SAVONIA

Lange Server

Graafinen käyttöliittymä

Arto Karttunen

Opinnäytetyö

__.__.____ _____

Valitse kohde.

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Arto Karttunen	
Työn nimi Lange Server	
Päiväys 10.4.2012	Sivumäärä/Liitteet 29
Ohjaaja(t) Kalevi Kolehmainen, lehtori, Arto Toppinen, yliopettaja	
Toimeksiantaja/Yhteistyökumppani(t) Savonia-amk Tutkimus- ja kehitysyksikkö, Itä-Suomen yliopisto, Kuopion yliopistollinen sairaala, Mega-Elektroniikka Oy, Hoxville Oy ja Delfin Technologies Oy	

Tiivistelmä

Tämän opinnäytetyön aiheena oli ohjelmoida Lange-projektia C#-ohjelmointikielellä mittausdataa keräävä ja demonstroiva ohjelmisto. Lange-projektin tavoitteena oli kehittää IP-pohjainen langaton mittausjärjestelmäpohja ja aikasynkronointi käyttäen IEE1588 2008 standardia.

Työn tavoitteena oli saada kehitettyä toimiva ohjelmisto mittausdatan keräämiseen ja esittämiseen. Ohjelmiston tarkoitus on edesauttaa kerätyn datan analysointia ja keräämistä, jota voidaan hyödyntää erinäisissä sovelluksissa. Työn tavoitteisiin ei päästy suunnitelmien mukaisesti valitulla tekniikalla, mutta koodia pystyttiin hyödyntämään toisessa jatkokehityksessä.

Avainsanat
GUI, WPF, WCF, UDP, aika, synkronointi

Julkinen

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s)			
Title of Thesis Lange project: Lange Server			
Date	10.4.2012	Pages/Appendices	29
Supervisor(s) Mr Kalevi Kolehmainen, Principal Lecturer and Mr Arto Toppinen, Principal Lecturer			
Client Organisation /Partners Savonia University Of Applied Sciences research and development unit, University Of Eastern Finland, University Hospital Of Kuopio, Mega-Electronics Ltd, Hoxville Ltd, Delfin Technologies Ltd			

Abstract

The aim of this project was to develop as a part of the Lange-project, a program with graphical user interface for collecting and demonstrating data by using WPF, WCF techniques and C#-programming language. Aim of the Lange-project was to develop IP-based wireless measurement base and time synchronization using IEE1588 2008 standard.

The aim of the study was to develop a functional solution to collect and present the collected measurement data. The main use of the program was to help analyze and collect the gathered data which can be used in different kinds of applications. The aim of the study was not achieved with the chosen technique, but the code created was usable in further development.

Keywords
GUI, WPF, WCF, UDP, time, synchronization

Public

Alkusanat

Tämä työ tehtiin syksyllä 2011 Savonia-ammattikorkeakoulun tietotekniikan koulutusohjelman opinnäytetyönä. Työn ohjaajina toimi Savonia-ammattikorkeakoulun tietotekniikan lehtori Kalevi Kolehmainen ja lehtori Veijo Pitkänen sekä Savonian tutkimus- ja kehitysyksikön puolelta yliopettaja Arto Toppinen. Haluan kiittää ohjaajia, Savonia-ammatikorkeakoulun tutkimus- ja kehitysyksikön työtovereita, puolisoa sekä perhettä, jotka mahdollistivat työn valmistumisen.

Kuopio, 10. huhtikuuta 2012

Arto Karttunen

SISÄLTÖ

LYHENTEET, SYMBOLIT JA KÄSITTEET

WPF	Windows Presentation Foundation
WCF	Windows Communication Foundation
UDP	User Datagram Protocol
TCP	Transfer Control protocol
EEG	(electroencephalography) Aivosähkökäyrä
EKG	(elektrokardiografia) Sydänsähkökäyrä
Interface	Rajapinta
GUI	(Graphical User Interface) Graaffinen käyttöliittymä
DAQ	(Data acquisition) Tiedon hankinta
Bitti	Binäärinumero 1/0

- 1 bitti = 2^1 = 2 tilaa
- 2 bittiä = 2^2 = 4 tilaa
- 4 bittiä = 2^4 = 16 tilaa
- 8 bittiä = 2^8 = 256 tilaa
- 16 bittiä = 2^{16} = 65 536 tilaa
- 32 bittiä = 2^{32} = 4 294 967 296 tilaa
- 64 bittiä = 2^{64} = 18 446 744 073 709 551 616 tilaa

Integer(Int)	Kokonaisluku -/+-merkkinen
UInt	Unsigned Integer +-merkkinen kokonaisluku
Int64	64-bittinen kokonaisluku
MAC-osoite	(Media Access Control) laitteiston yksilöivä tunnus

1 JOHDANTO

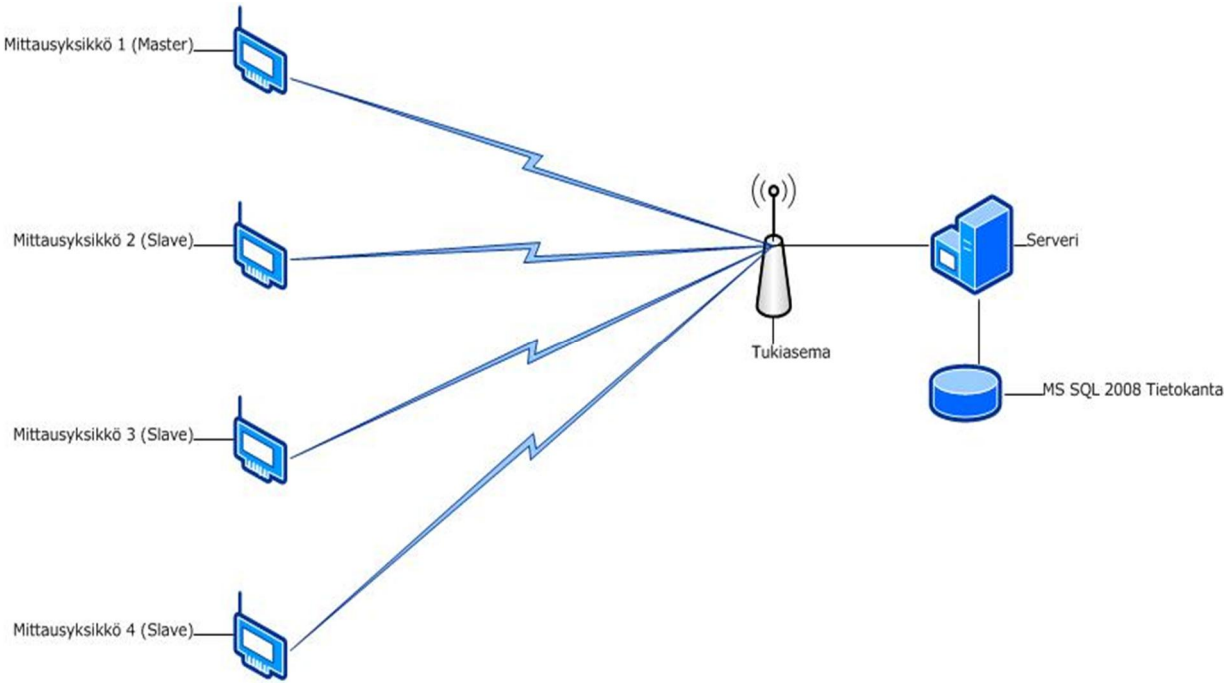
Tämä opinnäytetyön tavoitteena oli kehittää tiedonkeruuohjelmisto, joka mahdollistaa mittaustiedon tallentamisen tietokantaan sekä näyttää sen reaaliajassa käyttöliittymässä. Työ on tehty osana Lange-hanketta.[1]

Lange-hankkeen tarkoituksena on kehittää ja tutkia lääketieteelliseen käyttöön soveltuvaa monikanavaista, langatonta tiedonsiirtoa samanaikaisesti useasta anturista. Mittausyksiköiden tulee olla aikasynkronoitu toisiinsa vähintään 100µs tarkkuudella. Haasteena sovellukselle ovat akkukäyttöisten lähettimien tiedonsiirtorajoitus sekä riittävän aikatahdistustarkkuuden saavuttaminen.[1]

IP-pohjaiset langattomat tekniikat ovat kehittyneet nopeasti viime vuosina. Projektin tavoitteena on tuoda tämä tekniikka käyttöön lääketieteellisiin sovelluksiin, kuten esimerkiksi EEG ja EKG, joissa saavutettaisi suuri hyöty langattomuuden tuomasta edusta käytettävyyteen ja mittausmahdollisuuteen. Tällaisella sovelluksella voisi tarpeen mukaan aloittaa EEG ja EKG ottamisen potilaalta jo mahdollisesti ambulanssissa ja seurata aina osastolle asti ilman katkoja.[1]

2 LANGE-HANKE

Lange-hanke eli ”Langattomien sensoreiden käyttö lääketieteellisessä multiparametrimonitoroinnissa” on Tekes-rahoitteinen, Savonia-ammattikorkeakoulun, Itä-Suomen yliopiston, Kuopion yliopistollisen sairaalan, Mega-Elektroniikka Oy:n, Hoxville Oy:n ja Delfin Technologies Oy:n yhteistyöprojekti, jossa päätavoitteena on tutkia nykypäivän langattomien tekniikoiden käyttöä sensoriverkossa.[1]



Kuva 2.1 Esimerkki kuva sensoriverkosta

3 KÄYTETYT TEKNIIKAT

3.1 C#

C# (C sharp) on Microsoft-yhtiön .NET Framework:lle vuonna 2000 kehittämä ohjelmointikieli. C# on moderni, yksinkertainen sekä olio-pohjainen ja sen tavoitteena on ollut yhdistää C++:n tehokkuus ja Visual Basicin helppokäyttöisyys.

3.2 WPF

WPF (Windows Presentation Foundation) on vektorigrafiikkaan perustuva käyttöliittymäkirjasto Windows Client -sovellusten tekemiseen. WPF tarjoaa monia etuja verrattuna vanhempaan Winfors-tekniikkaan. Käyttöliittymän ulkoasu voidaan erottaa kokonaan koodista ja ulkoasun kuvaamiseen voidaan käyttää erillisiä tyylimäärittelyjä. Datan sidonta voidaan tehdä myös monella erilaisella tavalla. Animaatioiden toteuttaminen ei tarvitse lähdekoodin kirjoittamista lainkaan, vaan se voidaan kirjoittaa XAML-kielellä muun käyttöliittymä koodin joukkoon.[2]

3.3 WCF

WCF (Windows Communication Foundation) WCF muodostaa Vistan sisäisen viestijärjestelmän, joka tunnettiin aikaisemmin työnimellä *Indigo*. WCF yhdenmukaistaa muun muassa Windows käyttöjärjestelmän loki-, tapahtuma- ja ongelmatilanneviestikäytännöt ja luo ohjelmille yhtenäisen käytännön välittää tietoa keskenään. WCF korvaa aiemmat tekniikat, kuten COM/DCOM,. NET Remoting ja ASP.NET Web palvelut. WCF tarjoaa helppokäyttöisen hajautuksen sovelluskehityksessä. Ohjelmassa rajapintojen toteutus tehtiin WCF:llä. [3]

3.4 XAML

XAML (Extensive Application Markup Language) XAML on rakenteellinen kuvauskieli, joka pohjautuu XML-kieleen. XAML-kieltä käytetään esimerkiksi WPF-sovelluksissa käyttöliittymän ulkoasun määrittelyyn. Käyttöliittymän toiminnallisuus määritetään erikseen C#-ohjelmointikielellä omassa määritellyssä tiedostossaan. [4]

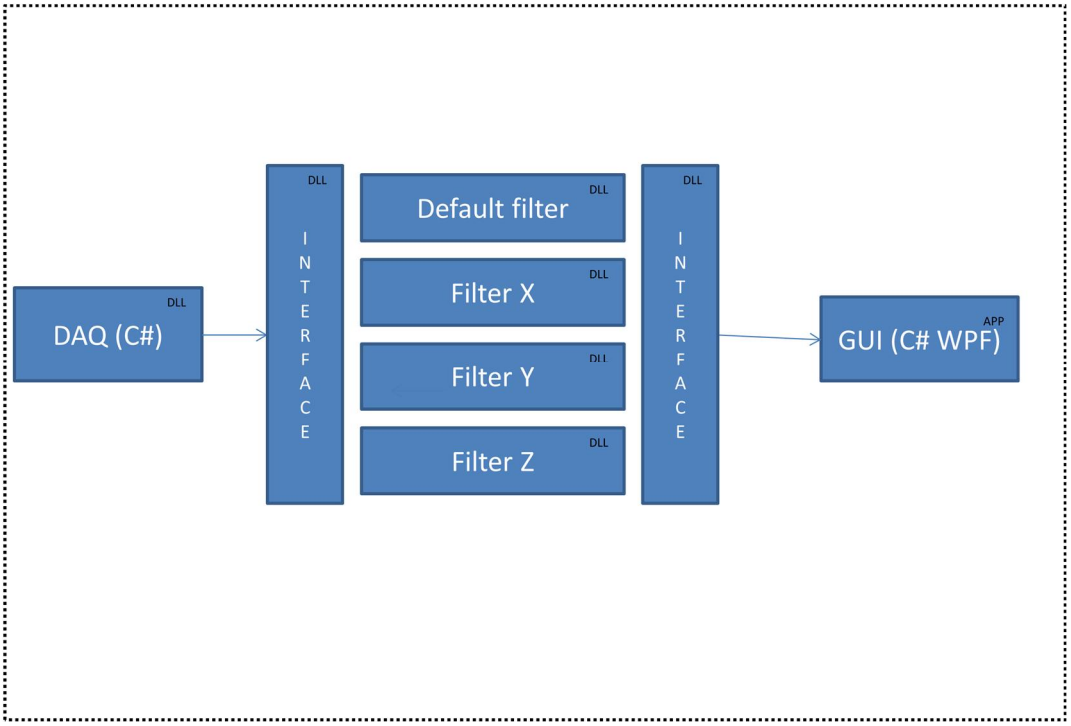
4 SUUNNITTELU

Projektin suunnitteluvaiheessa tehtiin suunnitelma projektin kulusta. Suunnitelma sisälsi käytettävät tekniikat, työvaiheet, aikataulutuksen sekä tavoitteet. Suunnitelman alussa perehdyttiin käytettäviin menetelmiin ja tutkittiin niiden soveltuvuutta työn tarpeisiin. Ohjelmiston kehitystä seurattiin viikoittaisessa palaverissa, jossa projektiin osallistuvat projektityöntekijät esittelivät saatuja tuloksia sekä työn edistymistä.

Asiakkaiden vaatimukset sovelluksen suhteen oli mahdollistaa sensoriverkon toiminnan testaus. Lisäksi toivomuksena oli, että mahdollinen serveri- ja bitti-paketin purku olisi erillisissä komponenteissa, jolloin komponentteja voitaisiin käyttää asiakkaiden omassa, tähän tarkoitukseen olemassa olevassa tiedonkeruuohjelmistossa.

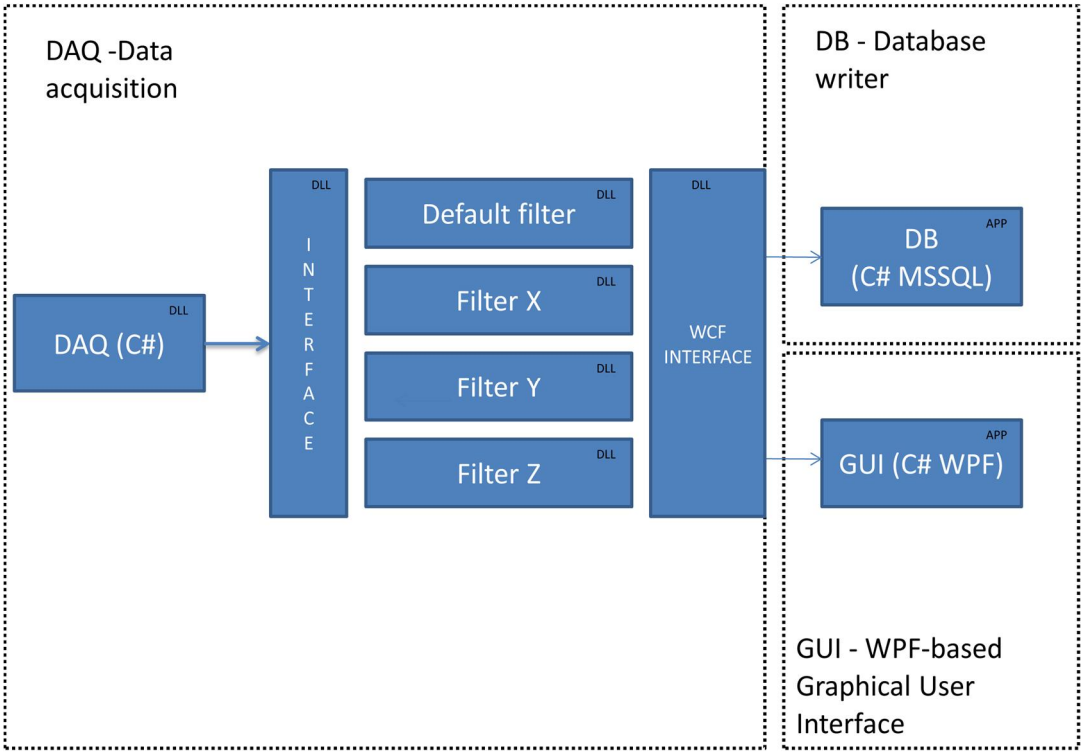
Haasteen suunnittelussa aiheutti mittaustulosten paljous. Mittaustilanteessa jokaiselta mittausyksiköltä saapuu yksi paketti 4 millisekunnin välein. Tämän lisäksi jokaisessa mittauspaketissa on kahdeksan mittauskanavan 32 mittaustulosta. Mittausyksiköiden määrä on maksimissaan 4 kyseisessä pilottiversiossa. Tämän tietomäärän näyttäminen reaaliajassa varaamatta liika tietokoneen muistia ja prosessoritehoa sekä samanaikainen sulavan datan näyttäminen olisi haasteellisinta ohjelman kehityksessä.

Ohjelmistosta päätettiin tehdä kolme kehitysversiota, joille asetettiin valmistumisaikataulu kehityksen varrella. Aluksi luotiin yksinkertainen versio, jossa ei ollut mukana tietokantaa ja tälle asetettiin valmistumistavoitteeksi 31.6.2011 (Kuva 4.1).



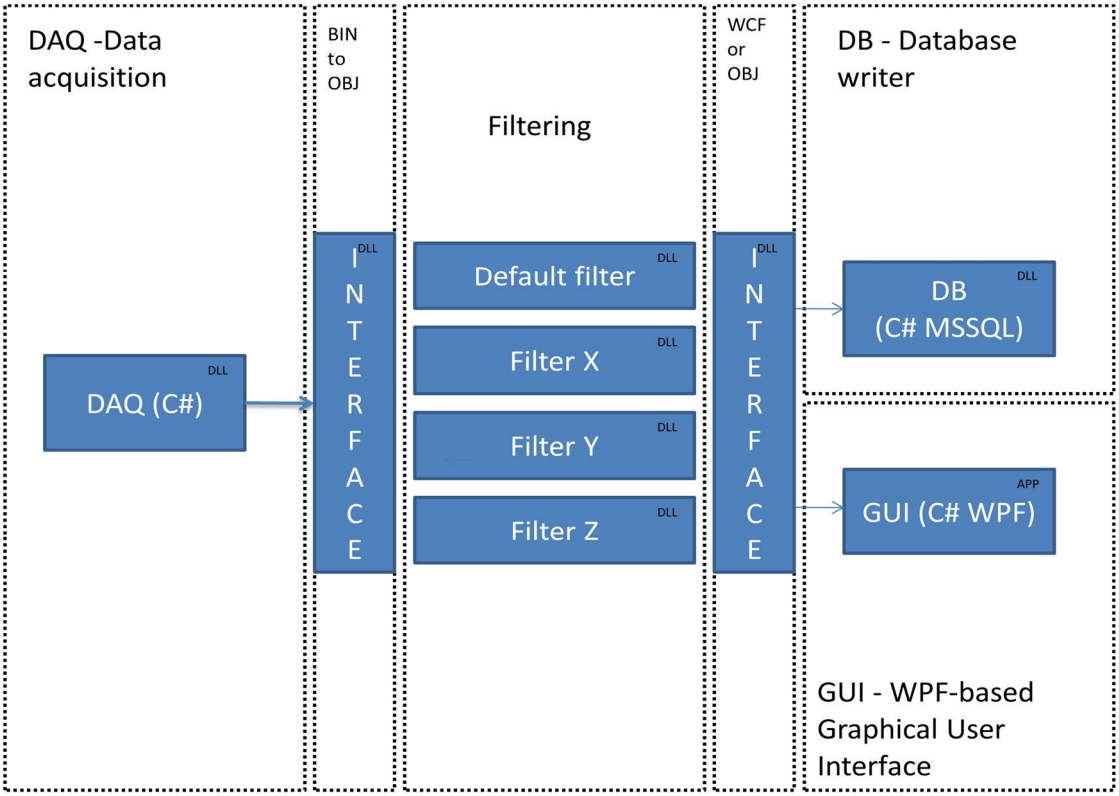
Kuva 4.1 Ensimmäinen testiversio 31.6.2011

Sovelluksen ensimmäinen versio ja työjärjestys suunniteltiin toteutettavaksi vasemmalta oikealle, kuten sovelluksen toimintakin. DAQ-komponentilla tarkoitetaan itse serveriä, rajapinnat ja filter-osa muodostavat oman komponentin ja GUI käyttöliittymän. Sovellus suunniteltiin jaettavaksi loogisiin kokonaisuuksiin, jolloin mahdollisten virheiden jäljitys ja sovelluksen kehityksessä tapahtuvat muutokset tai osien käyttö olisi helpompaa.



Kuva 4.2 Toinen testiversio 15.7.2011

Toinen versio, jossa poikkeuksena edelliseen versioon oli tietokanta toiminnallisuus ja WCF-rajapinta filtteri-osan jälkeen.



Kuva 4.3 Valmis ohjelma

Valmiissa ohjelmassa tietokanta-sovellus on muutettu dll-tiedostoksi mahdollisen rasitteen vähentämiseksi. Sen lisäksi tietokannasta piirto suunniteltiin kuvasta poiketen käyttöliittymän ja tietokanta-osan eikä WCF-rajapinnan kautta tehtäväksi.

5 OHJELMAN TOTEUTUS

5.1 Työkalut ja tekniikat

Sovelluksen kehityksessä käytettiin ohjelmistona Visual Studio 2010 Ultimatea ja tietokannaksi valittiin MS SQL 2008. Tässä vaiheessa mittausyksiköiden ohjelmisto ei ollut valmis, jotta sillä olisi saatu kunnollisia mittaustuloksia testausta varten. Siksi testausta varten luotiin bittipaketti, jota lähettämällä pyrittiin simuloimaan sensoriverkon tietoliikennettä lähettämällä usealla clientilla samankokoista bittipakettia kuin oikeassa tilanteessa.

WPF-käyttöliittymään päätettiin käyttää valmista kirjastoa piirtämiseen, koska muutamilla itse ohjelmoiduilla kirjastoilla ei päästy tarvittavaan nopeuteen. Valmis kirjasto D3(Dynamic Data Display) löytyi Microsoftin CodePlex vapaan lähdekoodin sivustolta. Luokkakirjaston käyttö oli sallittua tässä tapauksessa, koska kyseessä oli ei kaupallinen sovellus.

5.2 Serveri

Serverin toteutus aloitettiin perehtymällä vaatimuksiin sekä tutkimalla eri toteutustapoja, koska toteutuksesta ei ollut aikaisempaa kokemusta.

5.2.1 UDP Serveri ja Client

Serverin toimintaprotokollaksi valittiin UDP (User Datagram Protocol), joka oli käyttötarkoitukseen soveltuvampi kuin TCP (Transmission Control Protocol). UDP:ssä on useita eroavaisuuksia TCP:hen verrattuna. UDP:ssä ei esimerkiksi ole pakettien perille menon varmistusta. UDP ei myöskään käytä alkukättelyä eikä kolmivaiheista yhteyden lopettamista. Siten UDP:n yleisrasitteen arveltiin olevan pienempi kuin vaihtoehtoisessa TCP:ssä.

Serverille ei suunniteltu muuta tehtävää, kuin ottaa vastaa mittauslaitteista tuleva bittipaketti ja tarjota sitä eteenpäin. Bittipaketin käsittely tapahtui ohjelman seuraavassa vaiheessa. Sovelluksen testausta varten serveriin ohjelmoitiin myös client-ominaisuus, jolla pystyttiin testaamaan serveriä, tietokantaa ja piirtoa sekä muita ohjelman ominaisuuksia tarpeen mukaan.

```
//Luodaan UDPServer
public void DoServer()
{
    InitBuffers();
    AsyncCallback onReceiveFrom1 = new AsyncCallback(OnReceiveFrom);
    Socket socketA1
        = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
    EndPoint bindEndPointA1 = new IPEndPoint(IPAddress.Any, listenOnPort);
    // Sidotaan portti kuuntelemaan unicast-lähetystä.
    socketA1.Bind(bindEndPointA1);
    // Aloitetaan kuuntelu
    ReceiveFromData rfd = new ReceiveFromData(ref receiveBuffer1, receiveBuffer1.Length,
        ref bindEndPointA1, ref onReceiveFrom1, ref socketA1, null, this);
    rfd.BeginReceiveFrom();

}
```

Kuva 5.1 Serverin toteuttava koodi

(Kuva 5.1) UDP-serverin pääasiallinen koodi, jossa luodaan serverin toiminnallisuus

```
void DoClient()
{
    //Alustetaan puskurit
    InitBuffers();
    //Määritellään socketti
    Socket socketA1
        = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
    IPAddress sendTo = IPAddress.Parse(sendToAddr);

    EndPoint sendEndPointA1 = new IPEndPoint(sendTo, sendToPort);
    {
        AsyncCallback onReceiveFrom1 = new AsyncCallback(OnReceiveFrom);
        listenOnPort = 0;// sendToPort + 1;

        EndPoint bindEndPointA1 = new IPEndPoint(IPAddress.Any, listenOnPort);
        socketA1.Bind(bindEndPointA1);

        ReceiveFromData rfd = new ReceiveFromData(ref receiveBuffer2, receiveBuffer2.Length,
            ref bindEndPointA1, ref onReceiveFrom1, ref socketA1, null, this);
        rfd.BeginReceiveFrom();
    }
    //Lähetetään dataa messageCount muuttujassa määritelty määrä
    for (byte i = 0; i < messageCount; i++)
    {
        socketA1.SendTo(rawData, SocketFlags.None, sendEndPointA1);
        MyWriteLine("buffer " + i + " sent");
        Thread.Sleep(4);
    }
}
```

Kuva 5.2 Clientin toteuttava koodi

Client-toimintoa (Kuva 5.2) ei vaadittu sovellukseen, mutta se päätettiin toteuttaa, koska se oli erinomainen apu testauksessa mittausyksiköiden ohjelmoinnin ollessa kesken ja valmistuessa vasta huomattavasti myöhemmin.

```
namespace DAQTestConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            UDP_Server us = new UDP_Server();

            if ("client".CompareTo(args[0]) == 0)
            {
                us.sendToAddr = args[1];
                us.sendToPort = int.Parse(args[2]);
            }
            else
                if ("server".CompareTo(args[0]) == 0)
                {
                    us.IsServer = true;
                    us.listenOnPort = int.Parse(args[1]);

                }
                else
                {
                    Console.WriteLine("Käyttö: DaqTestConsole.exe client ip-osoite portti");
                    Console.WriteLine("Käyttö: DaqTestConsole.exe server kuunneltava portti");
                    return;
                }

            Console.WriteLine(args[0]);

            us.Run();
        }
    }
}
```

Kuva 5.3 Tekstipohjaisen käyttöliittymän käynnistys

Serverin ja Clientin pystyy käynnistämään komentokehotteen kautta, esimerkiksi clientin ”DaqTestConsole client 127.0.0.1 5555” ja serverin ”DaqTestConsole server 5555”.

5.3 Filtter

Filtter-osa (Kuva 4.1) sisältää ohjelmasta mittauslaitteilta tulevan bittidatan muuntamisen muuttujiksi omassa komponentissa. Komponentin tarkoituksena oli mahdollisesti suodattaa mittauksia, mutta koska tästä ei ollut tarkempia vaatimuksia toteutushetkellä, sillä ei ollut muuta tehtävää kuin muuttaa bittidata (Kuva 5.7, 5.8) muuttujiksi luokkamuuttujaan (Kuva 5.4, 5.5, 5.6) ja tarjota sitä pakettina eteenpäin.

```
public UdpMeasPacket OpenUDP(byte[] udpPacket, int MeasCount)
{
    int index = 0;
    UdpMeasPacket tmp = new UdpMeasPacket();
    MeasCount = 32;

    try// Muutetaan byte-tauluna tuleva data tarvittavaan, tehdään tarvittavat
    {  // bittisiirrot sopivissa Parse-metodeissa ja otetaan muuttujat talteen luokkamuuttujaan
        tmp.measHeader = new MeasHeader();
        tmp.measHeader.adcIdentifier = Parse8(udpPacket, ref index);
        tmp.measHeader.protocol_Identifier = Parse8(udpPacket, ref index);
        tmp.measHeader.sampling_freq = Parse16(udpPacket, ref index);

        for(int im = 0; im < 6; im++)
        {
            tmp.measHeader.SenderMAC += (Parse8(udpPacket, ref index) << (6-im));
        }

        tmp.measHeader.timestamp = new TimeInternal();
        tmp.measHeader.timestamp.seconds = Parse32(udpPacket, ref index);
        tmp.measHeader.timestamp.nanoseconds = Convert.ToInt32(Parse32(udpPacket, ref index));

        MeasData mData = new MeasData();

        for (int sample = 0; sample < 32; sample++)
        {
            MeasSample mSample = new MeasSample();

            mSample.statusBits = ParseStatus(udpPacket, ref index);

            for (int channel = 0; channel < 8; channel++)
            {
                mSample.measChannelList.Insert(channel, Parse24(udpPacket, ref index));
            }
            mData.measSampleList.Insert(sample, mSample);
        }
        tmp.measData = mData;
        return tmp;
    }
    catch (Exception ex)
    {
        System.Console.Out.WriteLine(ex.Message, ex.TargetSite.ToString());
        return tmp;
    }
}
```

Kuva 5.4 tavutaulun datan muunto muuttujiksi luokkamuuttujaan

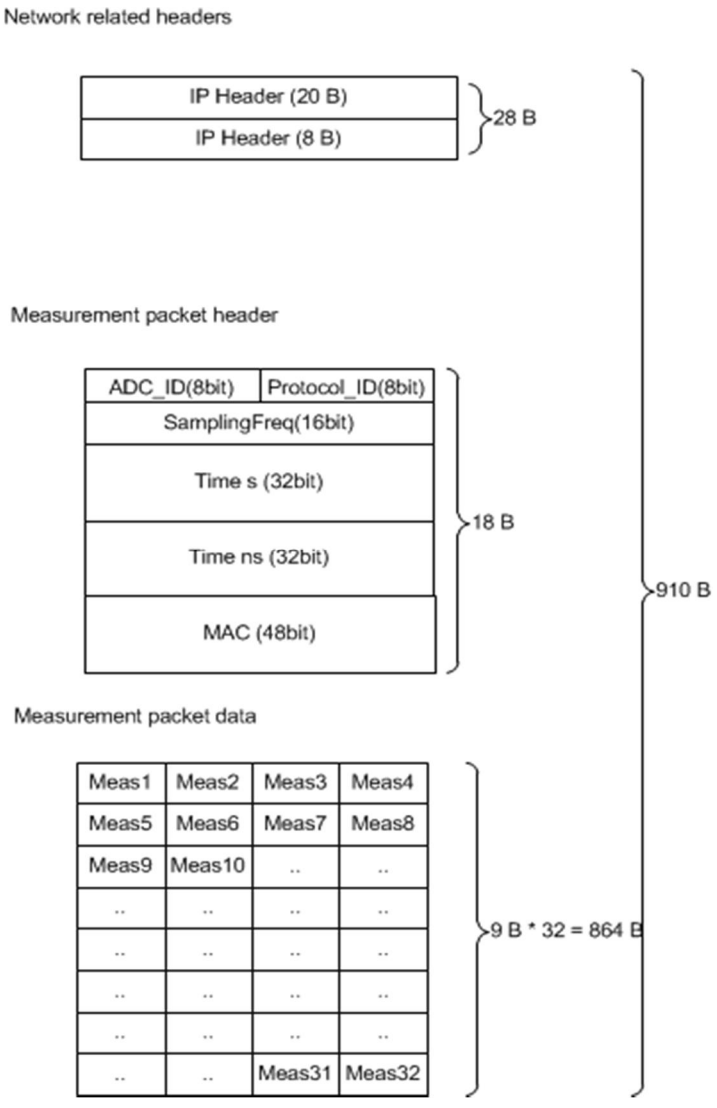
```
UInt64 Parse64(byte[] udpPacket, ref int index)
{
    UInt64 Parse64 = (UInt64)(udpPacket[index++] * 72057594037927936) + (UInt64)(udpPacket[index++] * 281474976710656) +
        (UInt64)(udpPacket[index++] * 1099511627776) + (UInt64)(udpPacket[index++] * 4294967296) +
        (UInt64)(udpPacket[index++] * 16777216) + (UInt64)(udpPacket[index++] * 65536) +
        (UInt64)(udpPacket[index++] * 256) + (UInt64)(udpPacket[index++]);
    return Parse64;
}
```

Kuva 5.5 Esimerkki 64-bittisestä Parse64 metodista

Esimerkkinä (Kuva 5.5) bittipaketista saatavan SendersMac 64-bittisen muuttujan arvon muunto bittidatasta 64-bittiseksi kokonaislukumuuttujaksi. Bittipaketti tulee käsittelyyn metodille, jossa paketti käsitellään Kuvan 5.4 mukaisessa järjestyksessä. Jokaisen luokkamuuttujan sisäisen muuttujan bittiarvo muunnetaan Parse8-, Parse16-, Parse32- tai Parse64-metodeissa kokonaisluvuksi.

```
public class UdpMeasPacket...
public class MeasHeader
{

    public byte adcIdentifier { get; set; }             // u8 adcIdentifier;
    public byte protocol_Identifier { get; set; }       // u8 protocol_identifier;
    public UInt16 sampling_freq { get; set; }           // u16 sampling_freq;
    public Int64 SenderMAC { get; set; }                // Lähettäjän MAC-osoite
    public TimeInternal timestamp = new TimeInternal(); // 2 x 32u kaksiosainen aikaleima

}

public class MeasData
{
    public List<MeasSample> measSampleList = new List<MeasSample>();
}

public class MeasSample...

public class TimeInternal...

public class StatusBits...
}
```

Kuva 5.6 Luokkamuuttuja kerättävää dataa varten

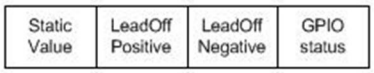


Kuva 5.7 Vastaanotettavan bittipaketin sisältämä data [5]

Measurement

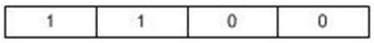
STATUS	(24 bits)	=	Status bits are descriped in page "Status"
CH1	(24 bits)	=	Measurement value from channel 1
CH2	(24 bits)	=	Measurement value from channel 2
CH3	(24 bits)	=	Measurement value from channel 3
CH4	(24 bits)	=	Measurement value from channel 4
CH5	(24 bits)	=	Measurement value from channel 5 (zeros if ADS 1294)
CH6	(24 bits)	=	Measurement value from channel 6 (zeros if ADS 1294)
CH7	(24 bits)	=	Measurement value from channel 7 (zeros if ADS 1294 or ADS1296 is used)
CH8	(24 bits)	=	Measurement value from channel 8 (zeros if ADS 1294 or ADS1296 is used)

Status bits (24 bits)



Static value	(4 bits)	=	Is always static
Lead Off Positive	(8 bits)	=	Status positive electrodes
Lead Off Negative	(8 bits)	=	Status negative electrodes
GPIO status	(4 bits)	=	Gives the status of gpio pins 1 to 4…

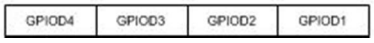
Static value



Lead Off Positive

Lead Off Negative

GPIO Status



Kuva 5.8 Tarkennus bittipaketin mittausdata osuudesta [5]

5.4 WCF-rajapinta

WCF-rajapinnat ohjelmoitiin komponenttien, kuten filttereiden, tietokannan ja käyttöliittymän välille. WCF-rajapinnalla mahdollistettiin ohjelman eri komponenttien käytettävyys muissa sovelluksissa sekä ohjelman jatkokehitys muokkaamatta itse ohjelmaa. Lisäksi asiakkaan tarkoitus oli käyttää sovelluksessa omaa käyttöliittymää.

```
namespace DaqWCF
{

    [ServiceContract]
    public interface IUdpMeasPacket
    {

        [OperationContract]
        void UdpMeasurements(UdpMeasPacket measurement);

    }

    [DataContract]
    public class UdpMeasPacket
    {

        [DataMember]
        public UdpMeasPacket measurement{ get; set; }
    }

}
```

Kuva 5.9 Esimerkki WCF-rajapinta

5.5 DAQ GUI

Käyttöliittymän toteutus päädyttiin tekemään WPF:llä, jossa käyttöliittymän ulkoasu suunniteltiin XAML-ohjelmointikielellä ja toiminnallisuus C#-ohjelmointikielellä. Piirtämisen toteutuksesta tehtiin joitakin eri versioita, joiden jälkeen päädyttiin käyttämään Dynamic Data Display eli D3-kirjastoa.

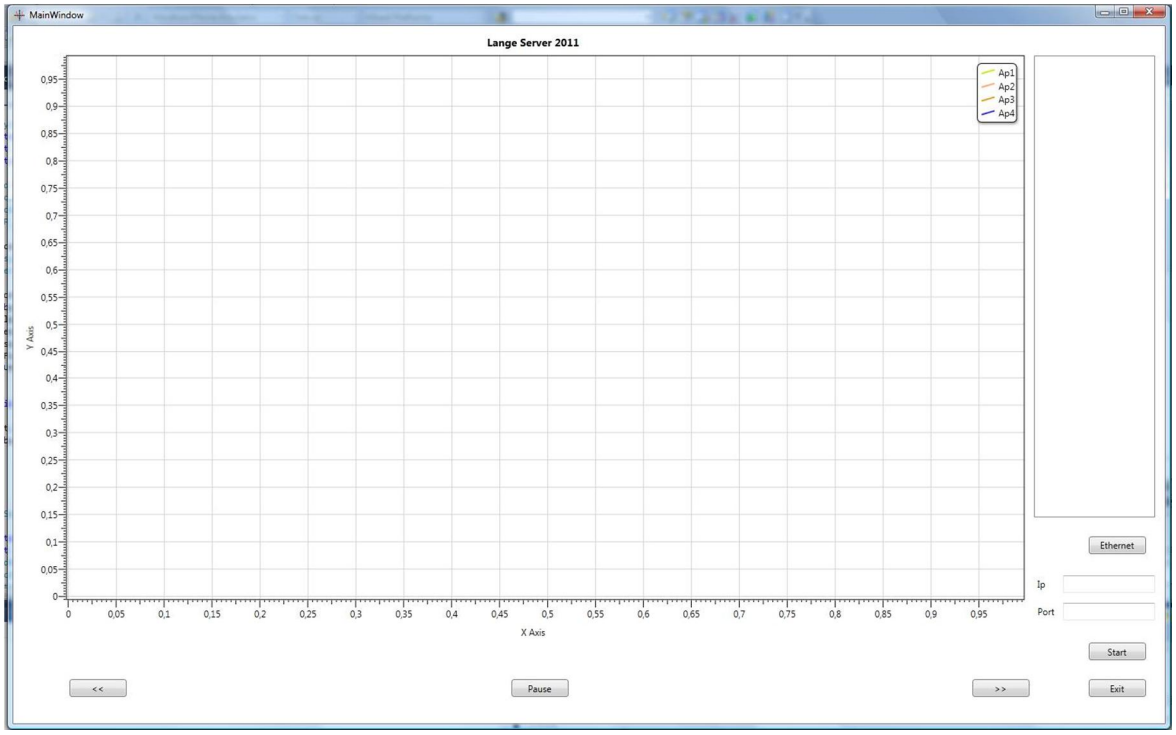
```
<Button
    Content="&lt;&lt;"
    Height="23"
    HorizontalAlignment="Left"
    Margin="130,592,0,0"
    Name="BtnBackward"
    VerticalAlignment="Top"
    Width="75"
    Click="BtnBackward_Click"
    DataContext="{Binding}"
/>
```

Kuva 5.10 Esimerkki painikkeen määritys XAML-ohjelmointikielellä

D3-kirjastossa on valmiina piirtoplotteri, joka ottaa vastaan Point-objektin. Ensimmäinen piste saa x-arvonsa datapaketissa olevasta aikaleimasta (UInt32 seconds ja UInt32 nanoseconds) kun y-arvo taas saadaan datapaketin mittausdataosasta kanavakohtaisesti. Loput 31 mittauspistettä saavat x-arvonsa datapaketin aikaleiman sekä paketin sisältävän mittaustaajuuden tulosta (UInt16 SamplingFreq).

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Create first source
    source1 = new ObservableDataSource<Point>();
    // Set identity mapping of point in collection to point on plot
    source1.SetXYMapping(p => p);
    // Create second source
    source2 = new ObservableDataSource<Point>();
    // Set identity mapping of point in collection to point on plot
    source2.SetXYMapping(p => p);
    // Create third source
    source3 = new ObservableDataSource<Point>();
    // Set identity mapping of point in collection to point on plot
    source3.SetXYMapping(p => p);
    // Create fourth source
    source4 = new ObservableDataSource<Point>();
    // Set identity mapping of point in collection to point on plot
    source4.SetXYMapping(p => p);
    // Add all graphs.
    plotter.AddLineGraph(source1, 2, "Ap1");
    plotter.AddLineGraph(source2, 2, "Ap2");
    plotter.AddLineGraph(source3, 2, "Ap3");
    plotter.AddLineGraph(source4, 2, "Ap4");
    // Start computation process in second thread
}
```

Kuva 5.11 Esimerkki piirturin alustamisesta neljälle viivalle.



Kuva 5.12 Käyttöliittymä Lange Server 2011

(Kuva 5.12) Käyttöliittymän D3D-piirturi alustettuna neljälle viivalle, IP-osoitteen ja portin syötön jälkeen kuuntelun aloitus reaaliajassa.

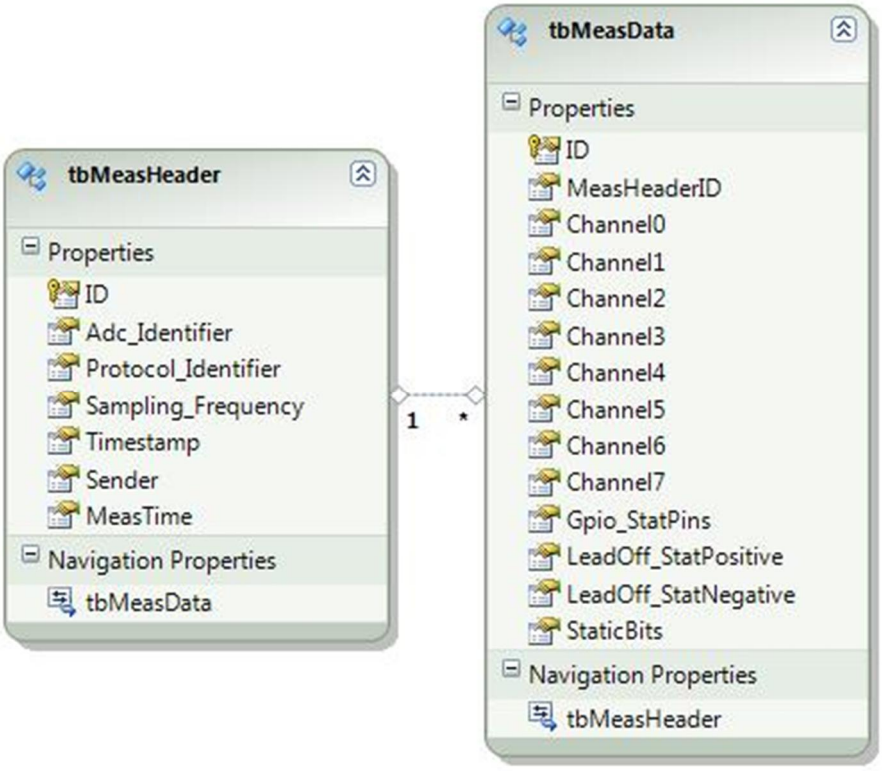
5.6 Tietokanta

Ohjelman tietokantana päätettiin käyttää MS SQL 2008. Tietokannan tarkoituksena oli mahdollistaa tulosten tarkastelu jälkikäteen käyttöliittymän kautta. Tietokannasta lukemista ei toteutettu loppuun asti, koska WPF-tekniikka ei soveltunut käyttötarkoitukseen hitaudestaan johtuen.

```
public void WriteToDatabase(UdpMeasPacket measurements)
{
    try
    {

        langedb = new LangeDBEntities();
        measH = new tbMeasHeader();
        int i = 0;

        measH.Adc_Identifier = measurements.measHeader.adcIdentifier;
        measH.Protocol_Identifier = measurements.measHeader.protocol_Identifier;
        measH.Sampling_Frequency = Convert.ToInt16(measurements.measHeader.sampling_freq);
        measH.Sender = measurements.measHeader.SenderMAC; // Senders partial MAC-address
        //measH.Time_Second = Convert.ToInt32(measurements.measHeader.timestamp.seconds);
        //measH.Time_Nanosecond = measurements.measHeader.timestamp.nanoseconds;
        decimal time = measurements.measHeader.timestamp.seconds;
        time += (decimal)(measurements.measHeader.timestamp.nanoseconds / 1000000000M);
        measH.MeasTime = time;
        measH.Timestamp = DateTime.Now;

        langedb.AddTotbMeasHeader(measH);
        langedb.SaveChanges();

        for (i = 0; i < 32; i++)
        {
            measD = new tbMeasData();

            measD.Channel0 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[0]);
            measD.Channel1 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[1]);
            measD.Channel2 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[2]);
            measD.Channel3 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[3]);
            measD.Channel4 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[4]);
            measD.Channel5 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[5]);
            measD.Channel6 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[6]);
            measD.Channel7 = Convert.ToInt32(measurements.measData.measSampleList[i].measChannelList[7]);
            measD.Gpio_StatPins = measurements.measData.measSampleList[i].statusBits.Gpio_StatusPins;
            measD.LeadOff_StatPositive = measurements.measData.measSampleList[i].statusBits.LeadOff_StatPositive;
            measD.LeadOff_StatNegative = measurements.measData.measSampleList[i].statusBits.LeadOff_StatNegative;
            measD.StaticBits = measurements.measData.measSampleList[i].statusBits.StaticBits;
            measD.tbMeasHeader = measH;


            langedb.AddTotbMeasData(measD);
        }
        langedb.SaveChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message, ex.TargetSite.ToString());
    }
```

Kuva 5.13 Tietokantaan kirjoitus

Mittausyksiköiltä tulevan datan lisäksi tietokantaan tallennettiin kirjoitushetkellä saatava aikaleima Timestamp -sarakkeeseen, jota voidaan käyttää mittauksen tarkastelussa (Kuva 5.13, Kuva 5.14).



Kuva 5.14 Tietokanta rakenne Lange

Tietokannan rakenne säilyi hyvin yksinkertaisena (Kuva 5.14). Yhtä tbMeasHeader-taulun riviä kohti tallentuu 32 tbMeasData-taulun riviä. Tuloksia voidaan hakea joko Laitteen Mac-osoitteen perusteella tai ajan mukaan, joka tallentuu tietokantaan kirjoituksen yhteydessä. tbMeasHeader-taulun ID toimii vierasavaimena tbMeasData-taulussa.

6 TESTAUS

6.1 Testaus suunnitelma

Tässä kappaleessa perehdytään ohjelmiston testaukseen erilaisilla testaustyökaluilla sensoriverkossa. Testaussuunnitelmia ei ehditty tehdä kiireellisyyden vuoksi. Sovellusta testattiin kuitenkin jatkuvasti kehityksen aikana ja lähes jokaisen muutoksen jälkeen testausta varten kehitetyllä clientillä.

6.2 Visual Studio 2010 Ultimate testaustyökalut

Testauksessa käytettiin Visual Studion tarjoamia monipuolisia testaustyökaluja, joita pystyi helposti ajamaan pienenkin muutoksen jälkeen. Testaustyökalut osoittautuivat käytännöllisiksi esimerkiksi bittidatan parsimisesta vastaavien metodien testaukseen.

6.3 Testaus clientilla

Serverin ohjelmointivaiheessa toteutettiin serveriin client-ominaisuus, jonka avulla pystyttiin nopeasti ja ilman valmisteluja testaamaan ohjelmaa eri toteutusvaiheissa. Clientilla lähetettiin vakiokokoista bittipakettia, jolloin piirtämisen testaukseen tarvittava aika, eli x-arvo jouduttiin tekemään kasvattamalla muuttujaa piirturin silmukassa. Muuten client-ominaisuus osoittautui erittäin hyväksi testivälineeksi.

6.4 Testaus sensoriverkossa

Sensoriverkon mittausyksiköiden ohjelmoinnin keskeneräisyyden vuoksi ohjelmistoa ei pystytty testaamaan neljällä mittausyksiköllä vaan tyydyttiin testaukseen yhdellä mittausyksiköllä sensoriverkossa.

7 YHTEENVETO

Tämän opinnäytetyön aikana tehtiin ohjelmisto sensoriverkon testaukseen. Opinnäytetyöhön kuului suunnittelu, toteutus sekä alustava testaus. Ikäväksemme jouduimme toteamaan käytetyistä tekniikoista WPF:än liian raskaaksi mittausten piirtämiseen eivätkä todennäköisesti muutkaan. NET tekniikat pääse riittävään nopeuteen. Käyttöliittymää lähdettiin hiomaan halutunlaiseksi ensin vain yhden mittauslaitteen datan näyttämiseen, jossa se osoittautui erittäin raskaaksi ja resursseja varaavaksi. Onneksemme kaikki työ ei kuitenkaan valunut hukkaan, vaan muuta osaa ohjelmasta voitiin hyödyntää.

Käyttöliittymä on toteutettu Labview:llä, johon Savonian tutkimus- ja kehitysyksiköstä löytyikin lisenssi ja osaaja. Projektin viimeistelyn laajempi testaus suoritetaan opinnäytetyön jälkeen, koska mittausyksiköiden ohjelmointi on vielä kesken eikä ohjelmaa pystytty testaamaan oikeassa testausympäristössä.

Opinnäytetyöni aikana pääsin perehtymään uusiin tekniikoihin, kehitystekniikoihin ja työkaluihin. Työhön perehtyessä tutustuin jo Lange-hankkeesta olemassa oleviin dokumentteihin, joita muut projektityötekijät olivat jo tehneet vuoden 2011 tammikuusta lähtien. Aloin jo hyvissä ajoin perehtymään WPF:n, jota tarvitsin jo ensimmäisen testausversion valmistumiseen. WPF:ssä en juuri huomannut muita eroavaisuuksia C#-ohjelmointikieleen verrattuna, paitsi että C#-ohjelmointikielessä oli toki joitakin poikkeuksia sekä käyttöliittymän ulkoasun suunnittelu tapahtui XAML-kielellä, joka vaati totuttelua. Käyttöliittymän suunniteluun olisi ollut tarjolla myös esimerkiksi Microsoft Expression Blend, joka tarjoaa monipuolisemmat työkalut suunnitteluun kuin Visual Studio. Myöhemmässä vaiheessa projektia tuli opeteltavaksi uusi tekniikka, WCF, johon perehtyminen vei oman aikansa. WCF-tekniikkaa käytettiin lähinnä rajapintojen luomiseen serverin, tietokannan ja käyttöliittymän välille, joten monet sen tarjoamat ominaisuudet jäivät vielä kokeilematta.

Sovelluksen kehittäminen oli mielenkiintoista ja palkitsevaa, johtuen uusista opittavista tekniikoista, kuten XAML, WCF, WPF sekä serverin ohjelmoinnista. Mielenkiintoista projektissa oli myös nähdä elektroniikkapuolen toteutus ja työn eteneminen. Työn alussa projektiryhmään kuului myös 5 vaihto-opiskelijaa Hollannista, Saksasta ja Belgiasta.

Opinnäytetyössä käyttöliittymän ohjelmointi oli kaikista aikaa vievin ja haastavin osakokonaisuus projektissa, koska tekniikka oli uusi ja eikä ollut tietoa kuinka käytännössä WPF ja allekirjoittanut tästä selviäisi. Yhteensä käyttöliittymästä ohjelmointiin kolme erillistä sovellusta, joista yksi osoittautui sopivaksi.

LÄHTEET

1. Sipilä, Matti & Toppinen, Arto. Langattomien sensoreiden käyttö lääketieteellisessä multiparametrimonitoroinnissa. Kuopio: Lange-hankkeen dokumentti, 2010.

2. Microsoft. Microsoftin msdn sivut WPF. [Verkkodokumentti] 2011.[viitattu: 16.9.2011]Saatavissa: http://msdn.microsoft.com/en-us/library/aa970268.aspx

3. Microsoft. Microsoft msdn sivut WCF.[Verkkodokumentti] 2011. [viitattu: 16.9.2011] Saatavissa: http://msdn.microsoft.com/library/ee958158.aspx

4. .Microsoft. msdn sivut XAML.[Verkkodokumentti] 2011. [viitattu: 16.9.2011] Saatavissa: http://msdn.microsoft.com/en-us/library/ms752059.aspx

5. Ruutinen, Jarno. Lange-dokumentit. Kuopio: Lange-hankkeen dokumentti, 2011.


SAVONIA