otsikkotiedostoon liittyvät asetukset yhdellä muutoksella:

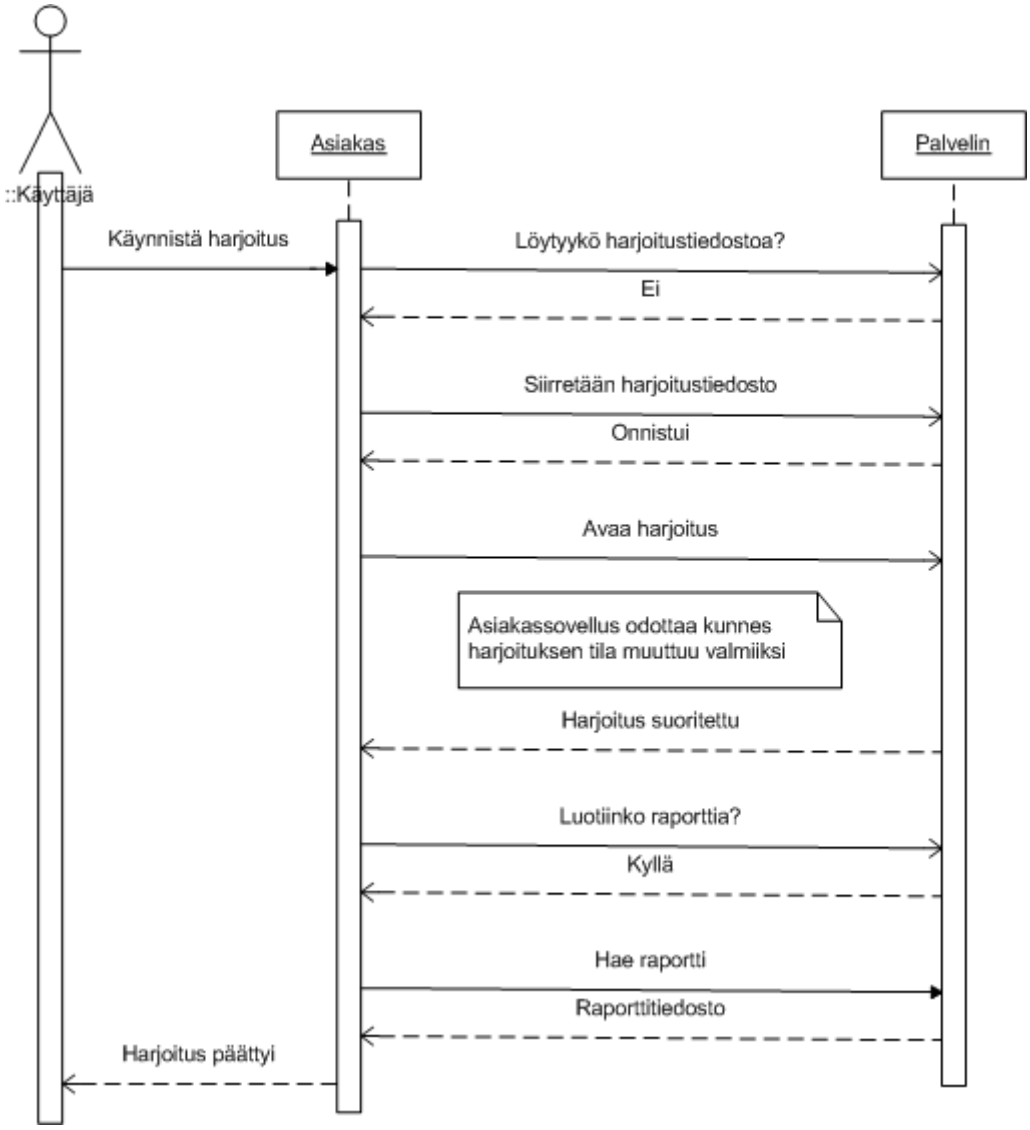
- C/C++ -> Precompiled Headers -> Create/Use precompiled headers: Use Precompiled header /Yu

## *7.6 Asiakas*

Asiakkaan eli koulutuksenhallintasovelluksen käynnistäessä simulaattoriharjoitusta täytyy ensimmäisenä alustaa kanava ja hakea etäolion proksi palvelinsovelluksesta. Asiakassovelluksessa täytyy palvelinsovelluksen tapaan asettaa kanavaolio tukemaan kaikkien .NET Frameworkin luokkien marshallointia. Asiakassovelluksessa voidaan asettaa kanavan portiksi nolla, jolloin Remoting-ajoympäristö valitsee automaattisesti käytettäväksi portiksi jonkin vapaan porttinumeron.

Käytettävän kanavan rekisteröimisen jälkeen haetaan etäolion proksiesiintymä palvelimelta käyttäen `System.Activator`-luokan staattista metodia `GetObject(Type, String)`. Tälle metodille annetaan argumentteina haettavan olion tyyppi ja etäolion yksilöivä URL-osoite. Tämä URL-osoite on oltava sama, kuin palvelinsovelluksessa etäolion rekisteröinnissä asetettu, jotta proksiesiintymä voidaan hakea palvelimelta. Haettavan proksiesiintymän tyyppi on `IRemoteObject`-rajapintaluokka eikä palvelinsovelluksessta toteutettu luokka, koska asiakassovelluksen kannalta riittää, että tiedetään käytettävissä oleva rajapinta. Tämän jälkeen jaetun rajapinnan tarjoamat toiminnot ovat asiakassovelluksen käytettävissä.

Kuvassa 7 on kuvattu sekvenssikaaviona harjoitustiedoston siirtäminen koulutuksenhallintasovelluksesta simulaattoriin, harjoituksen käynnistäminen ja harjoitusraportin tallentaminen. Koska kutsut etäoliolle estävät koodin suorituksen jatkamisen ja saattavat olla pitkäkestoisia, täytyy niitä varten luoda uusi säie, jossa kutsut etäoliolle tehdään, jotta käyttöliittymäsäie ei jumiudu. Vuorovaikutusta koulutuksenhallintasovelluksen käyttöliittymään ei käsitellä tässä työssä.




**Kuva 7. Harjoituksen käynnistämisen kuvaava sekvenssikaavio.**

# 8 Testaaminen

Toteutettua etähallintakomponenttia testattiin sen varsinaisessa käyttökohteessa eli koulutuksenhallintasovelluksessa, josta sillä käynnistettiin simulaattoriharjoituksia paikallisesti sekä lähiverkon yli SimDriller-työkonesimulaattorissa. Harjoitusajosta testattiin kuvassa 5 kuvatut loppumistilanteet. Testit suoritettiin Windows XP ja Windows Vista –käyttöjärjestelmissä. Suurin osa testeissä löydetyistä ongelmista liittyi koulutuksenhallintasovelluksen käyttöliittymäsäikeen ja etähallintakomponenttia kuuntelevan säikeen vuorovaikutukseen.

Testattaessa etähallintakomponenttia Windows Vista -käyttöjärjestelmässä havaittiin, että Windows Vista on varannut käyttöönsä dynaamisen porttialueen kuusi ensimmäistä TCP-porttia, joka estää niiden käyttämisen etähallintakomponentin porttina.

Testattaessa lähiverkon yli tapahtuvaa etäyhteyden muodostamisesta havaittiin, että simulaattori-PC:ssä F-Secure Internet Security -palomuurisovellus estää yhteyden muodostumisen. Tämä ongelma voitiin kiertää asettamalla palomuurista etähallintakomponentin käyttämä portti avoimeksi.

# 9 Jatkokehityssuunnitelmat

Työtä tehtäessä tuli esille muutama jatkokehitysajatus. Etähallintakomponenttiin lisätään toiminnallisuutta, jolla voidaan muutella ajon aikana simuloidun ohjausjärjestelmän tiloja siten, että ne voivat merkitä esimerkiksi vikaa simuloitavassa työkoneessa. Suunnitteilla on myös ominaisuus, jossa työkonesimulaattorin tuottamien raporttien kieliversioinnissa käytettävät kieliversiointitiedostot voitaisiin hakea työkonesimulaattoritietokoneelta etähallintakomponentin avulla käytettäväksi koulutuksenhallintasovelluksessa olevassa raporttinäkymässä.

# 10 Yhteenveto

Työn lopputuloksena valmistui etähallintakomponentti, joka toteutti sille määritetyt vaatimukset. Jatkokehityksen kannalta etähallintakomponentista tuli helposti laajennettava.

Etähallintakomponentti otettiin käyttöön SimDriller-työkonesimulaattorissa, jotta etähallintakomponentista saadaan käyttökokemuksia. Käyttökokemuksien myötä toivotaan myös tulevan uusia ajatuksia siitä, mitä tietoja halutaan simulaattoriajosta kerätä ja esittää koulutuksenhallintasovelluksessa.

Työn alussa Remoting-tekniikka ja C++/CLI-ohjelmointikieli olivat työn tekijälle täysin uusia tekniikoita, joten niiden tutkimiseen kului huomattavasti aikaa. Itse etähallintakomponentin toteuttaminen ei tuottanut missään vaiheessa suuria ongelmia ja sille asetetussa aikataulussa pysyttiin.

# Lähteet

1. Marcus Heege, Expert C++/CLI: .NET for Visual C++ Programmers. Apress. 2007. 330 s.
2. Microsoft Corporation. Common Type System Overview. [www-sivu]. [viitattu 3.11.2008]. Saatavilla: http://msdn.microsoft.com/en-us/library/2hf02550.aspx
3. Jeff Prosise, Programming Microsoft .NET. Microsoft Press. Redmond, Washington 2002. 773 s.
4. Inco Rammer ja Mario Szpuszta, Advanced .NET Remoting, Second Edition. Apress. United States of America, 2005. 579 s.