Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

Jaakko Tranberg

REAALIAIKAINEN LÄMPÖTILAMITTAUSTEN HALLINTA JA TALLENTAMINEN TIETOKANTAAN TCP/IP-PROTOKOLLAA HYVÄKSI KÄYTTÄEN.

Tietotekniikan koulutusohjelma
2017

Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

REAALIAIKAINEN LÄMPÖTILAMITTAUSTEN HALLINTA JA TALLENTAMINEN TIETOKANTAAN TCP/IP-PROTOKOLLAA HYVÄKSI KÄYTTÄEN.

Tranberg, Jaakko
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Toukokuu 2017
Ohjaaja: Auramo, Yrjö
Sivumäärä:
Liitteitä:

Asiasanat: Heraeus, C#, C-Sharp, Lämpötilamittaukset, SQL, Tietokannat, TCP/IP, Componenta, Windows.

Opinnäytetyön tavoitteena oli ohjelmoida ohjelmisto, jonka toimintoihin kuului sulan teräksen lämpötilatietojen keräys. Tiedot kerättiin Heraeus-lämpötila-antureita sekä Ethernet-verkkoa hyväksi käyttäen. Lisäksi ohjelmiston toimintoihin kuului tulosten tallennus SQL-tietokantaan myöhempää tarkastelua varten, sekä tiettyjen mittaustulosten edelleen lähetys toisille tietokoneille reaaliaikaisesti. Työkaluina toimivat Microsoft SQL Management Studio ja Visual Studio 2013, jossa käytetyksi ohjelmointikieleksi valittiin C#.

Ohjelmisto rakennettiin Windows-alustalle ja siihen kuului kaksi komponenttia. Niistä ensimmäinen on ohjelma, joka vastaanottaa Heraeus-antureiden lähettämiä lämpötilamittaukset sisältäviä TCP/IP-paketteja ja tallentaa ne SQL-tietokantaan sekä lähettää mittaustuloksia ohjelmiston toiselle komponentille. Se vastaanottaa tiettyjä pääohjelmasta lähetettyjä mittaustuloksia tarpeen mukaan reaaliaikaisesti ja näyttää ne tietokoneen näytöllä.

CONTROLLING AND RECORDING OF REALTIME TEMPERATURE MEASUREMENTS TO DATABASE USING TCP/IP PROTOCOL.

Tranberg, Jaakko
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
May 2017
Supervisor: Auramo, Yrjö
Number of pages:
Appendices:

Keywords: Heraeus, C#, C-Sharp, Lämpötilamittaukset, SQL, Tietokannat, TCP/IP, Componenta, Windows.

The purpose of this thesis was to create a software that collects temperature measurements of melted steel from Heraeus thermometer via Ethernet network. In addition, software was designed to record the results to SQL-database for later examining and to forward certain measurements to another computer and display the results on the screen in real time. Tools that were used was Microsoft SQL Management Studio and Visual Studio 2013, in which the programming language that was selected was C#.

Software was built for Windows platform and it included two components; main program that receives measurements from Heraeus thermometers via TCP/IP and records them in SQL-database and sends measurements for the other, secondary program on a different computer that receives measurements from the main program and displays them on the screen.

TERMIT JA LYHENTEET

Heraeus	Lämpömittareiden valmistaja.
SQL	Structured Query Language. Kyselykieli, jolla luodaan ja hallitaan relaatiotietokantoja.
C# / C-Sharp	Yleiskäyttöinen, oliopohjainen Microsoftin kehittämä ohjelmointikieli.
Visual Studio	Microsoftin kehittämä IDE.
TCP/IP	Transmission Control Protocol / Internet Protocol. Reititys- / tiedonsiirtoprotokolla.
Port	IP-osoitteeseen liitetty paketin pääte tai alkupiste.
IDE	Integrated Development Environment. Kokoelma ohjelmia, joilla voi kehittää, testata ja suunnitella ohjelmistoja.
ASCII	American Standard Code for Information Interchange. Standardi, joka määrittää merkkikoodauksen.
GUI	Grapical User Interface. Graafinen käyttöliittymä.

SISÄLLYS

1. JOHDANTO

Componenta on metallivaluja tarjoava Euroopassa toimiva yritys, sillä on toimistoja ja valimoja monessa maassa ja pääkonttori sijaitsee Helsingissä. Asiakkaita sillä on autoteollisuudesta aina maatalous- ja kaivosteollisuuteen asti. Työntekijöitä Componentalla on yli 5000 ja se on perustettu 1918 Helsingissä.

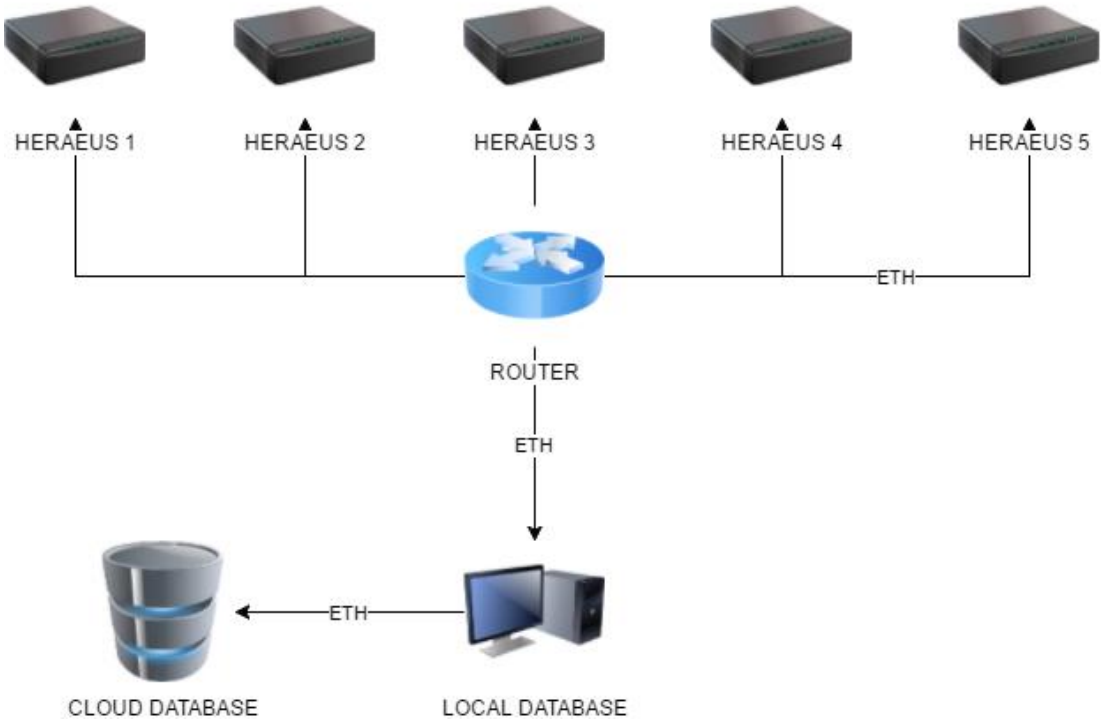
Componenta valmistaa metallisulasta valamalla mm. autoihin jarrulevyjä ja muita osia. Joskus osissa huomataan rakenteellisia materiaaliongelmia vasta valamisen jälkeen ja tällöin olisi hyvä tietää, että minkä lämpöistä metallisula on ollut minkäkin valmistuserän valamisen aikana. Tätä varten yritys halusi ohjelman, joka tallentaa lämpötilamittaukset kellonaikoineen tietokantaan myöhempää tarkastelua varten. Myöhemmin huomattiin myös, että olisi hyvä, jos magneettikouran ohjaamosta, josta käsin metalliromua syötetään sulatusuuniin, pystyttäisiin seuraamaan lämpötilamittauksia ja kellonaikaa, milloin viimeisin mittaus on tehty. Näin tiedettäisiin, milloin seuraavan metallisulan tarvitsisi olla valmiina valamista varten.

Yritysmaailmassa kustomoitujen ohjelmistojen tilaaminen ohjelmistotaloilta on usein ainoa keino hankkia ohjelmistoja, joissa on kaikki toiminnot, mitä yritys vaatii. Joskus suoraan yleisessä levityksessä oleva ohjelma täyttää kaikki tarpeet, mutta se on harvinaista, varsinkin sellaisissa tapauksissa, joissa tarpeet ovat vähänkään erikoisemmat. Tämän työn tapauksessa lämpötilamittareiden valmistajan, Heraeuksen, tarjoama ohjelmisto olisi tarjonnut vain osan toiminnoista ja olisi ollut kohtalaisen kallis, mutta koska Heraeus tarjoaa hyvät spec sheetit mittareilleen ja niissä on hyvät ja monipuoliset laite- ja verkko-ominaisuudet, niin oman ohjelmiston tekeminen on verrattain suoraviivainen asia. Tässä opinnäytteessä tarkastellaan ohjelmistorakennetta sekä käydään luomisprosessin eri vaiheita läpi.

2. JÄRJESTELMÄ

2.1 Verkko

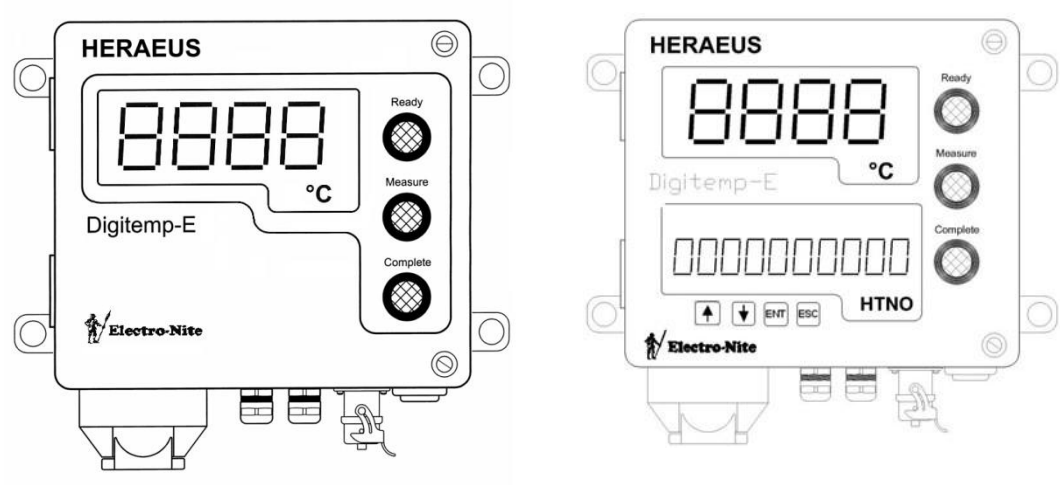
Yrityksellä on käytössään Ethernet-sisäverkko, jossa kaikki yrityksen data kulkee. Verkkoon on kytkettynä Heraeus-lämpötila-antureita, joilla metallisulan lämpötilaa tarkkaillaan. Verkkoon kytkettiin tietokone, joka on käytössä vain lämpötilatietojen tallennuksessa SQL-tietokantaan ja sillä on oma, kiinteä IP-osoite, kun normaalisti tietokoneet saavat osoitteensa DHCP-palvelimen kautta.



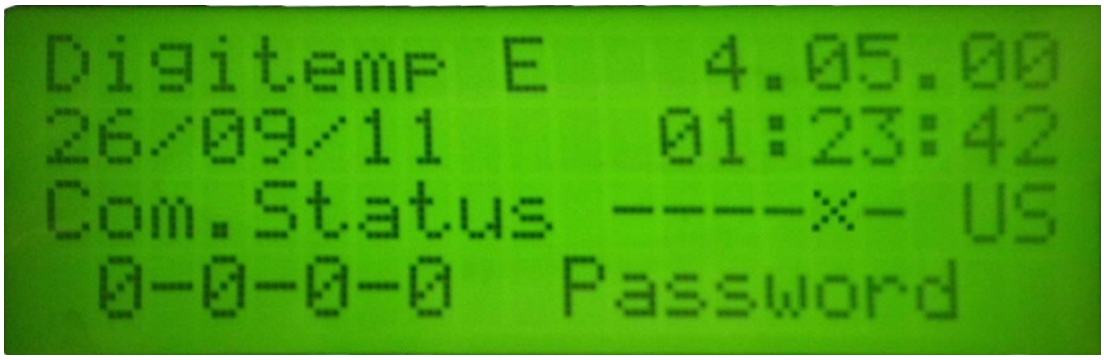
Kuva 1. Componentan lämpötila-anturien verkko

2.2 Heraeus Digitemp-E lämpötilamittarit

Kuva 2. Heraueus Digitemp-E lämpötilamittari (*Heraeuksen www-sivut 2017)*



Kuva 3. Digitemp-E lämpötilamittari kansi kiinni ja auki. (*Heraeus Electro-Nite 2011)*

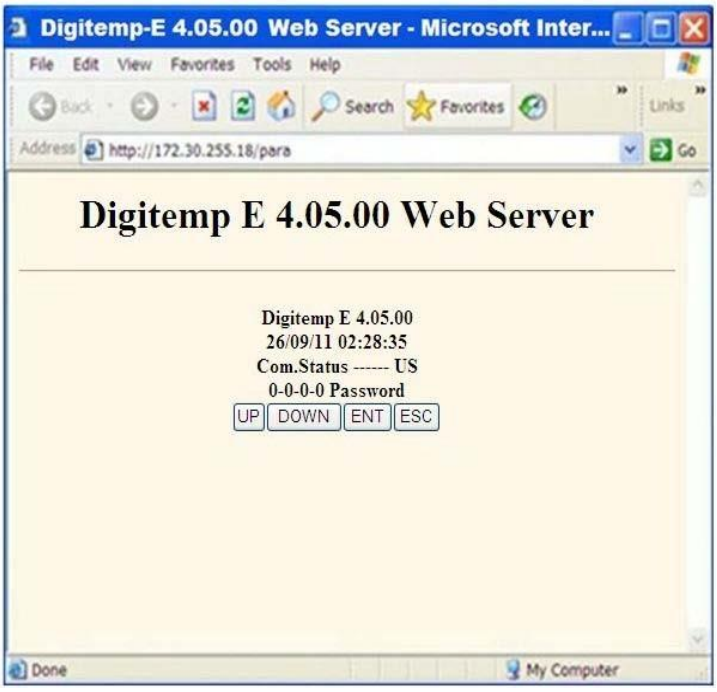


Kuva 4. Fyysisen käyttöliittymän näyttö. (*Heraeus Electro-Nite 2011)*

Kuva 5. Lämpötilan mittaukseen käytettävä kertakäyttöinen anturi. (*Heraeuksen www-sivut 2017)*

Yrityksellä on käytössään Heraeus Digitemp-E lämpötilamittarit. Jokaisella uudella mittauksella mittariin kiinnitetään kertakäyttöinen anturi, joka upotetaan sulaan metalliin ja lämpötilatieto luetaan mittarin näytöltä ja tämän jälkeen anturi hävitetään.

Lämpötilamittareissa on Ethernet-liitännät, joilla mittarin saa liitettyä lähiverkkoon, minkä jälkeen mittarin asetuksia pääsee säätämään selainpohjaisen käyttöliittymän kautta. Myös sarjaliitäntä ja langaton verkko ovat käytettävissä tarvittaessa, mutta tässä tapauksessa niitä ei käytetä. Asetusten säätäminen onnistuu myös avaamalla mittarin kansi, jonka alta paljastuu asetusten säädön mahdollistavat napit ja näyttö (kuva 4).



Kuva 6. Selainpohjainen käyttöliittymä.

Lämpötilamittaukset lähetetään mittareilta ASCII-muotoisina data-telegrammeina TCP/IP-protokollaa käyttäen. Nämä telegrammit voi joko määrittää itse tai käyttää valmiita pohjia.

Telegram 2: Set parameter 4.1 to 2 (short telegram with place)

```
- Date: 12.02.2010  Time: 11:29:38 -
Number ---------------------- HEX ----------------------   ----- ASCII ----
 0000  02 11 20 54  45 4D 50 20   31 36 30 33  20 43 20 20   .. TEMP 1603 C
 0010  50 4C 41 43  45 20 30 30   0D 0A 03                  PLACE 00...

#02#11 TEMP #R0-4000 #U  PLACE #P20#0D#0A#03
```

Kuva 7. Esimerkki data-telegrammista ja sen ohjelmoinnista. (*Heraeus Electro-Nite 2011)*

2.3 Tietokoneet

Tietokone, johon anturit lähettävät tietoja, on perus työpöytä-PC, joka on käytössä vain tiedonkeruussa. Alustana siinä toimii Windows 7 ja siinä pyörii tiedonkeruuohjelman lisäksi Microsoft SQL Server 2013. Kone vastaanottaa kaikki mittaustulokset sekä lähettää niitä eteenpäin isompaan tietokantaan ja sivuohjelmalle tarvittaessa.

3. SUUNNITTELU

3.1 Suunnittelun alkuvaiheet

Suunnittelu aloitettiin luonnostelemalla kynällä ja paperilla tarvittavia funktioita sekä käyttöliittymän mahdollinen ulkonäkö. Käyttöliittymän ulkonäöstä keskusteltiin yhteyshenkilön kanssa ja tultiin ratkaisuun, että pääohjelman pääikkunassa olisi kaaviossa käyrä, josta näkyy viimeisimmät mittaukset ja mittausten kellonajat sekä päivämäärät.

3.2 SQL Tietokanta

SQL-tietokannan suunnittelua varten oli valmiit kentät, jotka sen tulisi sisältää koska tiedot taulukoista siirrettäisiin sopivin väliajoin paikallisesta tietokannasta verkossa olevaan tietokantaan, josta Componentan työntekijät voivat tarvittaessa tehdä kyselyitä.

SQLQuery1.sql - DE...A5SBG\Jaakko (57))

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [Id]
      ,[TEMP]
      ,[ANTURI_NIMI]
      ,[ANTURI_IP]
      ,[ANTURI_PLACE]
      ,[TIME]
  FROM [Mittaukset].[dbo].[MITTAUKSET]
```

100 %

Results | Messages

	Id	TEMP	ANTURI_NIMI	ANTURI_IP	ANTURI_PLACE	TIME
1	308	1475	Junker-1	10.1.9.21	21	30-06-2015 15:48:11
2	309	1458	KP-uuni	10.1.9.20	20	30-06-2015 16:56:48
3	310	1491	Junker-1	10.1.9.21	21	01-07-2015 10:24:54
4	311	1493	Junker-1	10.1.9.21	21	01-07-2015 10:55:18
5	313	1518	Junker-1	10.1.9.21	21	01-07-2015 11:43:49
6	314	1521	Junker-1	10.1.9.21	21	01-07-2015 11:46:06
7	325	1518	Junker-1	10.1.9.21	21	01-07-2015 12:46:38
8	330	1502	Junker-1	10.1.9.21	21	01-07-2015 13:18:49
9	338	1464	Junker-1	10.1.9.21	21	01-07-2015 13:59:34
10	339	1464	Junker-1	10.1.9.21	21	01-07-2015 13:59:35
11	340	1464	Junker-1	10.1.9.21	21	01-07-2015 14:02:21
12	341	1464	Junker-1	10.1.9.21	21	01-07-2015 14:02:58

Kuva 8. Esimerkki ohjelmassa käytettävästä SQL-tietokannasta.

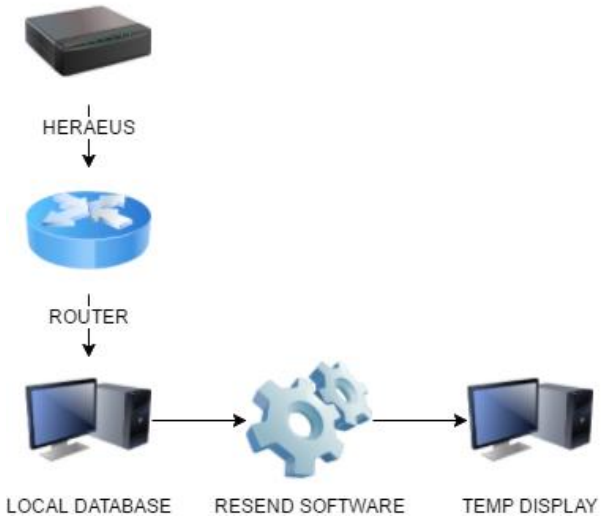
Tietokannassa on kuusi kenttää. Tiedoistaa neljä tulee ohjelmasta ja kaksi tulee SQL-palvelimelta. Ohjelmalta tulee TEMP, ANTURI_NIMI, ANTURI_IP ja ANTURI_PLACE. Mittausajankohdan määrittää tiedon tallennusaika, joka tulee SQL-palvelimelta ja mittauksen ID:n määrittää tietokanta.

3.3 Pääohjelman käyttöliittymä

Pääohjelman käyttöliittymän prototyyppejä toteutettiin muutamia päiviä, kunnes päädyttiin lopulliseen käyttöliittymän ulkonäköön. Suunniteluun käytettiin Visual Studion graafisia toimintoja tekemällä pelkkä ohjelman visuaalinen ulkoasu ilman toimivaa ohjelmakoodia ajan säästämiseksi.

3.4 Sivuohjelma

Tarvetta sivuohjelmalle ei aluksi edes huomattu olevan. Koska ohjelman käyttötarkoitus on kohtalaisen yksinkertainen, niin suunnitteluun ei tarvinnut käyttää paljoa aikaa. Se vähä mitä suunnitteluun käytettiin, tehtiin samalla tavalla kuin pääohjelmankin suunnittelu, eli graafinen ulkoasu ilman ohjelmakoodia – esittely ja hyväksyttäminen yhteyshenkilöllä ja sen jälkeen toteutus.



Kuva 9. TCP/IP-paketin kulku lämpötila-anturilta tietokantaan ja sieltä toisen koneen ruudulle lämpötilatiedon edelleen lähetyksessä.

4. KÄYTETTÄVÄT OHJELMISTOT JA ALUSTA

4.1 Visual Studio 2013

Visual Studio 2013 on Microsoftin kehittämä ja ylläpitämä IDE joka sisältää työkalut niin graafiseen suunnitteluun kuin konsolipohjaisten ohjelmien toteuttamiseen. Visual Studiolla on mahdollista toteuttaa ohjelmistoja monella eri ohjelmointikielellä. Esimerkiksi C#, C++, C, Visual Basic, .NET, F#, Python, Ruby ja monia muita. Tässä työssä käytettäväksi valittiin C# sen monipuolisuuden ja helppokäyttöisyyden vuoksi.

4.2 Microsoft SQL Management Studio

SQL Management Studio on käytännöllinen silloin, kun halutaan tarkastella koko tallennettua paikallista tietokantaa helposti samalta koneelta, johon tiedonkeruu tapahtuu. Sillä pystyy helposti myös luomaan tietokantapohjan, johon tiedot voi tallettaa.

4.3 Windows

Alustana Windows oli luonnollinen valinta, koska kaikki muutkin yrityksen koneet käyttivät Windowsia.

5. KÄYTETTÄVÄT TEKNIIKAT

5.1 TCP/IP

TCP- ja IP-protokollat huolehtivat IP-yhteyksien kuten esimerkiksi socket-yhteyden luomisesta ja reitityksestä. Socket-osoite koostuu IP-osoitteesta ja portista.

```
32              TcpClient client = new TcpClient();
33              client.Connect("127.0.0.1", 8520);
```

Kuva 10. Esimerkki socket-yhteyden luomisesta.

Esimerkissä luodaan C#-ympäristössä TCP/IP-yhteys IP-osoitteeseen 127.0.0.1 (Localhost) portissa 8520.

5.2 Säikeistys (Threading)

Säikeistys on ohjelmoinnissa käytettävä tekniikka, jonka avulla voidaan ajaa montaa eri ohjelmakoodia samaan aikaan. Silloin, kun prosessorit olivat vielä yksiytimisiä, tätä tekniikkaa käytettiin simuloimaan moniajoa ajamalla eri ohjelmakoodeja peräkkäin vuorottelemalla aina muutaman nanosekunnin kerrallaan, jolloin käyttäjälle tuli illuusio siitä, että kaikki tapahtuu yhtäaikaisesti. Vasta useammat prosessoriytimet mahdollistivat todellisen moniajon.

Tämän opinnäytteen perspektiivistä säikeistystä tarvitaan siksi, että ilman sitä, graafinen käyttöliittymä ei toimisi, koska prosessori olisi jumissa ajamassaan ohjelmakoodissa sen suoritusajan loppuun aina kun käyttäjä haluaisi esimerkiksi avata asetukset-ikkunan tai ohjelma vastaanottaisi mittaustuloksen.

C#-kielessä säikeistyksen voi toteuttaa monella eri tavalla. Tiedonkeruuohjelmassa se on toteutettu graafisen käyttöliittymän tarjoamalla BackGroundWorker-luokalla, paitsi tiedon vastaanotossa. Siellä se hoidettiin perus Thread-luokalla.

Tiedon vastaanotossa ei ollut syytä käyttää raskaampaa BackGroundWorker luokkaa, koska se on tarkoitettu lähinnä graafisen käyttöliittymän säikeistykseen ja sen toimintoja ei tiedon vastaanotossa olisi tarvittu.

```
700     private void bgwChartUpdate_DoWork(object sender, DoWorkEventArgs e)
701     {
702         XmlDocument docs = new XmlDocument();
703         docs.Load("asetukset.xml");
704
705         XmlNode dbFile = docs.SelectSingleNode("//Asetukset/Tietokannan_Sijainti/TK_Polku");
706         XmlNode dbSource = docs.SelectSingleNode("//Asetukset/Tietokannan_Sijainti/TK_Source");
707         Kalut kalu = new Kalut();
708         while (true)
709         {
710           String paikka = "";
711           try
712           {
713               this.Invoke(new MethodInvoker(delegate()
714               {
715               paikka = kalu.haePlace(Anturivalinnat.SelectedItem.ToString());
716               }));
717           }
718           catch (Exception ex)
719           {
720               Kalut kalu_ex = new Kalut();
721               kalu_ex.exToFile(ex.ToString());
722           }
723
724           try
725           {
726               SqlConnection myConnection = new SqlConnection(@"Data Source=" + dbSource.InnerText.ToString() +
727                                                              ";AttachDbFilename=" + dbFile.InnerText.ToString() +
728                                                              ";Integrated Security=True;Connect Timeout=10");
729               myConnection.Open();
730
731               SqlCommand sqlCommand = new SqlCommand("SELECT TOP 8 * FROM MITTAUKSET WHERE ANTURI_PLACE='" +
732                                                      paikka + "' ORDER BY TIME DESC", myConnection);
733               SqlDataReader sqlReader = sqlCommand.ExecuteReader();
734               bgwChartUpdate.ReportProgress(1, sqlReader);
735           }
736           catch (Exception ex)
737           {
738               Kalut kalu_ex = new Kalut();
739               kalu_ex.exToFile(ex.ToString());
740           }
741
742           Thread.Sleep(1000);
743
744       }
745   }
```

Kuva 11. Esimerkki säikeistyksestä BackGroundWorker-luokan avulla.

Kuvan esimerkin säikeellä päivitetään tiedonkeruuohjelman lämpötilakäyrä, minkä jälkeen säie nukkuu sekunnin ja päivittää käyrän uudelleen. Ensin säie avaa asetukset.xml-tiedoston, josta se hakee tietokannan sijainnin (dbFile) ja SQL-palvelimen instanssin nimen (dbSource). Tämän jälkeen tulee ikuinen While-silmukka, joka ottaa SQL-palvelimeen yhteyden ja hakee tietokannasta 8 viimeisintä tulosta valitulta anturilta, lajittelee tulokset laskevaan järjestykseen mittausajan

mukaan ja lähettää tuloksen Chartille ReportProgress()-metodia hyväksi käyttäen, joka tulostaa käyrän näytölle.

```
194             bgwChartUpdate.RunWorkerAsync();
```

Kuva 12. Säikeen käynnistus ohjelmakoodissa.

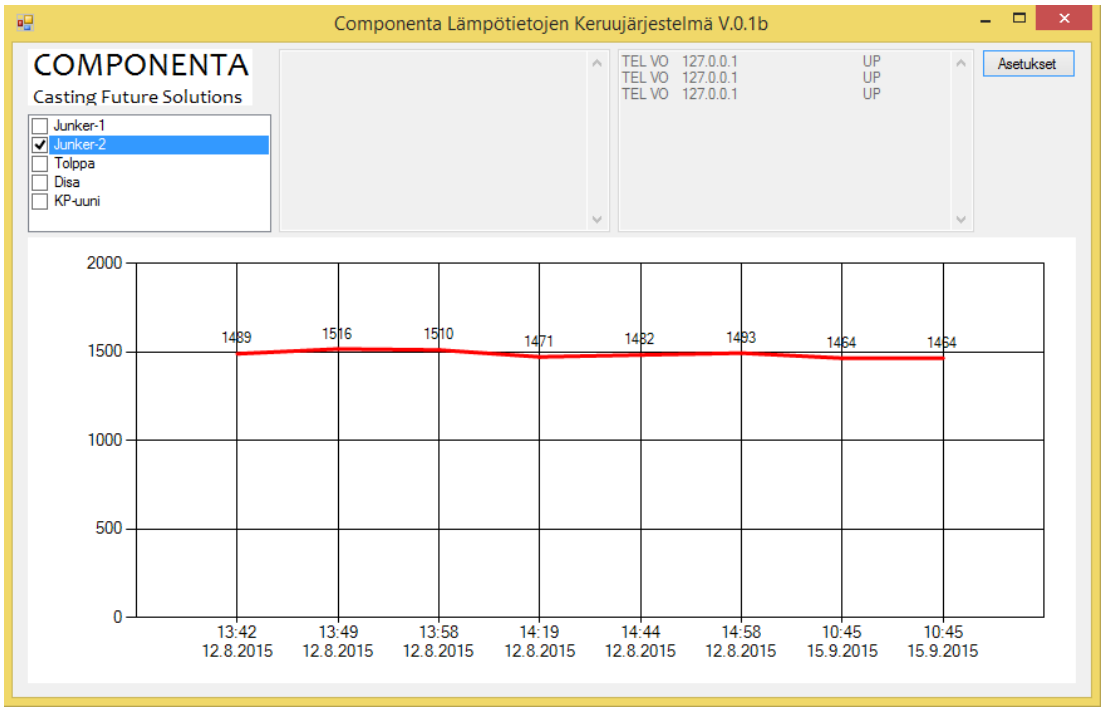
```
23              Thread ctThread = new Thread(doChat);
24              ctThread.Start();
25          }
26          private void doChat()
27          {
28            //Koodia...
```

Kuva 13. Esimerkki säikeistyksestä Thread-luokan avulla.

6. PÄÄOHJELMA

6.1 Toimintaperiaate

Ohjelman toimintaperiaate on verrattain yksinkertainen. Kun ohjelma on käynnissä koneella, se odottaa lämpötila-antureiden lähettämiä paketteja tietyissä, ennakkoon määritellyissä porteissa ja tietyiltä IP:ltä. Ohjelma vastaanottaa, käsittelee ja tallentaa oleelliset tiedot tietokantaan ja näyttää viimeisimmät mittaustulokset ruudulla. Sen lisäksi valittaessa lähettää ne jatkokäsittelyyn sivuohjelmalle, joka on käynnissä toisella tietokoneella.



Kuva 14. Pääohjelma

Vasemmasta ylälaidasta pystyy valitsemaan anturin, jonka mittauksia halutaan tarkastella. Keskimmäisessä yläikkunassa näkyy, mitkä anturit ovat ylhäällä ja mitkä alhaalla. Lisäksi oikeanpuoleisessa ikkunassa näkyy tiedon edelleenlähetyksen vastaanottajien tila. Oikean yläkulman nappulasta pääsee asetukset-valikkoon.

Anturilta lämpötilan mittauksen tulos tulee ASCII:na muodossa:

◄ TEMP 1464 C PLACE 21

♥♥

Paketissa ’ ◄’ on ilmoitus uudesta paketista jonka jälkeen seuraa ’TEMP XXXX C’ lämpötila celsius asteina ja sen jälkeen ’PLACE 21’ anturin paikka. Paikka määräytyy anturin IP-osoitteen viimeisestä kolmesta numerosta. Maksimimäärä antureille yhdessä aliverkossa on näin ollen 255 kappaletta. Viimeisenä tulee ’♥♥’, joka merkitsee paketin loppua.

Paketin sisältö on määritettävissä eri muotoihin antureiden käyttöliittymästä käsin.

```
object[] objMittaus = new object[2];
/// <summary>
/// Metodi lämpötilatietojen formatointiin tietokantaan sopivaksi
/// </summary>
/// <param name="mittaus"></param>
/// <returns>Palauttaa obj arrayn, jossa [0] on lämpötila ja [1] on anturin
/// sijainti (kaksi viimeisintä ip-numeroa)</returns>
public object[] lampoKantaan(string mittaus)
{
    //int i = 0;
    StringBuilder temp = new StringBuilder();
    StringBuilder place = new StringBuilder();
    StringBuilder tmp = new StringBuilder();
    foreach (char c in mittaus)
    {
        tmp.Append(c);
            if (tmp.Length >= 6)
            {
                if (tmp.ToString().Substring((tmp.Length - 5), 4) == "TEMP")
                {
                    temp.Append(mittaus.Substring(tmp.Length, 4));
                }
            }
            if (tmp.Length >=8)
            {
                if (tmp.ToString().Substring((tmp.Length - 6), 5) == "PLACE")
                {
                    place.Append(mittaus.Substring(tmp.Length, 2));
                }
            }
    }
    objMittaus[0] = temp.ToString();
    objMittaus[1] = place.ToString();
    tmp.Clear();
    temp.Clear();
    place.Clear();

    if (objMittaus[0].ToString().Length > 0 && objMittaus[1].ToString().Length > 0)
    {
    return objMittaus;
    }
    else
    {
        objMittaus[0] = "0000";
        objMittaus[1] = "00";
        return objMittaus;
    }
}
```

Kuva 15. Anturin lähettämän paketin formatointi tietokantaan sopivaksi.

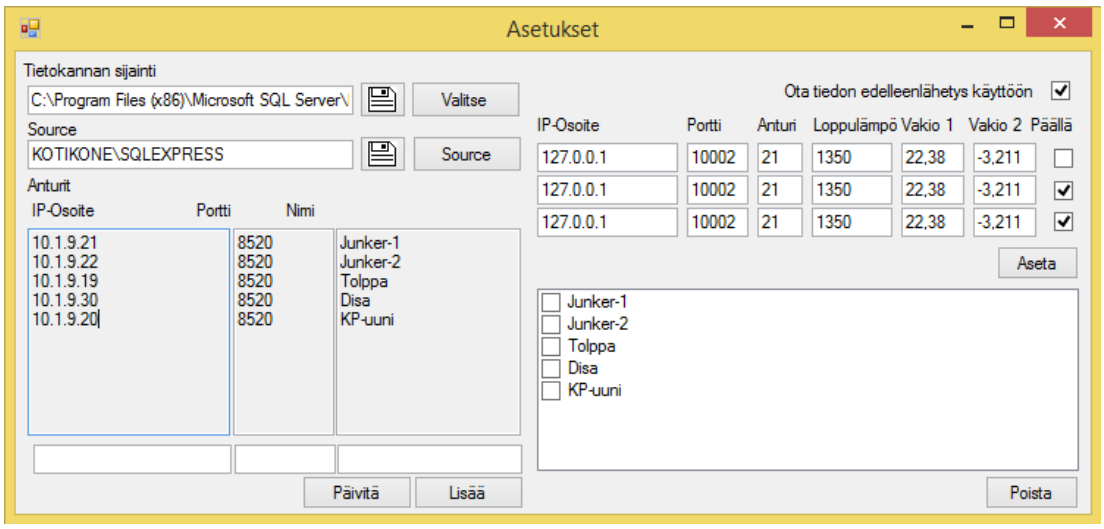
6.2 Asetukset -ikkuna

Pääohjelman asetuksista pääsee muokkaamaan kaikkia tarvittavia asetuksia. Nämä asetukset voi muuttaa myös muokkaamalla ’asetukset.xml’-tiedostoa, mutta GUI toteutettiin toiminnan helpottamiseksi.

```
<?xml version="1.0" encoding="utf-8"?>
<Asetukset>
  <Toiminta>
    <Server Mode="Off"></Server>
    <Client Mode="On"></Client>
  </Toiminta>
  <Tietokannan_Sijainti>
    <TK_Polku>C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\DATA\Mittaukset.mdf</TK_Polku>
    <TK_Source>DESKTOP-1KA5SBG\SQLEXPRESS</TK_Source>
  </Tietokannan_Sijainti>

  <Tiedon_Edelleenlahetys Mode="On">
    <TEL_IP_Status>ON</TEL_IP_Status>
    <TEL_IP>127.0.0.1</TEL_IP>
    <TEL_Portti>10002</TEL_Portti>
    <TEL_Anturista>21</TEL_Anturista>
    <TEL_Loppulampotila>1350</TEL_Loppulampotila>
    <TEL_Vakio1>22,38</TEL_Vakio1>
    <TEL_Vakio2>-3,211</TEL_Vakio2>

    <TEL_IP_Status_2>ON</TEL_IP_Status_2>
    <TEL_IP_2>127.0.0.1</TEL_IP_2>
    <TEL_Portti_2>10002</TEL_Portti_2>
    <TEL_Anturista_2>21</TEL_Anturista_2>
    <TEL_Loppulampotila_2>1350</TEL_Loppulampotila_2>
    <TEL_Vakio1_2>22,38</TEL_Vakio1_2>
    <TEL_Vakio2_2>-3,211</TEL_Vakio2_2>

    <TEL_IP_Status_3>ON</TEL_IP_Status_3>
    <TEL_IP_3>127.0.0.1</TEL_IP_3>
    <TEL_Portti_3>10002</TEL_Portti_3>
    <TEL_Anturista_3>21</TEL_Anturista_3>
    <TEL_Loppulampotila_3>1350</TEL_Loppulampotila_3>
    <TEL_Vakio1_3>22,38</TEL_Vakio1_3>
    <TEL_Vakio2_3>-3,211</TEL_Vakio2_3>

  </Tiedon_Edelleenlahetys>

  <Anturit>
    <Anturi Nimi="Junker-1">
      <A_IP>10.1.9.21</A_IP>
      <A_Portti>8520</A_Portti>
    </Anturi>
    <Anturi Nimi="Junker-2">
      <A_IP>10.1.9.22</A_IP>
      <A_Portti>8520</A_Portti>
    </Anturi>
    <Anturi Nimi="Tolppa">
      <A_IP>10.1.9.19</A_IP>
      <A_Portti>8520</A_Portti>
    </Anturi>
    <Anturi Nimi="Disa">
      <A_IP>10.1.9.30</A_IP>
      <A_Portti>8520</A_Portti>
    </Anturi>
    <Anturi Nimi="KP-uuni">
      <A_IP>10.1.9.20</A_IP>
      <A_Portti>8520</A_Portti>
    </Anturi>
  </Anturit>

</Asetukset>
```

Kuva 16. Asetukset xml-tiedostossa.

Vasemmalla ylhäällä näkyy tietokannan sijainti ja SQL-palvelimen instanssin nimi. Vasemmalla alhaalla näkyy kaikki käytössä olevat anturit, niiden IP-osoitteet, portit ja nimet. Oikealla ylhäällä on tiedon edelleenlähetyksen asetukset. Ensimmäisenä on toiminnon käyttöönotto ja poisto. Alempana on IP-osoitteet ja portti, joihin tieto edelleenlähetetään. Edelleenlähetyksen kohteita voi olla kerrallaan käytössä kolme. Alimpana oikealla on tarpeettomien antureiden poisto.



Kuva 17. Pääohjelman asetukset-ikkuna.

6.3 Toiminta tarkemmin

Tiedonkeruuohjelman käynnistyessä ohjelma käynnistää kolme BackGroudWorker-säiettä. Yksi lämpötilamittareiden ja tiedon edelleenlähetyksen vastaanottajien tilan seurantaa, yksi lämpötilakäyrän päivitystä varten ja viimeinen lämpötilamittausten vastaanottamista varten.

```
44          bgwAnturiPing.RunWorkerAsync();
45          backgroundWorker1.RunWorkerAsync();
46          bgwChartUpdate.RunWorkerAsync();
```

Kuva 18. Säikeiden käynnistys.

Lämpötilamittareiden ja tiedon edelleenlähetyksen vastaanottajien tilaa seurataan pingaamalla niitä ja katsomalla vastaako ne. Ohjelma hakee asetukset.xml-tiedostosta mittareiden IP-osoitteet ja portit ja lähettää niille Ping-komennolla paketin ja odottaa vastausta. Jos vastaus saapuu, niin mittari on ylhäällä ja sen statukseksi kirjoitetaan UP pääikkunaan. Muussa tapauksessa statukseksi kirjoitetaan DOWN. Tämä tapahtuu kolmen sekunnin välein.

```
391  foreach (var item in anturit_ip)
392  {
393      try
394      {
395          Ping pingSender = new Ping();
396          PingOptions options = new PingOptions();
397
398          options.DontFragment = true;
399
400          // Create a buffer of 32 bytes of data to be transmitted.
401          string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
402          byte[] pbuffer = Encoding.ASCII.GetBytes(data);
403          int timeout = 10;
404
405          PingReply reply = pingSender.Send(IPAddress.Parse(item), timeout, pbuffer, options);
406
407          if (reply.Status == IPStatus.Success)
408          {
409              if (item.Length < 10)
410              {
411                  sb_Anturit.AppendLine("Anturi\t" + item.ToString() + "\t" + testi.haeNimi(item.ToString()) + "\t\tUP\n\n");
412              }
413              else
414              {
415                  sb_Anturit.AppendLine("Anturi\t" + item.ToString() + "\t" + testi.haeNimi(item.ToString()) + "\tUP\n\n");
416
417              }
418          }
419          if (reply.Status == IPStatus.BadDestination ||
420              reply.Status == IPStatus.BadRoute ||
421              reply.Status == IPStatus.DestinationHostUnreachable ||
422              reply.Status == IPStatus.DestinationNetworkUnreachable ||
423              reply.Status == IPStatus.DestinationProhibited ||
424              reply.Status == IPStatus.DestinationUnreachable ||
425              reply.Status == IPStatus.TimedOut
426              )
427          {
428              if (item.Length < 10)
429              {
430                  sb_Anturit.AppendLine("Anturi\t" + item.ToString() + "\t" + testi.haeNimi(item.ToString()) + "\t\tDOWN\n\n");
431              }
432              else
433              {
434                  sb_Anturit.AppendLine("Anturi\t" + item.ToString() + "\t" + testi.haeNimi(item.ToString()) + "\tDOWN\n\n");
435              }
436          }
437      }
```

Kuva 19. Antureiden tilan testaaminen Ping-komennolla.

Lämpötilakäyrän päivityksestä vastaava säie tekee SQL-tietokantaan kyselyn kolmen sekunnin välein kahdeksasta viimeisimmästä mittaustuloksesta sen anturin kohdalla, joka on valittuna pääikkunan valikosta. Tämän jälkeen säie päivittää kyselytuloksen Chartille ja päivittää lämpötilakäyrän. (kuva 11).

Mittaustulosten vastaanottamisesta vastaava säie käynnistää TCP-kuuntelijan (TcpListener), joka kuuntelee asetetussa portissa tulevia yhteyksiä. Yhteyden havaitessaan säie kutsuu handleClinet()-funktiota, joka ottaa luodun yhteyden haltuunsa ja luo sille oman säikeensä. Mittaustulosten vastaanottamisesta vastaava säie pysyy käynnissä kuunnellen uusia yhteyksiä.

```
491     private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
492     {
493         TcpListener serverSocket = new TcpListener(IPAddress.Any, 8520);
494         TcpClient clientSocket = default(TcpClient);
495         int counter = 0;
496 
497         serverSocket.Start();
498 
499         counter = 0;
500         while (true)
501         {
502             counter += 1;
503             clientSocket = serverSocket.AcceptTcpClient();
504             handleClinet client = new handleClinet();
505 
506             client.startClient(clientSocket, Convert.ToString(counter));
507         }
508     }
```

Kuva 20. Mittaustulosten vastaanottamisesta vastaava säie.

Ensimmäiseksi handleClinet()-funktio luo säikeen yhteyttä varten ja hakee asetukset.xml-tiedostosta tarvittavat tiedot tiedon edelleenlähetystä ja tietokantaan tallentamista varten .

```
16      public class handleClinet
17      {
18          TcpClient clientSocket;
19          string clNo;
20          public void startClient(TcpClient inClientSocket, string clineNo)
21          {
22              this.clientSocket = inClientSocket;
23              this.clNo = clineNo;
24              Thread ctThread = new Thread(doChat);
25              ctThread.Start();
26          }
27          private void doChat()
28          {
29              XmlDocument doc = new XmlDocument();
30              doc.Load("asetukset.xml");
31
32              XmlNode onOff = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys");
33
34              XmlNode paallan = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_IP_Status");
35              XmlNode telip = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_IP");
36              XmlNode telport = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Portti");
37              XmlNode telplace = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Anturista");
38              XmlNode loppulampo = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Loppulampotila");
39              XmlNode vakio1 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Vakio1");
40              XmlNode vakio2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Vakio2");
41
42              XmlNode paallan_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_IP_Status_2");
43              XmlNode telip_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_IP_2");
44              XmlNode telport_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Portti_2");
45              XmlNode telplace_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Anturista_2");
46              XmlNode loppulampo_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Loppulampotila_2");
47              XmlNode vakio1_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Vakio1_2");
48              XmlNode vakio2_2 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Vakio2_2");
49
50              XmlNode paallan_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_IP_Status_3");
51              XmlNode telip_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_IP_3");
52              XmlNode telport_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Portti_3");
53              XmlNode telplace_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Anturista_3");
54              XmlNode loppulampo_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Loppulampotila_3");
55              XmlNode vakio1_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Vakio1_3");
56              XmlNode vakio2_3 = doc.SelectSingleNode("//Asetukset/Tiedon_Edelleenlahetys/TEL_Vakio2_3");
57
58
59
60              XmlNode dbFile = doc.SelectSingleNode("//Asetukset/Tietokannan_Sijainti/TK_Polku");
61              XmlNode dbSource = doc.SelectSingleNode("//Asetukset/Tietokannan_Sijainti/TK_Source");
```

Kuva 21. HandleClinet-funktio luo säikeen ja hakee tietoja.

Tämän jälkeen funktio luo kaksi object-taulukkoa, joita tarvitaan mittaustulosten käsittelyä varten.

```
82              object[] tempobj = new object[2];
83              object[] nimiobj = new object[2];
```

Kuva 22. Tietojen käsittelyä varten luodut object-taulukot.

```
91  SqlConnection myConnection = new SqlConnection("Data Source=" + dbSource.InnerText.ToString() +
92                                                 ";AttachDbFilename=" + dbFile.InnerText.ToString() +
93                                                 ";Integrated Security=True;Connect Timeout=10");
94
95  if (myConnection.State != System.Data.ConnectionState.Open) {
96      myConnection.Open();
97  }
98
99  requestCount = requestCount + 1;
100 NetworkStream networkStream = clientSocket.GetStream();
101 networkStream.Read(bytesFrom, 0, bytesFrom.Length);
102 dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
103 tempobj = kalu.lampoKantaan(dataFromClient);
104 nimiobj = kalu.haeNimiJaIP(tempobj[1].ToString());
105
106 int Temp = Convert.ToInt32(tempobj[0].ToString());
107 string Anturi_Nimi = nimiobj[1].ToString();
108 string Anturi_IP = nimiobj[0].ToString();
109 int Anturi_Place = Convert.ToInt32(tempobj[1]);
```

Kuva 23. Tietokantayhteyden avaus ja tulosten käsittely.

Tässä vaiheessa avataan tietokantayhteys ja asetetaan jo luoduille taulukko-muuttujille arvot alustamalla ne funktiokutsuilla lampoKantaan() ja haeNimiJaIP(). LampoKantaan() ottaa vastaan object-taulukon ja palauttaa string-taulukon, joka sisältää mitatun lämpötilan sekä tietokantaan tulevan PLACE-arvon. haeNimiJaIP() ottaa vastaan object-taulukon ja palauttaa string-taulukon, joka sisältää anturin nimen ja IP:n.

Tämän jälkeen ohjelmalla on tarvittavat tiedot, jotta mittaustulokset voidaan tallentaa tietokantaan. Tarvittavat tiedot ovat Anturin nimi ja IP, jolta mittaus tulee, mittauksen lämpötila ja PLACE-arvo.

```
111 SqlCommand myCommand = new SqlCommand("INSERT INTO MITTAUKSET VALUES(@TEMP, @ANTURI_NAME, @ANTURI_IP, @ANTURI_PLACE, @TIME)",
112                                       myConnection);
113
114 myCommand.Parameters.AddWithValue("@TEMP", Temp);
115 myCommand.Parameters.AddWithValue("@ANTURI_NAME", Anturi_Nimi);
116 myCommand.Parameters.AddWithValue("@ANTURI_IP", Anturi_IP);
117 myCommand.Parameters.AddWithValue("@ANTURI_PLACE", Anturi_Place);
118 myCommand.Parameters.AddWithValue("@TIME", DateTime.Now); //.ToString("dd-MM-yyyy HH:mm:ss"));
119
120 myCommand.ExecuteNonQuery();
```

Kuva 24. Tietojen tallennus tietokantaan.

Tässä kohtaa tarkistetaan, onko tiedon edelleenlähetys päällä. Jos on, niin kutsutaan TEL-funktiota, joka hoitaa tiedon eteenpäin saattamisen.

```
127     if (onOff.Attributes["Mode"].Value == "On")
128     {
129
130         if (telplace.InnerText.ToString().EndsWith(Anturi_Place.ToString()))
131         {
132             if (paallan.InnerText == "On")
133             {
134                 kalu.TEL(Temp.ToString(), Anturi_Place.ToString(), telip.InnerText.ToString(),
135                     telport.InnerText.ToString(), loppulampo.InnerText.ToString(),
136                     vakio1.InnerText.ToString(), vakio2.InnerText.ToString());
137             }
138         }
139         if (telplace_2.InnerText.ToString().EndsWith(Anturi_Place.ToString()))
140         {
141             if (paallan_2.InnerText == "On")
142             {
143                 kalu.TEL(Temp.ToString(), Anturi_Place.ToString(), telip_2.InnerText.ToString(),
144                     telport_2.InnerText.ToString(), loppulampo_2.InnerText.ToString(),
145                     vakio1_2.InnerText.ToString(), vakio2_2.InnerText.ToString());
146             }
147         }
148         if (telplace_3.InnerText.ToString().EndsWith(Anturi_Place.ToString()))
149         {
150             if (paallan_3.InnerText == "On")
151             {
152                 kalu.TEL(Temp.ToString(), Anturi_Place.ToString(), telip_3.InnerText.ToString(),
153                     telport_3.InnerText.ToString(), loppulampo_3.InnerText.ToString(),
154                     vakio1_3.InnerText.ToString(), vakio2_3.InnerText.ToString());
155             }
156         }
157     }
```

Kuva 25. Tiedon edelleenlähetyksen tilan tarkistus.

TEL-funktio vastaanottaa lämpötilatiedon, IP:n ja portin, jolle tieto lähetetään. Lisäksi, funktio vastaanottaa loppulampo, vakio1 ja vakio2 -muuttujat, mutta niillä ei tällä hetkellä tehdä mitään ja ovat siellä vain sen takia, jos niitä tulevaisuudessa tarvitaan.

```
27          public void TEL (string lampo, string place, string telIP, string telPort, string loppulampo, string vakio1, string vakio2)
28          {
29              double TelVakio1 = Convert.ToDouble(vakio1);
30              double TelVakio2 = Convert.ToDouble(vakio2);
31              int TelLoppulampo = Convert.ToInt32(loppulampo);
32              int Lampo = Convert.ToInt32(lampo);
33
34              string laskuri = Convert.ToString((TelLoppulampo - Lampo + TelVakio1) / TelVakio2);
35
36              TcpClient client = new TcpClient();
37
38              this._telPort = Convert.ToInt32(telPort);
39              IPAddress _telIP = IPAddress.Parse(telIP);
40
41              try
42              {
43
44                  client.Connect(_telIP, _telPort);
45                  NetworkStream serverStream = client.GetStream();
46                  serverStream.Flush();
47                  byte[] outStream = System.Text.Encoding.ASCII.GetBytes(place + lampo + laskuri);
48                  serverStream.Write(outStream, 0, outStream.Length);
49                  serverStream.Flush();
50
51                  client.Close();
52              }
```

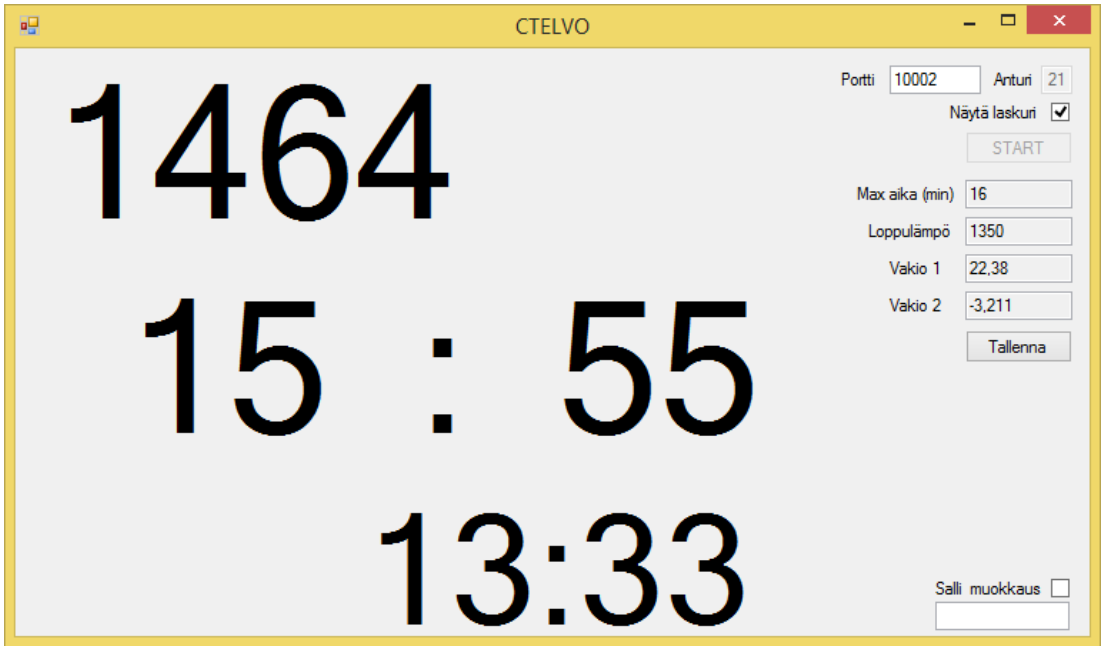
Kuva 26. TEL-funktio.

Funktio käyttää saamiaan IP ja porttitietoja ja ottaa socket-yhteyden vastaanottavaan koneeseen, jolle tämä lähettää NetworkStream-funktiota hyväksi käyttäen PLACE-arvon, lämpötilan ja laskuri-muuttujan.

7. SIVUOHJELMA

7.1 Toimintaperiaate

Kun ohjelma on käynnissä, se odottaa paketteja pääohjelmalta ennalta määritellyssä IP-osoitteessa ja portissa. Paketin saatuaan ohjelma näyttää ruudulla saamansa lämpötilan (1464 kuvassa 27), kellonajan (15:55 kuvassa 27) koska lämpötilatieto on saapunut ja laskee tietyn kaavan avulla ajan, kuinka kauan sulaa rautaa pystytään vielä käyttämään ennen kuin se on jäähtynyt liikaa. Liian kylmäksi jäähtynyt rautasula täytyy viedä sulatusuunille uudestaan. Ohjelma näyttää alaspäin laskevan laskurin, joka näyttää jäljellä olevan sulan raudan käyttöajan (13:33 kuvassa 27).



Kuva 27. Sivuohjelma.

Oikealla yläkulmassa näkyy kuunneltava portti ja anturi, millä mittaus on suoritettu. Sen alapuolella on asetukset, jotka mittaustulosten käyttäjä säätää. Lisäksi alimmaisena on salasanalla suojattu valinta, joka täytyy olla asetettuna, jotta asetuksia pystyy muokkaamaan. Tämä tehtiin, jotta työntekijä ei muuttaisi asetuksia vahingossa tai epähuomiossa.

$$\text{Loppulämpö - Mitattu lämpötila} + \frac{\text{Vakio 1}}{\text{Vakio 2}} = \text{Aika sekunteina.}$$

Kuva 28. Metallisulan käyttöajan laskukaava.

7.2 Toiminta tarkemmin

Ohjelma aloittaa lukemalla portti.txt-tiedostosta tarvittavat asetukset ja asettamalla ne ohjelmaikkunan tarvittaviin kohtiin.

```
25          using (StreamReader file = new StreamReader("portti.txt"))
26          {
27              while (true)
28              {
29                  try
30                  {
31                      txtPort.Text = file.ReadLine();
32                      txtLoppuLampo.Text = file.ReadLine();
33                      txtVakio1.Text = file.ReadLine();
34                      txtVakio2.Text = file.ReadLine();
35                      txtMaxAika.Text = file.ReadLine();
36                      file.Close();
37                  }
38                  catch (Exception)
39                  {
40                      return;
41                  }
42              }
43          }
```

Kuva 29. Ohjelma hakee tarvittavat asetukset.

Ohjelmassa on START-painike, jota painamalla ohjelma käynnistää pääsäikeen, joka hoitaa loput.

```
229         private void btnStart_Click(object sender, EventArgs e)
230         {
231             bgwMain.RunWorkerAsync();
232             btnStart.Enabled = false;
233             lblTemperature.Text = "0000";
234         }
```

Kuva 30. Pääsäikeen käynnistävä tapahtuma-funktio.

Pääsäie käynnistää Tcp-kuuntelijan, joka ottaa vastaan pääohjelmalta tulevat lämpötilamittaukset ja lähettää niistä aiheelliset tiedot kahdelle eri funktiolle, Temp() ja Anturi().

```
123 private void bgwMain_DoWork(object sender, DoWorkEventArgs e)
124 {
125     TcpListener serverSocket = new TcpListener(IPAddress.Any, Convert.ToInt32(txtPort.Text));
126     TcpClient clientSocket = default(TcpClient);
127 
128     serverSocket.Start();
129     while (true)
130     {
131     clientSocket = serverSocket.AcceptTcpClient();
132 
133     int requestCount = 0;
134     byte[] bytesFrom = new byte[63];
135     string dataFromClient = null;
136     Byte[] sendBytes = new byte[256];
137     string rCount = null;
138     requestCount = 0;
139     try
140     {
141         requestCount = requestCount + 1;
142         NetworkStream networkStream = clientSocket.GetStream();
143         networkStream.Read(bytesFrom, 0, 63);
144         dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
145         dataFromClient = dataFromClient.Substring(0, 63);
146 
147         this.Invoke(new MethodInvoker(delegate()
148         {
149             MainWindow._MainWindow.Temp(dataFromClient.Substring(2,4));
150 
151             MainWindow._MainWindow.Anturi(dataFromClient.Substring(0, 2));
152             if (timer2.Enabled == true) timer2.Stop();
153             timer.Start();
154         }));
155 
156         rCount = Convert.ToString(requestCount);
157         networkStream.Flush();
158 
159         networkStream.ReadByte();
160         networkStream.Close();
161     }
```

Kuva 31. Pääsäie ottaa tietoja vastaan ja lähettää ne tarvittaville funktioille.

```
95          public void Anturi(string place)
96          {
97              txtboxAnturi.Text = place;
98          }
99          public void Temp (string temp)
100         {
101             lblTemperature.Text = temp;
102             lblViimeinenMittaus.Text = DateTime.Now.ToString("HH:mm");
103
104             string seconds = Convert.ToString((Convert.ToInt32(txtLoppuLampo.Text) - Convert.ToInt32(temp) +
105                                                Convert.ToDouble(txtVakio1.Text)) / Convert.ToDouble(txtVakio2.Text));
106             double seconds_dbl = Convert.ToDouble(seconds);
107             int Seconds = Convert.ToInt32(Math.Round(seconds_dbl * 60, 0));
108             lblMinutes.Text = Convert.ToString(Seconds / 60);
109             lblSeconds.Text = Convert.ToString(Seconds % 60);
110         }
```

Kuva 32. Anturi- ja Temp-funktiot.

Anturi-funktio ei tee mitään muuta kuin asettaa PLACE-arvon ohjelman ikkunaan oikeaan kohtaansa.

Sen lisäksi, että Temp-funktio asettaa mitatun lämpötilan ja mittausajan ruudulle, se myös laskee jäljellä olevan ajan metallisulan käsittelyä varten ja asettaa tämän näkyviin.

Laskurin alaspäin laskemisesta huolehtii Timer-funktio, joka suoritetaan automaattisesti sekunnin välein ja on kehitetty juuri tätä tarkoitusta varten.

```
172 private void timer_Tick(object sender, EventArgs e)
173 {
174     if (timer2.Enabled == true) timer2.Stop();
175
176     lblMinutes.ForeColor = System.Drawing.Color.Black;
177     lblSeconds.ForeColor = System.Drawing.Color.Black;
178     label1.ForeColor = System.Drawing.Color.Black;
179     lblMinus.Visible = false;
180
181     lblSeconds.Text = Convert.ToString(Convert.ToInt32(lblSeconds.Text) - 1);
182     if (Convert.ToInt32(lblSeconds.Text) < 0 && Convert.ToInt32(lblMinutes.Text) < 0)
183     {
184         lblMinutes.Text = "00";
185         lblSeconds.Text = "00";
186     }
187     if (Convert.ToInt32(lblSeconds.Text) > 0 && Convert.ToInt32(lblMinutes.Text) >= Convert.ToInt32(txtMaxAika.Text))
188     {
189         lblMinutes.Text = txtMaxAika.Text;
190         lblSeconds.Text = "00";
191     }
192     {
193
194     }
195     if (Convert.ToInt32(lblSeconds.Text) == -1)
196     {
197         lblMinutes.Text = Convert.ToString(Convert.ToInt32(lblMinutes.Text) - 1);
198         lblSeconds.Text = "59";
199     }
200     if (Convert.ToInt32(lblSeconds.Text) == 0 && Convert.ToInt32(lblMinutes.Text) == 0)
201     {
202         timer.Stop();
203         timer2.Start();
204     }
205 }
```

Kuva 33. Timer-funktio.

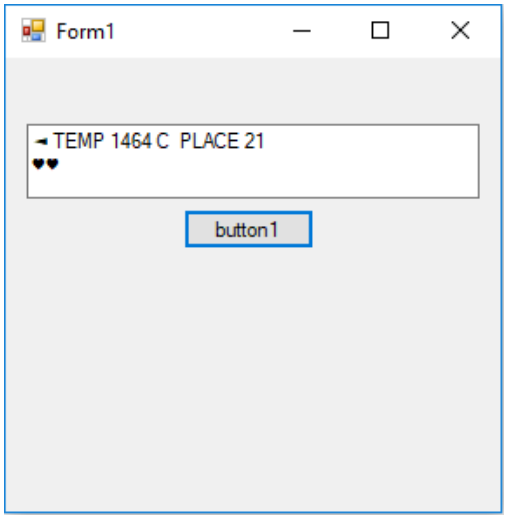
8. OHJELMIEN TESTAUS

8.1 Testialusta

Testialustana käytettiin samaa tietokonetta, jolla ohjelmisto kehitettiinkin. Testaukseen käytettiin myös Satakunnan Ammattikorkeakoulun Cisco-laboratorion tiloja ja laitteita. Ciscon kytkimeen asennettiin kehitysalustana toimineen koneen lisäksi neljä tietokonetta kiinni, joihin kolmeen asennettiin testiohjelma, jolla saatiin lähetettyä datapaketteja tiedonkeruuohjelmalle. Yhdelle koneelle asennettiin sivuohjelma ja näin saatiin testattua myös sen toimintaa. Näin saatiin simuloitua reaalimaailman olosuhteita.

8.2 Testiohjelma

Ohjelman testausta varten luotiin pieni, erillinen ohjelma, jolla lämpömittaria pystytään ‘emuloimaan’ lähettämällä samalaisia paketteja mitä lämpömittarilta tulisi. Näin vältyttiin siltä, että täytyisi käyttää fyysisiä mittareita. Ohjelma luo TCP/IP-yhteyden tiedonkeruuohjelmaan ja lähettää ruudulla näkyvän viestin ASCII-muodossa ennalta määrättyyn IP-osoitteeseen ja porttiin.



Kuva 34. Ohjelma, joka emuloi lämpömittaria lähettämällä paketteja pääohjelmalle.

```
22          byte[] buffer = new byte[63];
23
24          private void button1_Click(object sender, EventArgs e)
25          {
26              TcpClient client = new TcpClient();
27
28              string convert = textBox1.Text;
29              buffer = Encoding.UTF8.GetBytes(convert);
30              if (client.Connected == false)
31              {
32                 client.Connect("127.0.0.1", 8520);
33              }
34              NetworkStream serverStream = client.GetStream();
35              byte[] outStream = System.Text.Encoding.ASCII.GetBytes(textBox1.Text + "$");
36              serverStream.Write(outStream, 0, outStream.Length);
37              serverStream.Flush();
38              client.Close();
39          }
```

Kuva 35. Testiohjelman lähdekoodia.

8.3 Testaus pidemmällä aikavälillä

Tiedonkeruuohjelmaa pidettiin käytössä aina muutamia päiviä ja samalla seurattiin ovatko tietokantaan tallentuvat tiedot realistisia ja seurattiin tiedonkeruuohjelman tuottamia virheilmoituksia, jotka tallentuvat erilliseen tiedostoon.

```
30.9.2015 11:06:41

System.Net.NetworkInformation.PingException: An exception occurred during a Ping request. --->
System.ComponentModel.Win32Exception: Verkkosijaintiin ei saatu yhteyttä.
Katso Windowsin Ohjeesta lisätietoja verkon vianmäärityksestä
   at System.Net.NetworkInformation.Ping.InternalSend(IPAddress address, Byte[] buffer, Int32 timeout,
   PingOptions options, Boolean async)
   at System.Net.NetworkInformation.Ping.Send(IPAddress address, Int32 timeout, Byte[] buffer, PingOptions options)
   --- End of inner exception stack trace ---
   at System.Net.NetworkInformation.Ping.Send(IPAddress address, Int32 timeout, Byte[] buffer, PingOptions options)
   at CTKJ.MainWindow.bgwAnturiPing_DoWork(Object sender, DoWorkEventArgs e) in
   c:\Users\Omistaja\Documents\Visual Studio 2013\Projects\Testcase1\Testcase1\MainWindow.cs:line 572
```

Kuva 36. Esimerkki ohjelman tallentamasta virheilmoituksesta.

Ohjelmaa testattiin niin kauan, että voitiin olla verrattain varmoja ohjelman toimivuudesta. Sen jälkeen ohjelmisto asetettiin tuotannon käyttöön, jossa se on edelleen.

9. POHDINTA

Opin paljon ohjelmiston luonnin ja siitä tehdyn opinnäytetyön kirjoittamisen aikana. Varsinkin C#-ohjelmointikieli ja tietokannat tulivat tutuiksi. Myös se tosiasia realisoitui, että yritysmaailmassa alkuperäiset tavoitteet eivät yleensä ole lopulliset tavoitteet vaan asiat elää ja uusia ideoita syntyy työn edetessä.

Työ itsessään onnistui hyvin ja tavoitteet saavutettiin myös asiakkaan mielestä.

Ohjelmistoon olisi voinut lisätä vielä monia ominaisuuksia. Esimerkiksi kustomoidut tietokantahaut olisivat olleet yksi mahdollinen ominaisuus mutta se olisi todennäköisesti paisuttanut työn kokoa liikaa.

Tällä hetkellä ohjelmisto tekee mihin se on suunniteltu, eli kerää tietoa antureilta ja tallentaa sitä tietokantaan. Myös mittaustietojen edelleenlähetys on ahkerassa käytössä.

Tulevaisuudessa ohjelmistoa voisi laajentaa paljonkin, mikäli asiakkaan tarpeet muuttuisivat.

LÄHTEET

Componentan www-sivut. http://componenta.com/

Heraeuksen www-sivut. https://www.heraeus.com/ Viitattu 18.5.2017.

Heraeus Electro-Nite 2011. Instruction and Operating Manual Temperature Measuring Unit, Digitemp-E, Document Version: 4.05. Viitattu 18.5.2017.

Microsoftin www-sivut. https://www.microsoft.com/

Wikipedian www-sivut. https://en.wikipedia.org/