VYŠŠÍ ODBORNÁ ŠKOLA, STŘEDNÍ ŠKOLA,
CENTRUM ODBORNÉ PŘÍPRAVY
SEZIMOVO ÚSTÍ

ABSOLVENTSKÁ PRÁCE

Wi-Fi příjem a vizualizace dat od osmi čidel na obrazovce počíta-

če s OS Win_XP, Win_7 a Win_8

Sezimovo Ústí, 2013 **Erdt Martin**

Prohlášení

Prohlašuji, že jsem tuto absolventskou práci vypracoval samostatně za použití podkladů uvedených v přiloženém seznamu.

V Sezimově Ústí dne 25. 4. 2013

Podpis:……..……

Poděkování

Především bych rád poděkoval vedoucímu práce Ing. Jiřímu Bumbovi za jeho důsledné a zároveň vlídné vedení, podnětné nápady a čas, který věnoval mé práci. Poděkování patří rovněž panu Ing. Jiřímu Roubalovi, Ph.D. za jeho cenné komentáře k formátování tohoto textu. Dále bych rád poděkoval panu Mgr. Jaroslavovi Pejšovi za velmi cenné vzdělání v jazyce C#.

Děkuji všem mým blízkým za podporu během studia.

Anotace

Práce pojednává o softwaru pro příjem, zpracování a vizualizaci dat získaných až z 8 autonomních inteligentních čidel. Přenos dat probíhá bezdrátově prostřednictvím Wi-Fi pomocí protokolu TCP. Tento software je součástí zařízení pro monitorování vývoje teploty a dalších údajů na ploše letiště při závodech leteckých modelářů.

Annotation

This work is about the software for receiving, processing and visualization data collected from up to 8 autonomous intelligent sensors. Data are transferred wirelessly via Wi-Fi using TCP protocol. This software is part of device for monitoing the progress of temperature and other data on the airfield at the races of the air modelers.

Obsah

Absolutorní práce

Martin Erdt

1 Úvod

Současná doba se neobejde bez pokrokových technologií, které za mnoho z nás řeší velké množství práce. Drtivá většina technologií spadá do kategorie inteligentních systémů. Tato skupina technologií se vyznačuje digitalizací a leckdy i umělou inteligencí. Z tohoto je patrné, že řada systémů obsahuje nějaký software, který vykonává určitou logiku. Respektive reaguje na předem známé podněty. Kvalitní software je stále, vzhledem k času, potřebnému na jeho vývoj, pro mnoho subjektů finančně nedostupný. Na Vyšší odborné škole, Střední škole, Centru odborné přípravy Sezimovo Ústí čas od času takovýto software vzniká.

Tuto práci jsem si vybral na základě dlouhodobé zkušenosti a praxi v oblasti objektově orientovaného programování v jazyce C#. Zajímám se také o způsoby komunikace mezi zařízeními, což je nemalou součástí této práce.

Cílem této práce je vytvořit funkční program, který bude splňovat následující požadavky: příjem dat z maximálně 8 autonomních inteligentních čidel se zadaným zápisem dat, zpracování a vizualizace dat na obrazovce osobního počítače, notebooku, tabletu nebo mobilního telefonu.

2 Technologický postup

Aby práce byla úspěšně zakončena, je velmi důležité a podstatné dobré rozvržení projektu v programovacím jazyce C#, zvolit správný způsob komunikace, zpracování dat a vizualizaci. Projekt je rozvržen do 2 základních struktur. Struktury se nazývají **Core** a **GUI**. Struktura zvaná Core se zabývá logickou částí projektu například zpracováním přijatých dat. Zpracovaná data jsou z této struktury předány druhé struktuře, která se stará o bezchybné vykreslení těchto dat. Vzhledem k tomu, že tento projekt úzce souvisí s maturitní prací „Bezdrátové měření teploty“ pana Malenického ze školního roku 2012-2013 pana Malenického, je třeba pevně stanovit strukturu dat.

Software přijímá, zpracovává a vizualizuje data, která jsou odesílána z maximálně osmi autonomních inteligentních čidel rozmístěných po letišti. Existují 2 druhy čidel. Čidlo snímající teplotu [°C] a čidlo snímající rychlost [m/s] a směr větru [°]. Proto je velmi důležité rozlišit příchozí data z jednotlivých čidel. Tato čidla odesílají data každou sekundu na cílový počítač po bezdrátové technologii Wi-Fi pomocí protokolu TCP. Zasílaná data mají pevně stanovenou strukturu. Z vložených identifikačních údajů program zjistí, s kterým čidlem komunikuje, jaké hodnoty má očekávat a jak s nimi dále pracovat.

3 Software potřebný k vývoji

Pro vývoj této aplikace se nabízí několik **IDE.** Mezi dva významné a horké kandidáty patří **SharpDevelop** a **Visual Studio**. Obě tato IDE podporují jazyk C#.

3.1 SharpDevelop

SharpDevelop je přehledné IDE vývojové prostředí jazyků C# a VB.NET, které spadají pod platformu Microsoft .NET. Součástí programu je také například editor zdrojů, prohlížeč assembly a další součásti. Avšak není to plnohodnotné prostředí, které navrhla společnost Microsoft.

3.2 Visual Studio

Microsoft Visual Studio je vývojové prostředí (IDE) od Microsoftu. Může být použito pro vývoj konzolových aplikací a aplikací s grafickým rozhraním spolu s aplikacemi Windows Forms, webovými stránkami, webovými aplikacemi a webovými službami jak ve strojovém kódu, tak v řízeném kódu na platformách Microsoft Windows, Windows Mobile, Windows CE, .NET, .NET Compact Framework a Microsoft Silverlight.

Visual Studio obsahuje editor kódu podporující IntelliSense a refaktorování. Integrovaný debugger pracuje jak na úrovni kódu, tak na úrovni stroje. Další vestavěné nástroje zahrnují designer formulářů pro tvorbu aplikací sGUI, designer webu, tříd a databázových schémat. Je možné přidávat rozšíření, což vylepšuje funkčnost na téměř každé úrovni – od doplnění podpory pro verzovací systémy (jako Subversion a Microsoft Team Foundation Server) po nové nástroje, jako editory a vizuální designery pro doménově specifické jazyky, nebo nástroje pro další aspekty návrhu programu (jako klient Team Foundation Serveru Team Explorer).

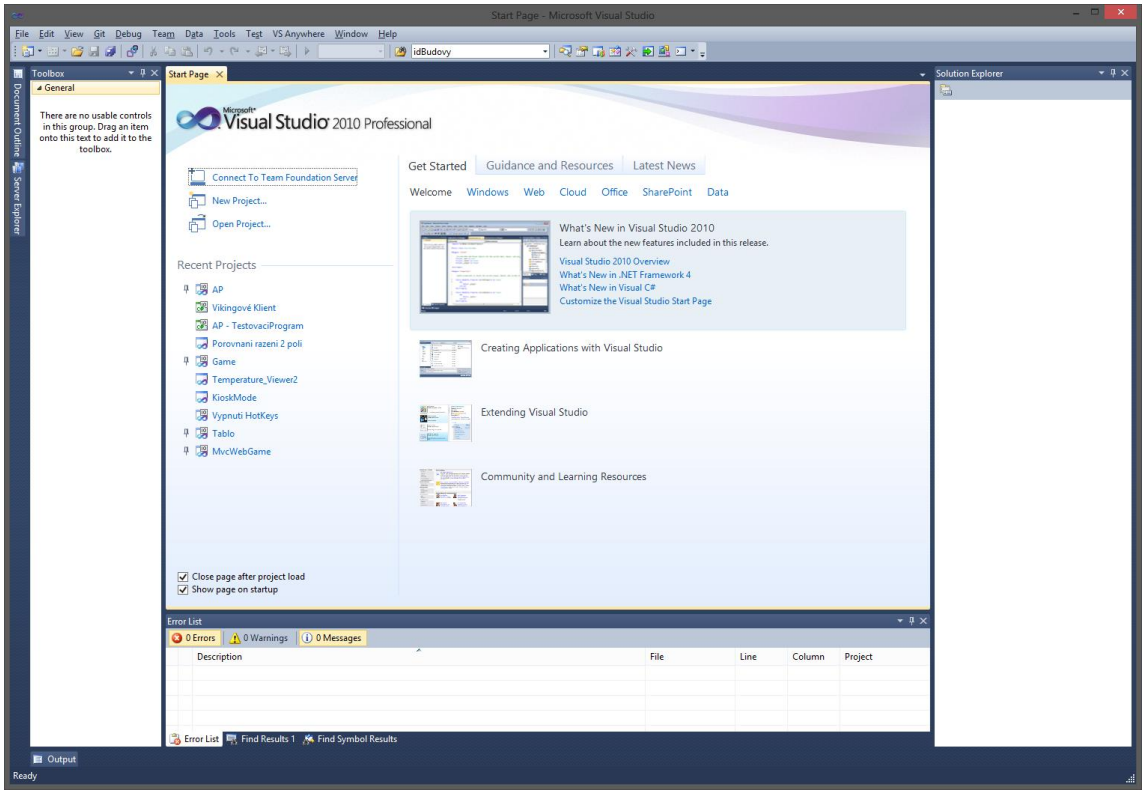
Visual Studio podporuje jazyky prostřednictvím jazykových služeb, což umožňuje, aby editor kódu a debugger podporoval jakýkoliv programovací jazyk. Mezi vestavěné jazyky patří C/C++ (použitím Visual C++), VB.NET (použitím Visual Basic

.NET) a C# (použitím Visual C#). Podpora dalších jazyků jako Oxygene, F#, Python a Ruby spolu s ostatními může být přidána jazykovými službami, které musí být nainstalovány zvlášť. Také je podporováno XML/XSLT, HTML/XHTML, JavaScript a CSS. Existují i verze Visual Studia pro určitý jazyk, které uživateli poskytují omezenější jazykové služby. Tyto individuální balíčky jsou Microsoft Visual Basic, Visual J#, Visual C# a Visual C++.

4 Založení projektu ve Visual Studio 2010

4.1 Úvodní obrazovka

Při spuštění Visual Studia 2010 Professional uživatele uvítá úvodní obrazovka. Ta je oproti předchozím verzím VS velmi přehledná.

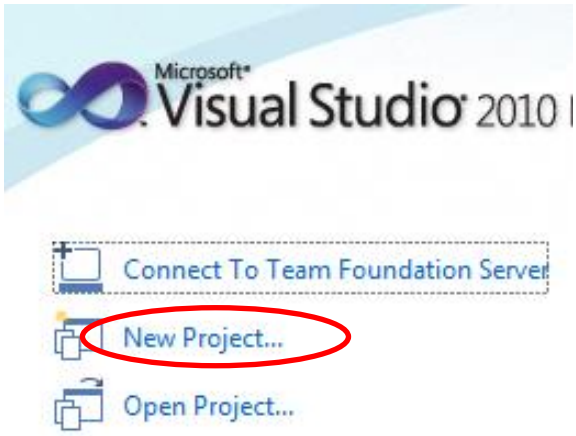


Obrázek 1 - Úvodní obrazovka Visual Studia 2010 Professional

Na této úvodní obrazovce VS nabádá k mnoha funkcím. Od výuky přes znovuotevření předešlých projektů, až po založení nového projektu. Odtud je možné do VS stáhnout přídavné moduly, které rozšiřují VS o nové funkce.

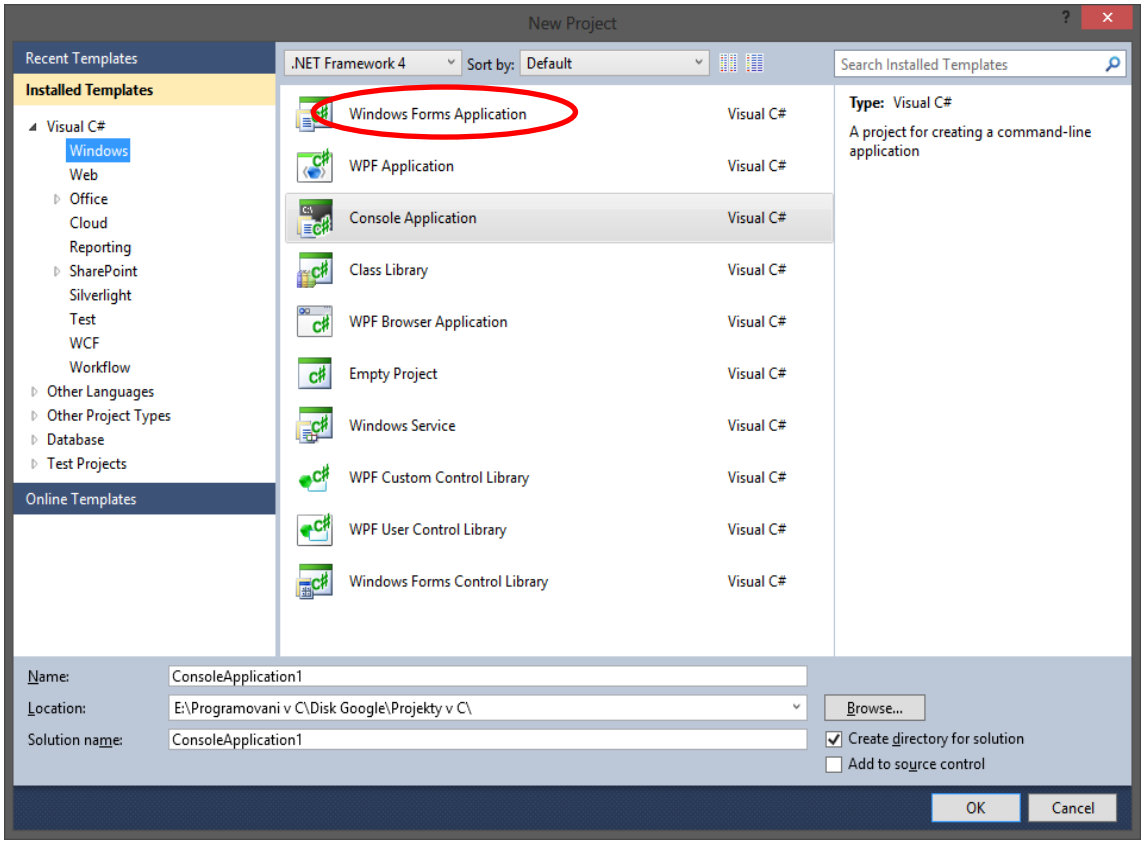
4.2 Založení nového projektu

Pro založení nového projektu je nutné kliknout na tlačítko New Project. To se nachází v levém horním rohu středového okna. Je to spíše odkaz než tlačítko.



Obrázek 2 - Tlačítko "New Project"

Po kliknutí na toto tlačítko se VS zeptá, jaký typ projektu bude programován a v jakém jazyce. VS nabízí velkou škálu projektů od konzolových aplikací, přes formulářové aplikace, aplikace pro mobil po webové aplikace. Pro účely této aplikace postačí formulářová aplikace (Windows Form Application).



Obrázek 3 - Okno s výběrem nového projektu

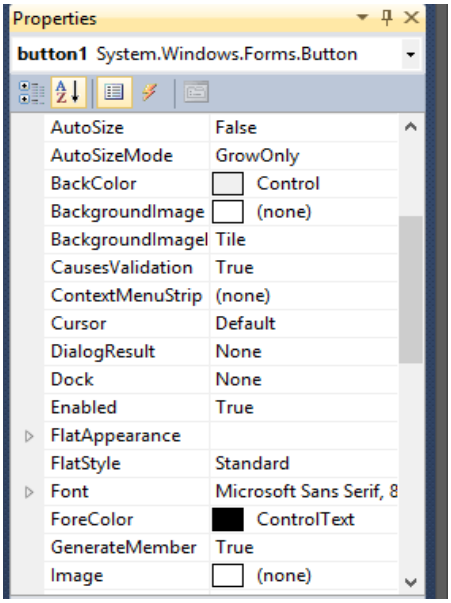
4.3 Formulářové okno pro tvorbu GUI

Ve VS je velmi snadné vytvořit grafickou stránku aplikace. Microsoft společně s VS dodává ohromnou škálu komponent, které stačí vybrat ze seznamu a přetáhnout je do okna. Komponenty jsou například tlačítka, seznamy, háčkovací okénka a spousty dalších. Každá z těchto komponent má své vlastnosti a události. Například vlastnosti komponenty Button (tlačítko) některé jsou:

Název vlastnosti	Popis vlastnosti	Hodnota
Name	Vlastnost Name obsahuje unikátní jméno tlačítka, které se již na tomto formuláři nesmí vyskytovat. Hodnotu této vlastnosti lze pojmenovat jakkoli, přesto se doporučují nějaká nepsaná pravidla	btnPotvrd
BackColor	Vlastnost určující pozadí tlačítka. Na výběr máme z velkého množství barev, popřípadě si můžeme namíchat vlastní barvu zadáním RGB.	Black

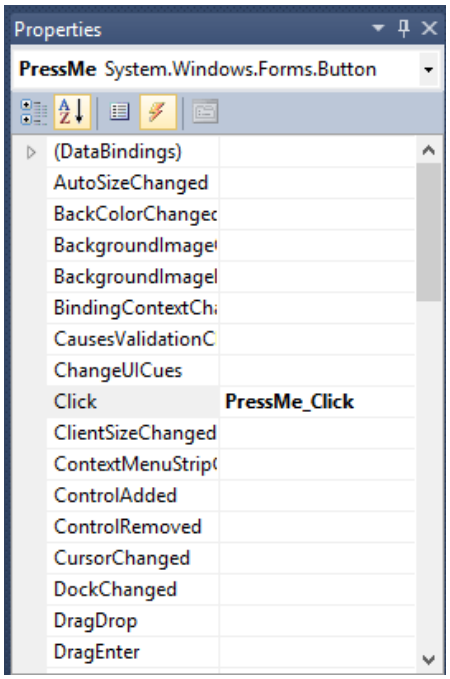
Tabulka 1 - Vlastnosti tlačítka

Existuje mnoho dalších vlastností v seznamu Properties (Vlastnosti). Na další stránce je výřez ze seznamu.



Obrázek 4 - Ukázka Vlastností tlačítka

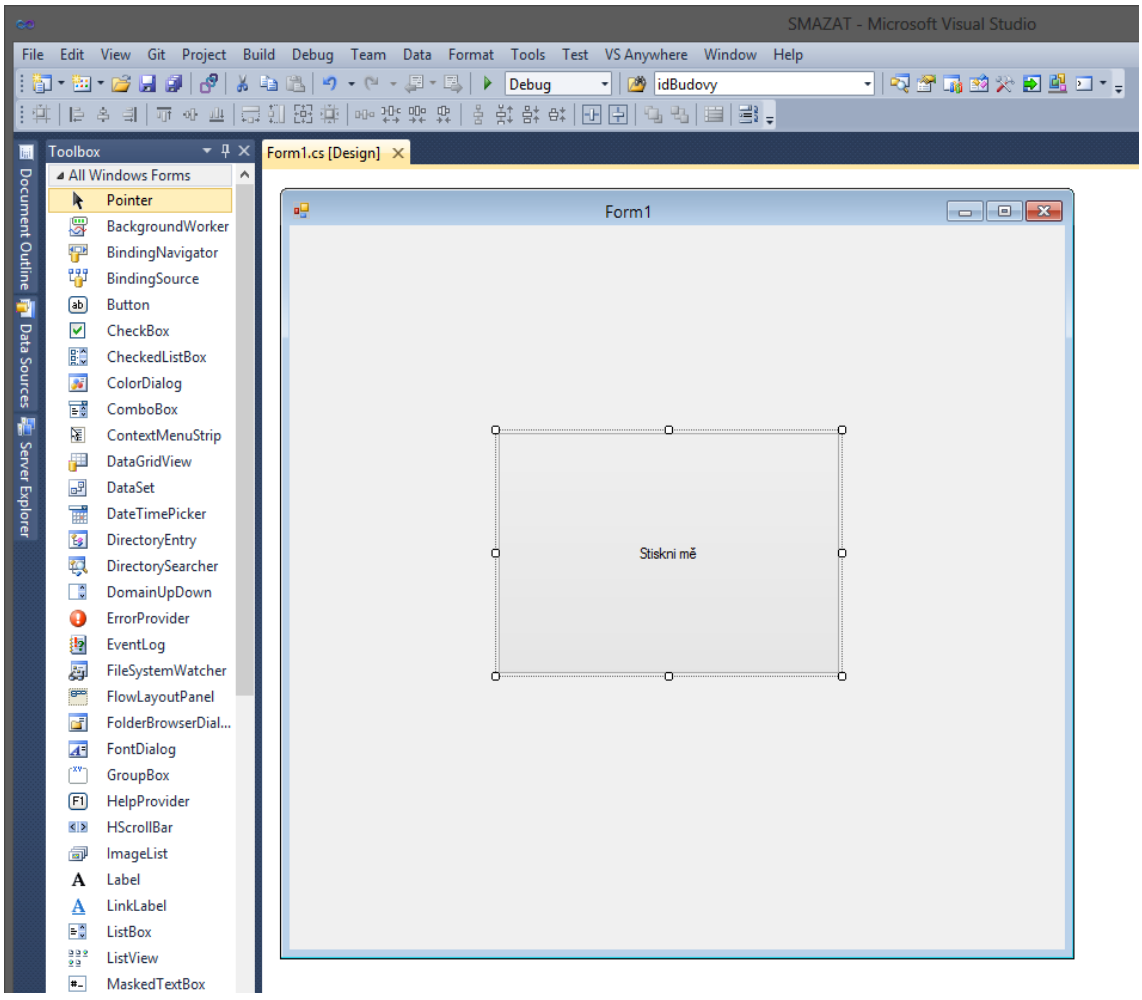
Velmi podobné jsou i události tlačítka. Pod pojmem událost si lze představit nějakou akci, například kliknutí na tlačítko, najetí myši na tlačítko, dvojklik na tlačítko a mnoho dalších. Seznam událostí vypadá takto:



Obrázek 5 - Ukázka událostí tlačítka

V tomto případě je vybrána událost **Click**, která vykoná nějakou akci po kliknutí na tlačítko. Událost se nyní jmenoje **PressMe_Click**, kde slovní spojení PressMe je hodnota z vlastnosti Name u tlačítka. Je možné si vytvořit vlastní název události u nějakého objektu, ale dělá se to velmi zřídka.

Nyní je třeba ukázat, jak vypadá formulářové okno v době návrhu aplikace. Na tomto okně je pro demonstraci vloženo tlačítko.

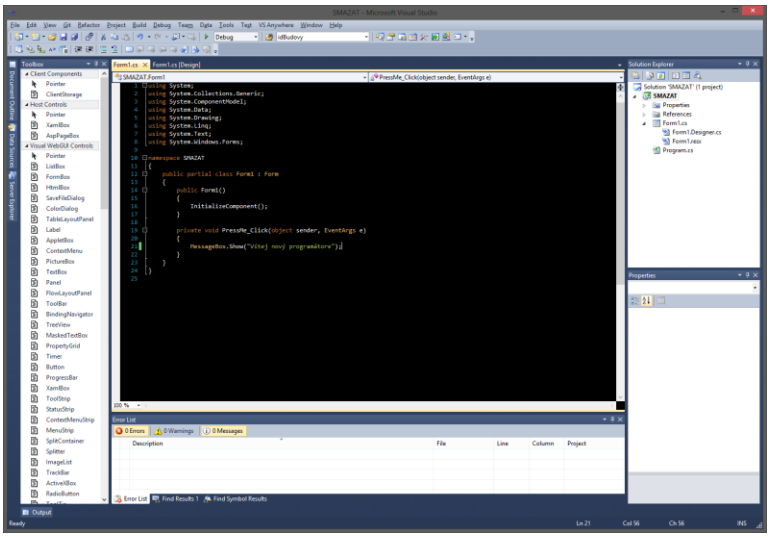


Obrázek 6 - Ukázka návrhu formuláře s umístěným tlačítkem

Na tomto obrázku je znázorněn formulář s velkým tlačítkem uprostřed, které má na sobě napsaný text „Stiskni mě“. Po levé straně formuláře je velký seznam zvaný Toolbox. Tento velký seznam obsahuje veškeré komponenty, které VS nabízí. V horní části je menu, které obsahuje veškeré možnosti VS.

4.4 Okno pro psaní kódu

Zde je znázorněné okno, kam se zapisuje kód, který nám vykonává potřebnou logiku programu. Tento kód je to, co programátor programuje. Barevné schéma není součástí VS.



Obrázek 7 - Ukázka okna pro psaní zdrojového kódu

Takto vypadá tedy okno, kam se zapisuje kód. Zápis kódu vypadá nějak takto:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SMAZAT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void PressMe_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Vítej nový programátore");
        }
    }
}
```

Třída 1 - Ukázka kódu

5 Řešení projektu

Samotné řešení projektu bylo rozděleno na 2 části. Na logickou část zvanou **Core** a grafickou část zvanou **GUI.** Obě tyto části spolu úzce souvisí a jedna bez druhé nemůže samostatně fungovat.

5.1 Core – jádro projektu

Core tvoří logickou část projektu. Zde probíhají veškeré negrafické operace, které je potřeba zpracovat před vykreslením dat na obrazovku. Obsahuje několik velmi důležitých tříd, kde jejich obsahem jsou metody.

5.1.1 Komunikace – výběr protokolu

Komunikace je základní stavební kámen pro přenos dat mezi čidly a počítačem. Pro tento účel se se nabízí 2 protokoly. Protokol **UDP** a protokol **TCP**. Každý z těchto uvedených protokolů má své kladné i záporné stránky.

Protokol UDP je znám jako nespolehlivý. Není konstruován tak, aby zajišťoval doručení dat a ani podobu při doručení. To znamená, že data se nám mohou ztratit, přijít zpřeházená nebo dokonce několikrát. Tento protokol se běžně používá tam, kde se vyžaduje jednoduchost. Jeho bezstavovost je užitečná zejména pro servery, které obsluhují velké množství klientů. Na oplátku zde není žádné řazení, žádné sledování spojení apod., což je velkou výhodou i nevýhodou oproti TCP.

Protokol TCP je spojově orientovaný protokol pro přenos dat. Tento protokol používá služby **IP** protokolu opakovaným odesíláním nespolehlivých paketů, takže při ztrátě paketu zajišťuje spolehlivost a přeuspořádáním paketů zajišťuje správné pořadí dat. Před započetím přijímání a odesílání dat je zapotřebí navázat spojení, které trvá do doby, než se klient odpojí nebo server přeruší naslouchání. Protože TCP je spojovaná transportní služba, musí se před odesíláním dat navázat spojení mezi klientem a serverem. K tomu slouží trojcestný handshaking. V průběhu navazování spojení se obě strany dohodnou na číslu sekvence a potvrzovacím čísle. Pro navázání spojení se odesílají da-

tagramy s nastavenými příznaky SYN a ACK. Jak z názvu (trojcestný handshaking) vyplývá, navázání spojení probíhá ve třech krocích:

- Klient odešle na server datagram s nastaveným příznakem SYN a náhodně vygenerovaným číslem sekvence (x), potvrzovací číslo = 0.
- Server odešle klientovi datagram s nastavenými příznaky SYN a ACK, potvrzovací číslo = x+1, číslo sekvence je náhodně vygenerováno (y)
- Klient odešle datagram s nastaveným příznakem ACK, číslo sekvence = x+1, číslo odpovědi = y+1.

Obě strany si pamatují číslo sekvence své i protistrany. Používají se totiž i pro další komunikaci a určují pořadí paketů. Když úspěšně proběhne trojcestný handshaking, je spojení navázáno a zůstane tak až do ukončení spojení. Ukončení spojení probíhá podobně jako jeho navázání. Používá se k tomu příznaků FIN a ACK:

- Klient odešle datagram s nastaveným příznakem FIN
- Server odpoví datagramem s nastaveným příznakem ACK
- Server odešle datagram s nastaveným příznakem FIN
- Klient odpoví s nastaveným příznakem ACK

Teprve po těchto čtyřech krocích je spojení ukončeno. Tímto se protokol TCP stal jasným a jediným kandidátem na použití v projektu a to díky garanci doručení dat a velké spolehlivosti.

Oba protokoly používají porty. Porty se používají k rozlišení, jaká aplikace právě komunikuje. Každá strana TCP spojení má přidruženo 16bitové bezznaménkové číslo portu (existuje 65535 portů) přidělené aplikaci a to si myslím, že by mělo být jednoznačně dostačující.

5.1.1.1 Třída Communication

Jak je již z názvu patrné, jedná se o třídu, která zajišťuje komunikaci mezi počítačem a čidlem. Tato třída pro každé připojené čidlo vytvoří vlastní vlákno, po kterém bude čidlo komunikovat s počítačem. Tím se zajistí, že může komunikovat několik, klidně i desítky, čidel najednou aniž by to mělo nějaký dopad na zpomalení aplikace, případně na zpomalení doručování dat. Při odpojení čidla, vlákno zanikne. Tato třída vypadá takto:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.Threading;
using System.Net;

namespace Core
{
    public class Communication
    {
        TcpListener server = null;
        IPAddress localAddr = IPAddress.Parse("127.0.0.1");

        Thread mainLoopThread;

        /// <summary>
        /// Konstruktor: Vytvoří server, který bude poslouchat
        /// na portu <c>port</c> a na IP <c>IP</c>.
        /// </summary>
        /// <param name="port">Číslo portu.</param>
        /// <param name="IP">IP pro naslouchání</param>
        public Communication(int port, string IP)
        {
            IPAddress localAddr = IPAddress.Parse(IP);
            server = new TcpListener(localAddr, port);
        }

        /// <summary>
        /// Konstruktor: Vytvoří server, která bude
        /// naslouchat na portu <c>port</c>.
        /// </summary>
        /// <param name="port"></param>
        public Communication(int port)
        {
            server = new TcpListener(localAddr, port);
        }

        /// <summary>
        /// Zahájí naslouchání.
        /// </summary>
        public void Start()
        {
            mainLoopThread = new Thread(MainLoop);
            mainLoopThread.IsBackground = true;
            mainLoopThread.Start();
        }
```

Třída 2 - Communication (1. část)

```
public void Stop()
 {
     server.Stop();
     mainLoopThread.Abort();
 }

private void MainLoop()
{
    try
    {
        server.Start();

        while (true)
        {
            TcpClient client = server.AcceptTcpClient();

            // po připojení klienta se mu přidělí vlákno
            Thread t=new Thread(new ParameterizedThreadStart
                                (CommunicationThreadLoop));
            t.Start(client); // spustí vlákno
        }
    }
    catch (ThreadAbortException)
    {
        // neděláme nic
    }
}

private void CommunicationThreadLoop(object clientObj)
{
    TcpClient client = clientObj as TcpClient;

    byte senzorID = 0;

    try
    {
        // buffer pro čtení dat ze sítě
        Byte[] bytes = new Byte[256];

        // získáme stream pro čtení a zápis dat ze/do sítě
        NetworkStream stream = client.GetStream();

        int dataLength;

        // přečteme ID senzoru
        while ((dataLength = stream.Read(bytes, 0, 1)) != 0)
        {
            senzorID = bytes[0];
            FireSensorConnected(senzorID);
            break;
        }

        while ((dataLength = stream.Read(bytes, 0, bytes.Length)) != 0)
        {
            var msg = MessageParser.Parse(senzorID, bytes);
            FireDataReceived(msg);
        }
     }
     catch
     {
         FireSensorDisconnected(senzorID);
     }
     client.Close();
 }
```

Třída 3 - Communication (2. část)

```
        /// <summary>
        /// Metoda události pro připojení senzoru
        /// </summary>
        /// <param name="sensorID">ID seznzoru</param>
        protected void FireSensorConnected(byte sensorID)
        {
                    if (SensorConnected != null)
                        SensorConnected(this, new SensorConnectedEventArgs(sensorID));
        }

        /// <summary>
        /// Metoda události při odpojení senzoru
        /// </summary>
        /// <param name="sensorID">ID seznzoru</param>
        protected void FireSensorDisconnected(byte sensorID)
        {
                    if (SensorDisconnected != null)
                        SensorDisconnected(this, new SensorDisconnectedEventArgs(sensorID));
        }

        /// <summary>
        /// Metoda události při přijatých datech
        /// </summary>
        /// <param name="message">Data</param>
        protected void FireDataReceived(DataMessage message)
        {
                    if (DataReceived != null)
                        DataReceived(this, new DataReceivedEventArgs(message));
        }

        /// <summary>
        /// Událost vyvolaná v případě, že se připojí nějaké čidlo.
        /// </summary>
        public event EventHandler<SensorConnectedEventArgs> SensorConnected;

        /// <summary>
        /// Událost vyvolaná v případě, že přijdou nějaká data ze sítě (od čidla).
        /// </summary>
        public event EventHandler<DataReceivedEventArgs> DataReceived;

        /// <summary>
        /// Událost vyvolaná v případě, že se odpojí nějaké čidlo.
        /// </summary>
        public event EventHandler<SensorDisconnectedEventArgs> SensorDisconnected;
    }
}
```

Třída 4 - Communication (3. část)

Tato třída obsahuje několik metod, které zajišťují bezproblémový chod komunikace. Metody jsou takové malé podprogramy, které vykonávají určitou část logiky.

5.1.1.1.1 Konstruktory třídy

Konstruktory třídy **Communication** jsou nazvané naprosto stejně (kdyby nebyly, nejednalo by se o konstruktory) a jsou veřejné, aby je bylo možné volat i z jiných míst kódu.

```
/// <summary>
/// Konstruktor: Vytvoří server, který bude
/// poslouchat na portu <c>port</c> a na IP <c>IP</c>.
/// </summary>
/// <param name="port">Číslo portu.</param>
/// <param name="IP">IP pro naslouchání</param>
public Communication(int port, string IP)
{
    IPAddress localAddr = IPAddress.Parse(IP);
    server = new TcpListener(localAddr, port);
}

/// <summary>
/// Konstruktor: Vytvoří server, která bude
/// naslouchat na portu <c>port</c>.
/// </summary>
/// <param name="port"></param>
public Communication(int port)
{
    server = new TcpListener(localAddr, port);
}
```

Metoda 1 - Konstruktory pro Communication

V této třídě jsou dva konstruktory a to díky tomu, že tento software byl vyvinut ještě dříve, než byla zkonstruována čidla. První konstruktor dostane na vstupu dva parametry port a IP adresu. Tyto parametry určují na jaké IP adrese a portu má program naslouchat klientům. Tento konstruktor je použit ve verzi pro ostrý provoz s čidly. Druhý konstruktor je pouze pro testovací účely, kde má pevně danou IP adresu na lokální IP, což je **127.0.0.1** a nastavuje se mu pouze port, který je rovněž předem nastaven na **9999**. Lokální IP je umístěna v proměnné: „***localAddr***“. Tento konstruktor je použit pro testovací účely, kdy místo reálného čidla je další program, který simuluje čidlo.

5.1.1.1.2 Metoda pro zahájení naslouchání

Nyní je čas započít naslouchání. Tato metoda spustí server a ten začne naslouchat jednotlivým čidlům ve vlastním vlákně, které běží na pozadí.

```
/// <summary>
/// Zahájí naslouchání.
/// </summary>
public void Start()
{
    mainLoopThread = new Thread(MainLoop);
    mainLoopThread.IsBackground = true;
    mainLoopThread.Start();
}
```

Metoda 2 - Zahájení naslouchání

Nejprve vytvoříme instanci nového vlákna, které poběží na pozadí aplikace. Toto vlákno bude obstarávat, zda se nějaký klient (čidlo) připojil nebo odpojil. Vytvořené vlákno volá metodu pro odchytávání připojených čidel. Metoda se jmenuje „*MainLoop*“. Díky běhu událostí ve vlastním vlákně je zajištěn plynulý chod aplikace.

5.1.1.1.3 Metoda pro ukončení naslouchání

Kdybychom tuto metodu vynechali, aplikace by mohla při vypnutí zahlásit chybu. Vše, co ve světě počítačů otevřeme, je dobré i zavřít.

```
/// <summary>
/// Ukončí naslouchání
/// </summary>
public void Stop()
{
    server.Stop();
    mainLoopThread.Abort();
}
```

Metoda 3 - Ukončení naslouchání

Metoda nedělá nic jiného, než že v konstruktoru, kde je vytvořen server, zastaví činnost a zruší vlákno „**mainLoopThread**“.

5.1.1.1.4 Odchytávání připojených čidel

Metoda se nazývá MainLoop, což v překladu znamená Hlavní smyčka. Tato smyčka běží na pozadí aplikace ve vlastním vlákně, které se jmenuje mainLoopThred. Metoda funguje jako nekonečný automat, tedy běží neustále dokola až do doby ukončení naslouchání.

```
/// <summary>
/// Hlavni smyčka vlákna, které bude přijímat
/// připojení klientů.
/// </summary>
private void MainLoop()
{
    try
    {
        server.Start();

        while (true)
        {
            // tady se to zablokuje a ceka to na pripojeni klienta
            TcpClient client = server.AcceptTcpClient();
            // po připojení klienta se mu přidělí vlákno
            Thread t = new Thread(new ParameterizedThreadStart(CommunicationThreadLoop));
            t.Start(client); // spustí vlákno
        }
    }
    catch (ThreadAbortException)
    {
        // nedelame nic
    }
}
```

Metoda 4 - Odchytávání připojených čidel

Celá metoda je umístěna pro jistotu v takzvaném „try catch bloku“, který nedělá nic jiného, než že se snaží o spuštění kódu v části **try**. Nezdaří-li se to (mělo by se to zdařit vždy), vykoná se kód v části **catch**. Je to patrné i z názvů jednotlivých částí. Try znamená zkusit a catch je zachytit. Tedy zkus něco, když se to nezdaří, zachyť výjimku.

Zde je vidět, že tady se spouští server jako takový. Následuje nekonečná smyčka, která naslouchá případným zájemcům o připojení. Po připojení čidla, se pro toto čidlo vytvoří nové vlákno, na kterém bude komunikovat. Nově vytvořené vlákno tedy obsahuje klíčovou metodu CommunicationThreadLoop, která nám zachytává zprávy od čidel.

5.1.1.1.5 Zachycení zpráv od jednotlivých čidel

Úkolem této metody je příjem dat zaslaných jednotlivými čidly. Tato metoda běží pro každé připojené čidlo ve vlastním vlákně. Z tohoto tvrzení je tedy patrné, že existuje tolik vláken, kolik je připojených čidel. Každé čidlo má přiděleno vlastní vlákno, které přijímá jeho data. Takto vypadá metoda:

```
/// <summary>
/// Naslouchání klientovi
/// </summary>
/// <param name="clientObj"></param>
private void CommunicationThreadLoop(object clientObj)
{
    TcpClient client = clientObj as TcpClient;
    byte senzorID = 0;

    try
    {
        // buffer pro čtení dat ze sítě
        Byte[] bytes = new Byte[256];

        // získame stream pro čtení a zápis dat ze/do síte
        NetworkStream stream = client.GetStream();
        int dataLength;

        // precteme ID senzoru
        while ((dataLength = stream.Read(bytes, 0, 1)) != 0)
        {
            senzorID = bytes[0];
            FireSensorConnected(senzorID);
            break;
        }

        while ((dataLength = stream.Read(bytes, 0, bytes.Length)) != 0)
        {
            var msg = MessageParser.Parse(senzorID, bytes);
            FireDataReceived(msg);
        }
    }
    catch
    {
        FireSensorDisconnected(senzorID);
    }
    client.Close();
}
```

Metoda 5 - Zachycení zpráv od jednotlivých čidel

Na první pohled se tato metoda může zdát jednoduchá, ale opak je pravdou, zde už začíná přituhovat. Bylo by nesmyslné zde popisovat tuto metodu řádek po řádku, a tak tuto metodu popíšu trochu jinak. V metodě existuje buffer, což je v překladu zásobník. Do tohoto zásobníku se budou ukládat data. Následuje cykl, který přečte ID senzoru. Pokud je ID senzoru platné, uložíme si ho do proměnné senzorID a zavoláme událost FireSensorConnected s parametrem senzorID, což je naše zachycené ID. Tato událost nám zajistí, že se nám zobrazí nově připojený senzor v připojených zařízeních. A pak teprve následuje čtení toku dat.

Čtení probíhá stylem, že dokud je co číst, tak to ukládáme do bufferu, který jsme si nazvali **bytes**. Dalo by se prohlásit, že toto je nekorektní. Může se stát, že se přečte jen část zprávy. V praxi se to stává opravdu jen velmi zřídka – o to je to horší. Naopak

se může přečíst více zpráv najednou. Proto je třeba přečtená data postupně číst a zapisovat do bufferu a ihned co v bufferu budou alespoň 4 bajty (délka zprávy), tak tyto první 4 bajty zpracovat jako zprávu. Po zpracování dat se tyto 4 bajty z bufferu odstraní, a pokud jsou zaznamenány ještě další bajty, ponechají se v bufferu, dokud není načtena další celá zpráva. Toto dělá druhý cyklus, který pak volá událost **FireDataReceiver**.

5.1.1.1.6 Události pro připojení i odpojení čidel a příjem dat.

Všechny tyto události nám zaručují, že program bude neustále aktuální. Připojí-li se čidlo, nemusíme pravidelně testovat, zda se nějaké čidlo připojilo, ale událost sama upozorní systém (program), že nastala nějaká změna v připojení a systém už ví, že si má zažádat o nový seznam čidel. Takto obdobně to funguje u všech událostí, proto není nutné je dále rozepisovat a stačí uvést kód pro danou událost.

```
protected void FireSensorConnected(byte sensorID)
{
    if (SensorConnected != null)
        SensorConnected(this, new SensorConnectedEventArgs(sensorID));
}

protected void FireSensorDisconnected(byte sensorID)
{
    if (SensorDisconnected != null)
        SensorDisconnected(this, new SensorDisconnectedEventArgs(sensorID));
}

protected void FireDataReceived(DataMessage message)
{
    if (DataReceived != null)
        DataReceived(this, new DataReceivedEventArgs(message));
}
public event EventHandler<SensorConnectedEventArgs> SensorConnected;
public event EventHandler<DataReceivedEventArgs> DataReceived;
public event EventHandler<SensorDisconnectedEventArgs> SensorDisconnected;
```

Metoda 6 - Události pro připojení i odpojení čidel a příjem dat

5.1.1.2 Třída SensorConnectedEventArgs

Tato třída není až tak plnohodnotnou třídou. Neobsahuje mnoho metod ani nevykonává nějakou specifickou část logiky. Jedná se o třídu, která se chová jako parametr k události „Připojení senzoru“, která nastane v okamžiku připojení nějakého čidla. Třída je velmi krátká a vypadá takto:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    /// <summary>
    /// Parametr události "připojení senzoru". Obsahuje ID senzoru, který se připojil.
    /// </summary>
    public class SensorConnectedEventArgs : EventArgs
    {
        public byte SensorID { get; private set; }

        public SensorConnectedEventArgs(byte sensorID)
        {
            SensorID = sensorID;
        }
    }
}
```

Třída 5 - SensorConnectedEventArgs

5.1.1.3 Třída SensorDisconnectedEventArgs

Téměř identická třída jako třída: **SensorConnectedEventArgs**. Znovu se jedná o parametr, ale tentokrát pro událost odpojení senzoru. Tato třída, stejně jako předcházející třída, obsahuje pouze ID senzoru. Obě tyto třídy se používají v metodách, které jsou popsány v části: **5.1.1.1.6 Události pro připojení i odpojení čidel a příjem dat**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    /// <summary>
    /// Parametr události "odpopojení senzoru". Obsahuje ID senzoru, který se odpojil.
    /// </summary>
    public class SensorDisconnectedEventArgs : EventArgs
    {
        public byte SensorID { get; private set; }

        public SensorDisconnectedEventArgs(byte sensorID)
        {
            SensorID = sensorID;
        }
    }
}
```

Třída 6 - SensorDisconnectedEventArgs

5.1.1.4 Třída DataReceivedEnevtArgs

Obdobně, jako v předcházejících dvou případech se nejedná o plnohodnotnou třídu. Je to také parametr, ale nyní pro událost příchozí zprávy. Tato třída obsahuje zprávu, kterou zaslalo nějaké čidlo. I tato třída se používá v části: 5.1.1.1.6.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    /// <summary>
    /// Parametr události "připojení senzoru". Obsahuje zprávu, kterou čidlo odeslalo.
    /// </summary>
    public class DataReceivedEventArgs : EventArgs
    {
        public DataMessage Message { get; private set; }

        public DataReceivedEventArgs(DataMessage message)
        {
            Message = message;
        }
    }
}
```

Třída 7 - DataReceivedEventArgs

5.1.1.5 Třída DataMessage

V tomto případě se jedná o mateřskou třídu, která obsahuje pouze parametry. Mateřská třída je taková, ze které můžeme dědit nějaké parametry/vlastnosti. Z této třídy dědíme dva parametry **SensorID** i **BatteryInfo**. Oba tyto parametry dědíme do dalších tříd s názvy: **WindMeterDataMessage** a **ThermometerDataMessage**.

Dědit můžeme pouze ty vlastnosti, které jsou stejné. Bylo by zbytečné v každé třídě pro jednotlivé čidlo psát společné prvky. Zabralo by to nejen čas, ale i případné úpravy aplikace by byly obtížné. Takto máme společné prvky v jedné třídě a můžeme je snadno najít a upravit.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    public abstract class DataMessage
    {
        public byte SensorID { get; set; }
        public byte BatteryInfo { get; set; }
    }
}
```

Třída 8 - DataMessage

5.1.1.6 Třída ThermometerDataMessage

Tato třída je dceřinou třídou od třídy **DataMessage**. Obsahuje dodatečné informace ke konkrétnímu čidlu. Jedná se o čidlo na měření teploty. V této třídě je patrné, v jakém formátu přicházejí data od čidel teploty. Formát dat, lze říci rámec.

ID čidla	Teplota1	Teplota2	Stav baterie

Tabulka 2 - Schéma příchozích dat od teploměru

- **ID čidla**: (identifikační číslo) může nabývat 256 hodnot od 0 do 255
  - Hodnoty 2 – 200 reprezentují čidlo teploty, zbylá jsou vyhrazena pro čidlo větru
  - **Teplota1** i **Teplota2**: Jedná se o takzvané binární číslo se znaménkem.
  - Ukázka 1: 0000 0000 0000 1000 = +0,5°C
  - Ukázka 2: 1111 1111 1111 1000 = -0,5°C
    - Prvních 5 bitů udává znaménko
    - Teplota
    - Teplota
    - Teplota za desetinnou čárkou
- **Stav baterie**: Kapacita baterie od 100[%] do 0[%].

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    class ThermometerDataMessage : DataMessage
    {
        public float Temperature { get; set; }
        public override string ToString()
        {
            return string.Format("TMSG id={0} batt={1} temp={2:0.000}", this.SensorID,
                                 this.BatteryInfo, this.Temperature);
        }
        internal static DataMessage Parse(byte[] data)
        {
            var msg = new ThermometerDataMessage();
            msg.SensorID = data[0];
            msg.Temperature = (float)BitConverter.ToInt16(data, 1) / 16;
            msg.BatteryInfo = data[3];
            return msg;
        }
    }
```

Třída 9 - ThermometerDataMessage

5.1.1.7 Třída WindMeterDateMessage

Tato třída je velmi podobná třídě: 5.1.1.6, avšak odlišně pracuje a jiným způsobem vyhodnocuje prostřední 2 bajty. Rovněž je tato třída dceřinou třídou třídy **DataMessage**. Má velmi podobný rámec (uspořádání dat) jako předchozí třída.

ID čidla	Rychlost větru	Směr větru	Stav baterie

Tabulka 3 - Schéma příchozích dat od větroměru

- **ID čidla**: stejné jako u předchozí třídy: 5.1.1.6
- **Rychlost větru**: Udáván v [m/s], bajt může nabývat hodnoty od 0 do 255.
- **Směr větru**: Udáván v [°] a může nabývat 16 hodnot, od 0 do 15. Kde číslo udává násobek 22,5°.
- **Stav baterie**: stejné jako u předchozí třídy: 5.1.1.6

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    class WindMeterDataMessage : DataMessage
    {
        public float WindSpeed { get; set; }
        public float WindWay { get; set; }
        public override string ToString()
        {
            return string.Format("WMSG id={0} batt={1} speed={2:0.000} m/s way={3:0.000}°",
                                 this.SensorID, this.BatteryInfo, this.WindSpeed,
                                this.WindWay);
        }
        internal static DataMessage Parse(byte[] data)
        {
            const float FULL_ANGLE = 360f;
            const float STEP_COUNT = 16;
            var msg = new WindMeterDataMessage();
            msg.SensorID = data[0];
            msg.WindSpeed = data[1];
            msg.WindWay = data[2] / 16 * (FULL_ANGLE / STEP_COUNT);
            msg.BatteryInfo = data[3];
            return msg;
        }
    }
}
```

Třída 10 - WindMeterDataMessage

Z těchto dvou tříd by mělo být naprosto jasné, za jakým účelem je použita dědičnost. Kdyby nebyla použita dědičnost, nezachovali bychom paradigma OOP a zároveň by budoucí rozšiřování programu bylo daleko složitější. Takto každá třída obsahuje pouze data/informace k jedné konkrétní věci a lze ji snadno upravit nebo nahradit, aniž by to ovlivnilo jiné části programu.

5.1.1.8 Třída MessageParser

Po připojení čidla je nutné zjistit, jaký typ čidla se připojil. V našem případě se mohou připojit pouze 2 typy čidel, čidlo na měření větru a čidlo na měření teploty. Na základě této třídy zjistíme čidlo, které čidlo komunikuje, a data od tohoto čidla vracíme pomocí příslušné třídy, která je pro konkrétní typ čidla navržena. Tato třída opět demonstruje sílu objektově orientovaného programování, kde případná změna v jakékoli části programu neovlivní ostatní části.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    class MessageParser
    {
        public static DataMessage Parse(byte sensorID, byte[] data)
        {
            if (sensorID >= 2 && sensorID <= 200)
                return ThermometerDataMessage.Parse(data);
            else if(sensorID >= 201 && sensorID <= 250)
                return WindMeterDataMessage.Parse(data);
            else
                throw new Exception(string.Format("Neznámý typ senzoru s ID = {0}.",
                                                  sensorID));
        }
    }
}
```

Třída 11 - MessageParser

Ukázka třídy však pojednává ještě o třetí možnosti, která je nastavená jako zakázaný stav. Pokud přijde ID senzoru od 2 do 200, jedná se o teploměr a očekáváme data v určitém formátu. Pokud přijde ID senzoru od 201 do 250, hovoříme o senzoru na snímání větru. Pokud přijde ID senzoru, které není obsažené v uzavřené množině povolených ID pro teploměr nebo senzor větru, mluvíme o neznámém typu senzoru, a tedy zakázaném stavu.

5.1.2 Dočasné uchování přijatých dat

K tomu, aby bylo možné data vykreslovat se stanovenou zpětnou dobou, je důležité mít data uchovaná po dobu jejich platnosti. Tato aplikace uchovává data po dobu 5 minut, což je 300 sekund. Data starší než 300 sekund jsou již nepotřebná a proto jsou nahrazována novými daty. Tento úkol lze realizovat několika způsoby. Jedním z nich je použití databáze, což je velmi nevhodný způsob použití. Anebo nekonečnou smyčkou (kruhovým seznamem), což je v tomto případě velmi efektivní a žádoucí.

V případě použití databáze se program rozroste o nemalý počet tříd a metod. Databáze mohou být v tomto směru pomalé, zvláště je-li uskladněno ohromné množství dat. Výsledkem tohoto by se mohlo stát, že vykreslení bude pomalé a ojediněle i nepřesné. Protože bychom museli porovnávat sloupec, který by obsahoval data typu DateTime, kde porovnání již není stejně rychlé jako porovnání 2 čísel. Následoval by další problém, který by musel selektovat řádky dat z databáze pomocí podmínky WHERE, která by nám stanovovala od jakého data a času do jakého data času máme řádky vytáhnout. To je další nemalé porovnávání, kde musíme projít celou ohromnou databázi (všechny záznamy) a vybrat pouze 300 dat, která vyhovují našim podmínkám. Je samozřejmé, že každý z těchto záznamů musíme nejprve porovnat. A představa databáze, kde je uloženo 864000 dat za 10 dní, a to pouze pro jedno čidlo je velmi, velmi nepříjemná, protože čidel máme až 8 + čidlo větru.

Použití kruhového seznamu je daleko efektivnější metodou k řešení tohoto problému. Kruhový seznam se vyznačuje pevně danou délkou dat. Díky tomu nepotřebujeme proměnlivou paměť, ale pouze přesně stanovenou velikost, což má za následek setření s prostředky. Další velmi podstatnou výhodou je, že data se nemusí hledat. Není potřeba žádné třídění, žádné podmínky, žádná WHERE. Pro každé čidlo je zde jeden seznam a do grafu vykreslujeme všechna data obsažená v seznamu. Když je seznam plný, tedy je v něm uloženo 300 dat, tak další data se zapíší na 1 místo. Proč? Protože na 1. místě jsou už data starší 300 sekund, a proto již není důležité si je pamatovat. Proto je možné zapsat nová data na 1. místo.

5.1.2.1 Ukázka kruhového seznamu

Aby se kruhový seznam snáze demonstroval, byla nastavena jeho velikost na 6 sekund namísto 300. Seznam nemá při vytvoření žádné hodnoty a ukazatel počátku seznamu je také prázdný. Existují pouze indexy, které reprezentují, jak velký seznam bude. V našem případě bude velký 6 sekund. Ukazatel konce je velmi důležitý, aby program věděl, kde data v seznamu jsou nejstarší. Ukazuje na index. Tím docílíme správného vykreslování dat. Kdyby program neměl přehled o konci dat, tak by nebylo možné data vykreslovat ve správném pořadí, tedy v takovém, ve kterém přišla.

Index:	**0**	**1**	**2**	**3**	**4**	**5**
Hodnota:						
Ukazatel poč.:						

Tabulka 4 - Kruhový seznam: prázdný

Nyní se budou zapisovat hodnoty: **5**, **2**, **7**, **6**, **1**, **9**, **3**, **8**, **7**, **0**

Index:	**0**	**1**	**2**	**3**	**4**	**5**
Hodnota:	5	2	7	6	1	9
Ukazatel k.:	0					

Tabulka 5 - Kruhový seznam: prvních 6 dat

Z nějakého konkrétního čidla přišlo prvních 6 naměřených teplot, které se zapsali na všech volných míst v sezamu. Na tomto není nic neobvyklého. Ukazatel konce je nastaven na index 0, protože zde ten seznam začal.

Teď se zapíše další hodnota: **3**

Index:	**0**	**1**	**2**	**3**	**4**	**5**
Hodnota:	3	2	7	6	1	9
Ukazatel k.:	1					

Tabulka 6 - Kruhový seznam: Sedmé data, posunutí ukazatele

Všimněte si, že se hodnota zapsala na index 0, tedy seznam se plní opět od začátku. Proto je důležité posunout ukazatel konce na nejstarší hodnotu v seznamu, což je nyní index 1.

A teď ukázka, jak to vypadá, když se zapíší všechna data: **8, 7, 0**

Index:	**0**	**1**	**2**	**3**	**4**	**5**
Hodnota:	3	8	7	0	1	9
Ukazatel k.:	4					

Tabulka 7 - Kruhový seznam: Zápis zbylých dat

Z tohoto je patrné, že data se dále zapisují jakoby od začátku s tím, že si pamatujeme, který index je nejstarší. Když ukazatel dojede na konec indexu, tedy bude roven 5, tak při další iteraci se mu nastaví automaticky hodnota 0 a začne se od začátku.

5.1.2.2 Třída DataStorage

Tato třída slouží k uchovávání dat pro jednotlivá čidla. Zde se data uchovávají pouze po dobu jejich platnosti. V okamžiku, kdy datům vyprší platnost, jsou nahrazena nově příchozími daty. Ukázka třídy:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    /// <summary>
    /// Zde budeme uchovávat data.
    /// </summary>
    public class DataStorage
    {
        int valueCount = 5 * 60;
        Dictionary<byte, SensorData> data = new Dictionary<byte, SensorData>();
        object globalLock = new object();

        public DataStorage(int valueCount)
        {
            this.valueCount = valueCount;
        }

        public void AddNewValue(byte sensorID, float temperature, DateTime time)
        {
            lock (globalLock)
            {
                if (!data.ContainsKey(sensorID))
                    data.Add(sensorID, new SensorData(valueCount, time));
                System.Diagnostics.Debug.WriteLine(string.Format("AddNewValue: {0} {1} {2}",
                                                    sensorID, time, temperature));
                data[sensorID].AddValue(temperature, time);
            }
        }

        public float[] DataToGraph(byte sensorID, DateTime queryTime)
        {
            lock (globalLock)
            {
                if (!data.ContainsKey(sensorID))
                {
                    var blankData = new float[valueCount];
                    for (int i = 0; i < blankData.Length; i++)
                        blankData[i] = float.NaN;
                    return blankData;
                }
                return data[sensorID].GetData(queryTime);
            }
        }
    }
```

Třída 12 - DataStorage (1. část)

```
    public class SensorData
    {
        public float[] Data { get; set; }
        public int EndPos { get; set; }
        public DateTime LastTime { get; set; }

        public SensorData(int valueCount, DateTime startTime)
        {
            Data = new float[valueCount];
            for (int i = 0; i < Data.Length; i++)
                Data[i] = float.NaN;
            EndPos = 0;
            LastTime = startTime;
        }

        public void AddValue(float value, DateTime time)
        {
            if (time == LastTime)
            {
                System.Diagnostics.Debug.WriteLine(string.Format("time == LastTime"));
                if (!float.IsNaN(value))
                    Data[EndPos] = value;
            }
            else
            {
                var skipSec = (int)Math.Round((time - LastTime).TotalSeconds - 1);
                System.Diagnostics.Debug.WriteLine(string.Format("skipSec {0}", skipSec));
                for (int i = 0; i < skipSec; i++)
                {
                    EndPos++;
                    if (EndPos == Data.Length)
                        EndPos = 0;
                    Data[EndPos] = float.NaN;
                }

                EndPos++;
                if (EndPos == Data.Length)
                    EndPos = 0;
                Data[EndPos] = value;
                LastTime = time;
            }
        }

        public float[] GetData(DateTime queryTime)
        {
            AddValue(float.NaN, queryTime);
            var result = new float[Data.Length];
            Array.Copy(Data, EndPos + 1, result, 0, Data.Length - EndPos - 1
            Array.Copy(Data, 0, result, Data.Length - EndPos - 1, EndPos + 1);
            return result;
        }
    }
}
```

Třída 13 - DataStorage (2. část)

5.1.2.2.1 Konstruktor třídy

Obstarává, aby se do proměnné **valueCount** dostal parametr zvaný naprosto stejně. Parametr obsahuje naměřené hodnoty, které zaslalo čidlo.

```
public DataStorage(int valueCount)
{
    this.valueCount = valueCount;
}
```

Metoda 7 - Konstruktor pro DataStorage

5.1.2.2.2 Metoda přidává hodnotu do kruhového seznamu

Metoda se jmenuje **AddNewValue**, což z anglického překladu znamená: přidej novou hodnotu. Tuto metodu voláme, když nám přijde nová zpráva z teplotního čidla.

```
public void AddNewValue(byte sensorID, float temperature, DateTime time)
{
    lock (globalLock)
    {
        if (!data.ContainsKey(sensorID))
            data.Add(sensorID, new SensorData(valueCount, time));
        System.Diagnostics.Debug.WriteLine(string.Format("AddNewValue: {0} {1} {2}",
                                                         sensorID,
                                                          time, temperature));
        data[sensorID].AddValue(temperature, time);
    }
}
```

Metoda 8 - Přidá hodnotu do kruhového seznamu

V metodě existuje zámek, který nám na okamžik zamkne **DataStorage**, protože k němu přistupuje program z více vláken. K přepsání dat by nemělo dojít nikdy, protože data z čidel dostáváme cca po 1 sekundě. A zápis do **DataStorage** trvá 2 tisíciny sekundy. Ale přesto je dobré vyvarovat se případným chybám a na okamžik to zamknout. Následuje podmínka, že pokud **Dictionary** neobsahuje senzor, který zaslal data, tak se tento senzor do slovníku přidá. Řádek začínající slovem „**System**“ je pouze pro testovací účely a nemá žádný vliv na výsledek programu. Nakonec zapíšeme data zaslaná senzorem do Dictionary (slovníku) s aktuálním časem.

5.1.2.2.3 Metoda pro vykreslování dat do grafu

Tato metoda se volá při vykreslování dat do grafu. Zde je znovu použit zámek.

```
public float[] DataToGraph(byte sensorID, DateTime queryTime)
{
    lock (globalLock)
    {
        if (!data.ContainsKey(sensorID))
        {
            // když vůbec nemáme senzor evidovaný (zatím od něj žádná data nepřišla)
            var blankData = new float[valueCount];
            for (int i = 0; i < blankData.Length; i++)
                blankData[i] = float.NaN;
            return blankData;
        }

        return data[sensorID].GetData(queryTime);
    }
}
```

Metoda 9 - Vykreslování dat do grafu

Podle komentáře je patrné, že pokud se připojí nový senzor, který ještě nebyl evidován, a proto od něj zatím nepřišla žádná data, je nejdříve zaevidován. Jinak program vrátí hodnoty dat, tzn. kdy byla data přijatá, ID senzoru a naměřenou hodnotu.

5.1.2.2.4 Vnořená třída SensorData – konstruktor třídy

Třída jako taková obstarává pouze kruhový seznam pro jedno konkrétní přihlášené čidlo. Takto vypadá konstruktor této třídy:

```
public SensorData(int valueCount, DateTime startTime)
{
    Data = new float[valueCount];
    for (int i = 0; i < Data.Length; i++)
        Data[i] = float.NaN;
    EndPos = 0;
    LastTime = startTime;
}
```

Metoda 10 - Konstruktor třídy SensorData

Konstruktor jako parametr dostane hodnotu, kterou čidlo naměřilo, a datum s časem, který je absolutní a měří se od začátku pole. Při vytvoření zapíšeme na všechny indexy v kruhovém seznamu hodnotu **NaN**, což je stav, který nemá žádnou hodnotu. A nastavíme si koncovou pozici na 0.

5.1.2.2.5 Přidání hodnoty do kruhového seznamu

Pro přidání hodnoty do kruhového seznamu slouží metoda AddValue, která z anglického překladu znamená: „přidej hodnotu“.

```
public void AddValue(float value, DateTime time)
{
    if (time == LastTime)
    {
        System.Diagnostics.Debug.WriteLine(string.Format("time == LastTime"));
                if (!float.IsNaN(value))
                        Data[EndPos] = value;
    }
    else
    {
        var skipSec = (int)Math.Round((time - LastTime).TotalSeconds - 1);
        System.Diagnostics.Debug.WriteLine(string.Format("skipSec {0}", skipSec));
        for (int i = 0; i < skipSec; i++)
        {
            EndPos++;
            if (EndPos == Data.Length)
                EndPos = 0;
            Data[EndPos] = float.NaN;
        }

        EndPos++;
        if (EndPos == Data.Length)
            EndPos = 0;
        Data[EndPos] = value;
        LastTime = time;
    }
}
```

Metoda 11 - Metoda přidávající data do kruhového seznamu

Vstupními parametry metody jsou hodnota a čas. Ze všeho nejdříve metoda porovná, zda zadaný čas je stejný jako poslední čas, pokud ano, pokusí se zjistit, zda hodnota je NaN, není-li, přiřadí na poslední pozici nová data.

Pokud není zadaný čas stejný jako poslední čas, metoda spočítá kolik je potřeba přeskočit sekund. To vykoná tak, že od času, který přišel v parametru, odečteme poslední čas. Poté v cyklu provede přeskočení na požadovanou hodnotu a tam nastaví NaN. Metoda testuje ještě další podmínku. Pokud se konečná pozice rovná délce kruhového seznamu, tak další pozice je zas na indexu nula.

Nakonec zapíšeme hodnotu, zvětšíme index poslední pozice o jedničku a uložíme si čas jako poslední čas. Znovu kontrolujeme, zda při zapisování nejsme již na konci, pokud ano, tak zapisujeme na začátek.

5.1.2.2.6 Metoda, která rozřízne data a slepí je ve správném pořadí

Tuto metodu volá metoda DataToGraph 3.1.3.2.3.

```
public float[] GetData(DateTime queryTime)
{
    AddValue(float.NaN, queryTime);
    var result = new float[Data.Length];
    Array.Copy(Data, EndPos + 1, result, 0, Data.Length - EndPos - 1);
    Array.Copy(Data, 0, result, Data.Length - EndPos - 1, EndPos + 1);
    return result;
}
```

Metoda 12 - Metoda pro uspořádání zobrazovaných dat

Metoda vezme konec kruhového seznamu a přilepí ho na začátek. Toto je důležité pro správné vykreslování. Udělá to tak, že rozřízne kruhový seznam mezi koncem a začátkem a slepí je opačně.

5.1.3 Třída Device

Třída Device reprezentuje zařízení. Je to mateřská třída, která obsahuje společné vlastnosti všech typů zařízení, které je možné připojit. Vlastnosti zařízení mohou být například: ID zařízení, úroveň nabití akumulátoru, barva přiřazená zařízení, poloha v prostoru.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace Core
{
    public abstract class Device
    {
        byte deviceID;
        Color color = Color.Black;
        Point location;
        int batteryLevelPercent;

        internal Device(byte deviceID)
        {
            this.deviceID = deviceID;
        }

        public byte DeviceID
        {
            get { return deviceID; }
        }

        public int BatteryLevelPercent
        {
            get { return batteryLevelPercent; }
            set { batteryLevelPercent = value; }
        }

        /// <summary>
        /// Barva přiřazená zařízení.
        /// </summary>
        public Color Color
        {
            get { return color; }
            set { color = value; }
        }

        /// <summary>
        /// Poloha v prostoru o velikosti (-100 až +100)x(-100 až +100).
        /// </summary>
        public Point Location
        {
            get { return location; }
            set { location = value; }
        }
    }
}
```

Třída 14 - Device

Tato třída obsahuje obecné vlastnosti pro připojené zařízení. Všechny tyto vlastnosti jsou nastavené jako veřejné, aby je bylo možné volat i z jiných míst. Vlastnosti:

- **DeviceID** – Identifikační ID zařízení
- **BetteryLevelPercent** – stav akumulátoru
- **Color** – reprezentující barva zařízení
- **Location** – umístění zařízení

5.1.4 Třída ThermometerDevice

Jedná se o dceřinou třídu od třídy Device 4.1.4. Tato třída je doplněna o další vlastnosti zařízení. Tentokrát je třída konstruována na jeden konkrétní typ zařízení a to teploměr.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    /// <summary>
    /// Teploměr.
    /// </summary>
    public class ThermometerDevice : Device
    {
        public ThermometerDevice(byte deviceID) : base(deviceID)
        {
        }

        public override string ToString()
        {
            return string.Format("Teploměr ID={0}", DeviceID);
        }
    }
}
```

Třída 15 - ThermometerDevice

Z mateřské třídy tato třída podědí deviceID, což je ID připojeného zařízeni. A následuje přepsání metody ToString(), na vlastní metodu, která bude vracet textový řetězec ve formátu: „Teploměr ID = [indetifikační číslo teploměru].“. Nařpíklad „Teploměr ID = 8“

5.1.5 Třída WindMeterDevice

Velmi obdobná třída jako třída ThermometerDevice 5.1.5, dědí z třídy Device 5.1.4. Rozdíl mezi touto třídou a třídou ThermometerDevice je takový, že tato třída je pro zařízení, které měří směr a rychlost větru.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    public class WindMeterDevice : Device
    {
        internal WindMeterDevice(byte deviceID) : base(deviceID)
        {
        }

        public override string ToString()
        {
            return string.Format("Větroměřák ID={0}", DeviceID);
        }
    }
}
```

Třída 16 - WindMeterDevice

Naprosto stejná struktura třídy jako u ThermometerDevice, s nepatrným rozdílem u metody ToString.

Na místo: „Teploměr ID = [číslo teploměru]“.

Vypisujeme „Větroměřák ID = [číslo teploměru]“.

5.1.6 Třída ServerStatus

Naprosto unikátní třída obsahující výčet, zda server běží nebo neběží.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Core
{
    public enum ServerStatus
    {
        Stopped,
        Running,
    }
}
```

Třída 17 - ServerStatus

Z anglických názvů je patrné, že slovíčko Stopped znamená, že server je zastaven a slovíčko Running znamená, že server běží. Tato třída se používá ve třídě Communication 5.1.1.1.

5.1.7 Třída MainManager

Tato třída je hlavní představitelkou celého jádra projektu, tedy části CORE. V této třídě probíhá veškeré sloučení tříd. Jinak řečeno, tato třída si volá ostatní třídy a pracuje s nimi podle potřeby.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Timers;
using System.Drawing;

namespace Core
{
    public class MainManager
    {
        static MainManager instance;
        public event EventHandler ChangeDeviceListEvent;

        public static MainManager Instance
        {
            get
            {
                if (instance == null) instance = new MainManager();
                return instance;
            }
```

Třída 18 - MainManager (1. část)

```
List<Device> connectedDevices = new List<Device>();
Communication comm = new Communication(9999);
DataStorage data = new DataStorage(5*60);
Random r = new Random();

private MainManager()
{
    comm.SensorConnected += new EventHandler<SensorConnectedEventArgs>
                            (comm_SensorConnected);
    comm.SensorDisconnected += new EventHandler<SensorDisconnectedEventArgs>
                               (comm_SensorDisconnected);
    comm.DataReceived += new EventHandler<DataReceivedEventArgs>(comm_DataReceived);
}

public void comm_DataReceived(object sender, DataReceivedEventArgs e)
{

    connectedDevices.Single(x => x.DeviceID ==
                                        e.Message.SensorID).BatteryLevelPercent =
                                        e.Message.BatteryInfo;

    DateTime dateTime = DateTime.Now;
    dateTime = new DateTime(dateTime.Year, dateTime.Month, dateTime.Day,
               dateTime.Hour, dateTime.Minute, dateTime.Second);

    if (e.Message is ThermometerDataMessage)
    {
        data.AddNewValue(e.Message.SensorID, (e.Message as ThermometerData
                         Message).Temperature, dateTime);
    }
}

public void comm_SensorConnected(object sender, SensorConnectedEventArgs e)
{
    connectedDevices.Add(new ThermometerDevice(e.SensorID) { Color =
                        Color.FromArgb(r.Next(255),r.Next(255),r.Next(255))});
    FireChangeDeviceListEvent();
}

private void FireChangeDeviceListEvent()
{
    if (ChangeDeviceListEvent != null)
         ChangeDeviceListEvent(this, new EventArgs());
}
public void comm_SensorDisconnected(object sender, SensorDisconnectedEventArgs e)
{
    connectedDevices.Remove(connectedDevices.Find(x => x.DeviceID == e.SensorID));
    FireChangeDeviceListEvent();
}
public Device[] GetConnectedDeviceList()
{
    return connectedDevices.ToArray();
    //throw new System.NotImplementedException();
}
public void ServerStart()
{
    comm.Start();
    MainManager.Instance.ServerStatus = Core.ServerStatus.Running;
}
public void ServerStop()
{
    comm.Stop();
    MainManager.Instance.ServerStatus = Core.ServerStatus.Stopped;
}
```

Třída 19 - MainManager (2. část)

```
        public float[] GetHistoryData(byte sensorID)
        {
            DateTime dateTime = DateTime.Now;
            dateTime = new DateTime(dateTime.Year, dateTime.Month, dateTime.Day,
                       dateTime.Hour, dateTime.Minute, dateTime.Second);

            return data.DataToGraph(sensorID, dateTime);
        }
    }
}
```

Třída 20 - MainManager (3. část)

5.1.7.1 Konstruktor třídy

V tomto konstruktoru třídy se zaregistrují 3 události, na které následně reagujeme.

```
private MainManager()
{
    comm.SensorConnected += new EventHan-
dler<SensorConnectedEventArgs>(comm_SensorConnected);
    comm.SensorDisconnected +=
                    new EventHandler<SensorDisconnectedEventArgs>(comm_SensorDisconnected);
    comm.DataReceived += new EventHandler<DataReceivedEventArgs>(comm_DataReceived);
}
```

Metoda 13 - Konstruktor třídy MainManager

5.1.7.2 Metoda FireChangeDeviceList

Úkolem této metody je provádět změnu v seznamu připojených zařízení. Připojí-li se nové zařízení, metoda aktualizuje, tzn., že do něj přidá nové zařízení. Součástí této metody je i událost, která je volána při připojení nového zařízení. Zobrazená metoda je včetně události.

```
public event EventHandler ChangeDeviceListEvent;

private void FireChangeDeviceListEvent()
{
    if (ChangeDeviceListEvent != null) // což není null, pokud se někdo na to zaregistroval
        ChangeDeviceListEvent(this, new EventArgs());
}
```

Metoda 14 - ChangeDeviceListEvent

První řádka kódu je veřejná událost zvaná **ChangeDeviceListEvent**. Zbylý kus kódu je soukromá metoda této třídy, která se dá volat pouze v této třídě. Tato metoda zjišťuje, zda událost **ChangeDeviceListEvent** je různá od null. Přesněji a lidsky řečeno,

testuje, zda nastala nějaká změna v seznamu zařízení. To se předpokládá, jinak by nebyla volána událost **ChandeDeviceListEvent**. V tom případě se událost přilásí

5.1.7.3 Metoda comm_SensorConnected

Tato metoda je veřejná a zavolá se v případě připojení zařízení. Při úspěšném připojení je zařízení přidáno na seznam zařízení a zavolána událost ChangeDeviceListEvent, která upozorňuje, že nastala změna.

```
public void comm_SensorConnected(object sender, SensorConnectedEventArgs e)
{
    connectedDevices.Add(new ThermometerDevice(e.SensorID) { Color =
                        Color.FromArgb(r.Next(255),r.Next(255),r.Next(255))});

    FireChangeDeviceListEvent();
}
```

Metoda 15 - comm_SensorConnected

Při připojení zařízení se do seznamu zařízení přidá ID senzoru a náhodně se mu vybere nějaká barva pomocí RGB. R = Red (červená), G = Green (zelená), B = Blue (modrá). Různými hodnotami těchto tří barev je program schopen namíchat téměř jakoukoli barvu. Barvy se míchají tak, že se jim nastavuje hodnota od 0 do 255. Malá ukázka barev:

R	G	B	Výsledná barva
20	132	43	#14842B
197	86	202	#C556CA
112	186	175	#7ABAAF
178	57	50	#B23932
0	0	0	#000000
255	255	255	#FFFFFF
255	255	0	#FFFF00
255	0	255	#FF00FF

Tabulka 8 - Ukázka RGB barev

Program mohl být naprogramován tak, aby si uživatel při každém připojeném čidlu vybral, jakou barvu u něj chce mít, avšak toto by bylo z profesionálního hlediska

nevhodné. Pro uživatele by měl být program co nejsnadnější a nejpřehlednější. A na základě tohoto nepsaného pravidla je postavena i logika tohoto programu. Program vybere uživateli barvu sám z 8 dobře rozlišitelných barev. Uživatel připojí čidlo a ihned s ním může pracovat, nemusí se starat o nastavování programu, program vše dělá za uživatele. Od toho programy jsou, usnadňují nám práci.

5.1.7.4 Metoda comm_SensorDisconnected

Veřejná metoda, která se zavolá při odpojení čidla. Při zavolání metoda projde seznam čidel, najde odpojené a smaže jej. Následně zavolá událost ChangeDeviceListEvent, která aktualizuje seznam čidel v GUI.

```
public void comm_SensorDisconnected(object sender, SensorDisconnectedEventArgs e)
{
    connectedDevices.Remove(connectedDevices.Find(x => x.DeviceID == e.SensorID));
    FireChangeDeviceListEvent();
}
```

Metoda 16 - comm_SensorDisconnected

5.1.7.5 Metoda comm_DataReceived

Tato metoda zpracovává příchozí zprávy = aktualizuje stav baterie čidla a data zaslaná čidlem. V případě teplotního čidla je to teplota, v případě větrného čidla je síla a směr větru.

```
public void comm_DataReceived(object sender, DataReceivedEventArgs e)
{
    connectedDevices.Single(x => x.DeviceID == e.Message.SensorID).BatteryLevelPercent =
                                                                    e.Message.BatteryInfo;
    DateTime dateTime = DateTime.Now;
    dateTime = new DateTime(dateTime.Year, dateTime.Month, dateTime.Day, dateTime.Hour,
                            dateTime.Minute, dateTime.Second);
    if (e.Message is ThermometerDataMessage)
    {
        data.AddNewValue(e.Message.SensorID, (e.Message as
                                              ThermometerDataMessage).Temperature, dateTime);
    }

    if (e.Message is WindMeterDataMessage)
    {
    }
}
```

Metoda 17 - comm_DataReceived

Nejprve se aktualizuje stav baterie na novou hodnotu, kterou nám zaslalo čidlo. Následně se uloží datum a čas, kdy čidlo zaslalo data, respektive, kdy byla data přijata.

A následně se rozlišuje, od jakého typu čidla byla data přijata. Pokud byla přijata od teploměru, zavolá se metoda **AddNewValue** 5.1.3.2.2, kam se zadají potřebné parametry, jako ID senzoru, teplota a datum s časem. V případě snímače větru je to podobné.

5.1.7.6 Metoda ServerStart

Tato metoda se volá zvenčí. Metoda zajišťuje spuštění serveru a nastavení stavu na running.

```
public void ServerStart()
{
    comm.Start();
    MainManager.Instance.ServerStatus = Core.ServerStatus.Running;
}
```

Metoda 18 - ServerStart

5.1.7.7 Metoda ServerStop

Tato metoda je velmi podobná předchozí metodě, jen nespouští naslouchání, ale zastavuje již spuštěné naslouchání a nastavuje stav serveru na stopped.

```
public void ServerStop()
{
    comm.Stop();
    MainManager.Instance.ServerStatus = Core.ServerStatus.Stopped;
}
```

Metoda 19 - ServerStop

5.1.7.8 MetodaGetHistoryData

Metoda vrací data v předpřipraveném formátu do grafu s patřičným datem a časem.

```
public float[] GetHistoryData(byte sensorID)
{
    DateTime dateTime = DateTime.Now;
    dateTime = new DateTime(dateTime.Year, dateTime.Month, dateTime.Day, dateTime.Hour,
                            dateTime.Minute, dateTime.Second);

    return data.DataToGraph(sensorID, dateTime);
}
```

Metoda 20 - GetDataHistory

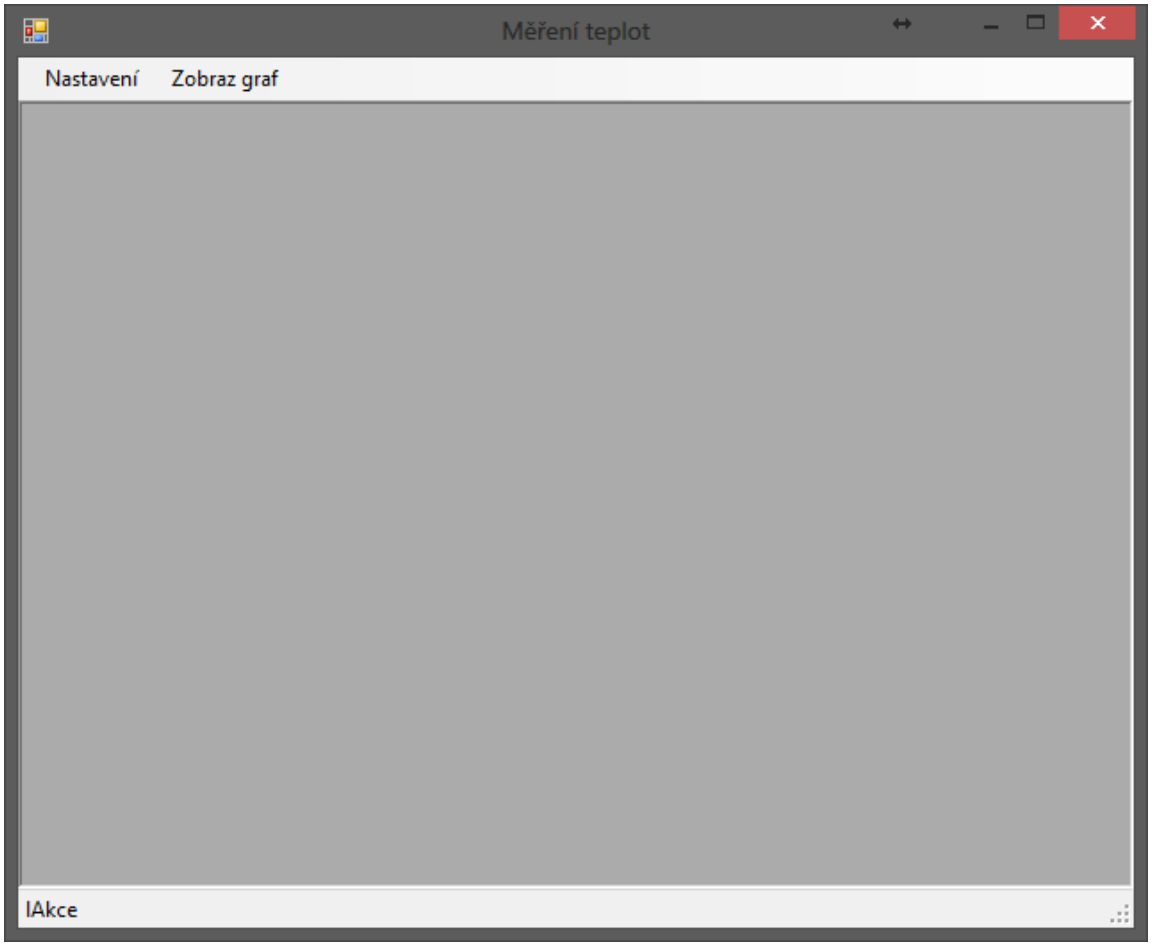
Nejprve je uložen čas do proměnné dateTime. Tato proměnná je posléze nastavena na požadovaný formát: rok, měsíc, den, hodina, minuta, vteřina. Následně se zavolá další metoda DataToGraph s parametry Id senzoru a datum čas.

5.2 GUI – grafické rozhraní projektu

Grafické rozhraní aplikace je navrženo ve stylu MDI, kde je hlavní okno bráno jako rodičovské a zbylá okna jsou umístěna v něm. Nemůže se tedy stát, aby některá okna byla mimo aplikaci. Díky tomu na taskBaru (windows liště) není otevřených mnoho oken, ale pouze jedno okno, které může mít i několik desítek vlastních oken.

5.2.1 Rodičovské okno (Parent Window)

Toto okno je tady hlavním nosníkem GUI. Okno obsahuje horní menu, kde je několik záložek, které otevírají jiná okna do tohoto rodičovského okna. Ve spodní části okna je takzvaný status panel, který vypisuje události v době, kdy nastanou.



Obrázek 8 - Rodičovské okno (Parent window)

Na přiloženém obrázku je vidět menu i status panel. Menu obsahuje 2 položky, první z nich je položka nastavení. Po kliknutí na tuto položku se otevře childWindow (dětské okno). V tomto okně se nastavuje program a rozmístění čidel. Nastavení programu obsahuje nastavení IP adresy na které má program naslouchat čidlům. Pro testovací účely je IP adresa nastavena na 127.0.0.1 = lokální IP adresa. Další položkou v horním memu je Zobraz graf. Kliknutím na tuto položku se zobrazí childWindow s informacemi o připojených zařízení a grafem. Takovýchto oken je možné otevřít neomezeně. Můžeme si například otevřít jedno okno pro 2 zařízení a další okno pro 1 zařízení.

5.2.1.1 Kód Rodičovského okna

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Core;
using System.Threading;

namespace AP
{
    public partial class fWorkingTable : Form
    {
        public fWorkingTable()
        {
            InitializeComponent();
        }

        fNastaveni formNastaveni = null;
        fGraf formGraf = null;

        private void nastaveníToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ZobrazNastaveni();
        }

        private void ZobrazNastaveni()
        {
            if (formNastaveni == null || formNastaveni.IsDisposed)
                formNastaveni = new fNastaveni();
            formNastaveni.MdiParent = this;
            formNastaveni.Show();
        }

        private void fWorkingTable_Load(object sender, EventArgs e)
        {
            MainManager.Instance.ServerStart();
        }
```

Třída 21 - Rodičovské okno (1. část)

```
private void fWorkingTable_FormClosing(object sender, FormClosingEventArgs e)
        {
            MainManager.Instance.ServerStop();
        }

        private void button1_Click(object sender, EventArgs e)
        {
        }

        private void zobrazGrafToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ZobrazGraf();
        }

        private void ZobrazGraf()
        {
            //if (formGraf == null || formGraf.IsDisposed)
                formGraf = new fGraf();
            formGraf.MdiParent = this;
            formGraf.Show();
        }
    }
}
```

Třída 22 - Rodičovské okno (2. část)

Při pohledu na kód je patrné, že se skládá z nějakých událostí. Například událost button1_Click je událost, která nastane po kliknutí na tlačítko.

5.2.1.1.1 Událost pro zobrazení okna nastavení

Tato metoda je v podstatě událost, která je vyvolána kliknutím na tlačítko: „**nastavení**“. Událost se jmenuje: „**nastaveníToolStripMenuItem_Click**“.

```
/// <summary>
/// Událost pro kliknutí na tlačítko nastavení
/// </summary>
/// <param name="sender">Odesílatel události - tlačítko Nastavení</param>
/// <param name="e">Událost</param>
private void nastaveníToolStripMenuItem_Click(object sender, EventArgs e)
{
    ZobrazNastaveni();
}
```

Metoda 21 - Událost pro zobrazení okna s nastavením

Zde voláme metodu: „**ZobrazNastaveni**“, která nemá žádné parametry.

5.2.1.1.2 Metoda ZobrazNastavení

Metoda, která se volá, když je potřeba zobrazit okno s nastavením. Program by mohl být vytvořen i bez této metody. Avšak z logického hlediska by se mělo okno s nastavením otevřít maximálně 1x. Po dobu otevření tohoto okna je zakázáno otevření

dalších oken s nastavením. Jaký je pro to důvod? Kdyby uživatel v prvním okně nastavení nastavil například IP adresu pro naslouchání na: „192.168.0.1“ a v druhém okně na „192.168.1.1“, vznikl by konflikt. Záleželo by, které okno bylo uložené jako poslední. Uložení nastavení probíhá kliknutím na tlačítko: „Uložit“. Kdyby bylo jako první uloženo nastavení prvního okna a poté druhého, výsledná IP adresa pro naslouchání čidlům by byla: „192.168.1.1“. Toto pravidlo platí i v obráceném pořadí. Tedy okno, které by bylo uloženo jako poslední by nastavovalo IP adresu. Tato metoda tedy blokuje otevření více oken s nastavením a zároveň okno otevírá. Metoda vypadá takto:

```
/// <summary>
/// Zobrazí nastavení, pokud již není zobrazeno
/// </summary>
private void ZobrazNastaveni()
{
    if (formNastaveni == null || formNastaveni.IsDisposed)
        formNastaveni = new fNastaveni();
    formNastaveni.MdiParent = this;
    formNastaveni.Show();
}
```

Metoda 22 - Metoda pro zobrazení okna nastavení

Jednoznačný kód metody nejprve testuje, zda již nějaké okno s nastavením není otevřené, pokud není, vytvoříme instanci okna, nastavíme mu rodičovské okno a pak jej zobrazíme. V případě, že okno existuje, zde není uveden žádný kód, protože není potřeba. Není potřeba na základě toho, že pokud není splněna podmínka, nevytvoří se instance okna a okno bez jeho instance není možné zobrazit.

5.2.1.1.3 Událost při načítání rodičovského okna

Tato událost nastane v okamžiku, kdy se rodičovské okno začne načítat. Tedy při spuštění programu. Událost se jmenuje: „**fWorkingTable_Load**“.

```
/// <summary>
/// Při načítání okna se nám spustí server
/// </summary>
/// <param name="sender">Odesílatel - okno</param>
/// <param name="e">Událost</param>
private void fWorkingTable_Load(object sender, EventArgs e)
{
    MainManager.Instance.ServerStart();
}
```

Metoda 23 - Událost při spuštění aplikace

Zde spustíme server. Tento jediný řádek spustí řetězovou reakci příkazů v předešlých třídách v Core části aplikace.

5.2.1.1.4 Událost při zavírání rodičovského okna

Tato událost nastane v okamžiku, kdy se rodičovské okno začne zavírat. Tedy uživatel klikne na křížek v pravém horním rohu aplikace nebo pomocí kombinace kláves **ALT+F4**. Událost se jmenuje: „**fWorkingTable_FormClosing**“.

```
/// <summary>
/// Při zavírání okna se nám zastaví server
/// </summary>
/// <param name="sender">Odesílatel - okno</param>
/// <param name="e">Událost</param>
private void fWorkingTable_FormClosing(object sender, FormClosingEventArgs e)
{
    MainManager.Instance.ServerStop();
}
```

Metoda 24 - Událost při zavírání aplikace

Obdobný případ jako u předchozí události. Zastavíme server. Tento řádek spustí řetěz jiných událostí v předchozích třídách v Core části aplikace.

5.2.1.1.5 Událost pro zobrazení okna s grafem

Tato metoda je v podstatě událost, která je vyvolána kliknutím na tlačítko: „**Zobraz graf**“. Událost se jmenuje: „**zobrazGrafToolStripMenuItem_Click**“.

```
/// <summary>
/// Zobrzí okno s grafem, lze zobrazit více oken
/// </summary>
/// <param name="sender">Odesílatel - tlačítko Zobraz graf</param>
/// <param name="e">Událost</param>
private void zobrazGrafToolStripMenuItem_Click(object sender, EventArgs e)
{
    ZobrazGraf();
}
```

Metoda 25 - Událost pro zobrazení okna s grafem

Zde se volá metoda: „**ZobrazGraf**“, která nemá žádné parametry.

5.2.1.1.6 Metoda ZobrazGraf

Tato metoda se volá v případě, když program otevírá okno s grafem. Vzhledem k situaci, že je žádoucí, aby bylo možné otevřít více nezávislých oken na sobě, tak se při každém kliknutí na tlačítko: „**Zobraz graf**“ se vytvoří nová instance tohoto okna. Tím

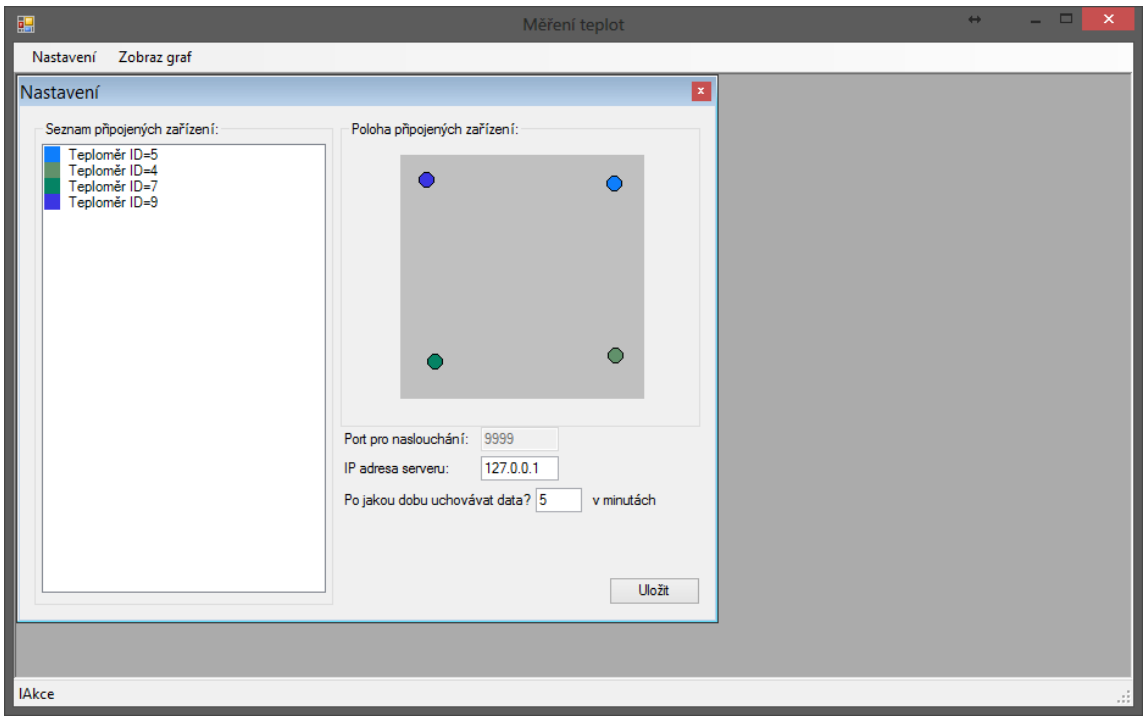
se zaručí, že každé okno může vykreslovat jiné grafy nezávisle na ostatních oknech. Například uživatel si bude chtít v prvním okně vykreslit graf pro čidla 1, 3, 5, 8 a v druhém okně pro čidla 2, 4, 6, 7 a ve třetím okně například čidla 1, 2, 3, 4. Všechno toto je možné. Protože okna nejsou na sobě nikterak závislá, tak hodnoty pro čidlo 1 v prvním okně budou naprosto stejné jako hodnoty v třetím okně pro totéž čidlo. To je dáno díky části **Core**, která nám zpracovává data ve vlastní třídě a vlákně.

```
/// <summary>
/// Zobrazí nový graf
/// </summary>
private void ZobrazGraf()
{
    //if (formGraf == null || formGraf.IsDisposed)
        formGraf = new fGraf();
    formGraf.MdiParent = this;
    formGraf.Show();
}
```

První řádek je zakomentován. Kdyby nebyl, šlo by vytvořit pouze jedno okno s grafem a to je v tomto případě nežádoucí.

5.2.2 Okno Nastavení

V tomto okně, které není možné zobrazit vícekrát v jednom okamžiku, se nastavuje IP adresa na které má program naslouchat čidlům. Defaultní nastavení **IP adresy** je **127.0.0.1**, které se také říká **lokální IP**. Tato adresa je tam nastavena pro testovací účely.



Obrázek 9 - Ukázka okna Nastavení

Na přiloženém obrázku je zobrazené okno nastavení se 4 připojenými čidly. V levé straně okna je seznam připojených zařízení. V pravé horní části je plocha, kde jsou kolečka reprezentující čidla. Každé kolečko má svoji příslušnou barvu, aby bylo zřetelné, jaké kolečko reprezentuje jaké čidlo. U koleček můžete měnit polohu podle skutečné polohy. Změna polohy se provádí kliknutím a držením levého tlačítka myši na kolečku a následným táhnutím do libovolné strany. Uvoleněním tlačítko kolečko umístíte. Ve spodní pravé části je defaultně nastaven port na **9999**, který je neměnný. Dále se zde nastavuje již zmíněná **IP adresa** a doba, po kterou chceme uchovávat data. V defaultním nastavení je doba nastavena na 5 minut, což je doba, kterou požadovali

zadavatelé projektu. Doba je zadávána v minutách. Tlačítkem „**Uložit**“ uložíme nastavení.

5.2.2.1 Kód okna Nastavení

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Core;

namespace AP
{
    /// <summary>
    /// Formulář nastavení.
    /// </summary>
    public partial class fNastaveni : Form
    {
        EventHandler changeDeviceListEventHandler;

        public fNastaveni()
        {
            InitializeComponent();
        }

        void fNastaveni_FormClosed(object sender, FormClosedEventArgs e)
        {
            MainManager.Instance.ChangeDeviceListEvent -= changeDeviceListEventHandler;
        }

        /// <summary>
        /// Pokud MainManager oznámí změnu seznamu připojených zařízení, tak obnovíme
        /// seznam...
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        void Instance_ChangeDeviceListEvent(object sender, EventArgs e)
        {
            this.Invoke((MethodInvoker)delegate()
            {
                UpdateConnectedDeviceList();
            });
        }


        private void UpdateConnectedDeviceList()
        {
            var devices = MainManager.Instance.GetConnectedDeviceList();
            lbConnectedDevices.Items.Clear();
            lbConnectedDevices.Items.AddRange(devices);
            sensorLocationEditor.Devices = devices;
        }
```

Třída 23 - Okno nastavení (1. část)

```
        private void fNastaveni_Load(object sender, EventArgs e)
        {
            changeDeviceListEventHandler = new EventHandler(Instance_ChangeDeviceListEvent);
            this.FormClosed += new FormClosedEventHandler(fNastaveni_FormClosed);
            MainManager.Instance.ChangeDeviceListEvent += changeDeviceListEventHandler;
            UpdateConnectedDeviceList();
        }

        private void lbConnectedDevices_DrawItem(object sender, DrawItemEventArgs e)
        {
            const int SPACE = 5;

            if (e.Index < 0)
                return;
            e.DrawBackground();
            Device device = (sender as ListBox).Items[e.Index] as Device;
            Brush myBrush = new SolidBrush(device.Color);
            e.Graphics.FillRectangle(myBrush, new Rectangle(e.Bounds.Left, e.Bounds.Top,
                                    e.Bounds.Height, e.Bounds.Height));
            e.Graphics.DrawString(((ListBox)sender).Items[e.Index].ToString(), e.Font,
                        Brushes.Black, new Rectangle(e.Bounds.Left + e.Bounds.Height +
                        SPACE, e.Bounds.Top, e.Bounds.Width - e.Bounds.Height - SPACE,
                        e.Bounds.Height), StringFormat.GenericDefault);
            e.DrawFocusRectangle();
        }

        private void btnSettingSaveSetting_Click(object sender, EventArgs e)
        {
            this.Visible = false;
        }
    }
}
```

Třída 24 - Okno nastavení (2. část)

V dalších částech této dokumentace budou rozebrány jednotlivé nejdůležitější metody.

5.2.2.1.1 Událost při zavírání okna Nastavení

```
void fNastaveni_FormClosed(object sender, FormClosedEventArgs e)
{
    // pri zavreni okna Nastaveni odregistrujeme obsluhu udalosti ChangeDeviceListEvent
    MainManager.Instance.ChangeDeviceListEvent -= changeDeviceListEventHandler;
}
```

Metoda 26 - Událost při zavírání okna Nastavení

V komentáři této události, která se jmenuje **fNastaveni_FormClosed**, je napsána funkce. Při zavírání tohoto okna se odregistruje obsluha události **ChangeDeviceListEvent**, protože již není potřeba měnit seznam v tomto okně.

5.2.2.1.2 Metoda, která naslouchá změnám v připojených zařízeních

Tato metoda naslouchá změnám v připojených zařízeních. Pokud třída **MainManager** 5.1.8 oznámí změnu v seznamu připojených čidel, zde je změna zachycena a následně provedena reakce. Metoda se jmenuje **Instance_ChangeDeviceListEvent**.

```
/// <summary>
/// Pokud MainManager oznámí změnu seznamu připojených zařízení, tak si stáhneme seznam
/// znovu...
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void Instance_ChangeDeviceListEvent(object sender, EventArgs e)
{
    this.Invoke((MethodInvoker)delegate()
    {
        UpdateConnectedDeviceList();
    });
}
```

Metoda 27 - Metoda, která naslouchá změnám v připojených zařízeních

Protože událost je vyvolána z jiného vlákna, než ve kterém bylo vytvořeno **GUI**, je třeba použít **Invoke** = předání volání do vlákna, ve kterém **GUI** (**this** neboli tento **form/okno**) bylo vytvořeno.

5.2.2.1.3 Metoda, která aktualizuje seznam připojených zařízení

Jak už je z nadpisu patrné, tato metoda aktualizuje seznam připojených zařízení. Tato metoda se jmenuje **UpdateConnectedDeviceList**.

```
/// <summary>
/// Aktualizuje seznam pripojenych zarizeni.
/// </summary>
private void UpdateConnectedDeviceList()
{
    var devices = MainManager.Instance.GetConnectedDeviceList();
    lbConnectedDevices.Items.Clear();
    lbConnectedDevices.Items.AddRange(devices);

    // nastavime vlastnost editoru se seznamem zarizeni
    sensorLocationEditor.Devices = devices;
}
```

Metoda 28 - Metoda, která aktualizuje seznam připojených zařízení

Při zavolání této metody se provedou následující úkony v tomto pořadí: nejprve se zavolá metoda **MainManager.Instance.GetConnectedDeviceList**, která vrátí nový seznam s připojenými zařízeními, následně se vyčistí seznam, který byl doposud zobra-

zen, a poté se do tohoto seznamu přidají nová zařízení. A nakonec se nastaví vlastnost editoru se seznamem zařízení.

5.2.2.1.4 Událost při načítání okna Nastavení

Tato událost je zavolána, když klikneme na tlačítko „Nastavení“. Po kliknutí na toto tlačítko se začne otevírat okno s nastavením. To je chvíle, kdy se zavolá tato událost, vykoná kód uvnitř a až poté se zobrazí okno s nastavením. Událost se jmenuje: „**fNastaveni_Load**“.

```
private void fNastaveni_Load(object sender, EventArgs e)
{
    changeDeviceListEventHandler = new EventHandler(Instance_ChangeDeviceListEvent);
    this.FormClosed += new FormClosedEventHandler(fNastaveni_FormClosed);
    MainManager.Instance.ChangeDeviceListEvent += changeDeviceListEventHandler;
    UpdateConnectedDeviceList();
}
```

Ještě předtím, než zobrazíme okno, zaregistrujeme se k událostem, které aktualizuje seznam zařízení. A nakonec je zažádáno o první seznam.

5.2.2.1.5 Událost vlastní vykreslení seznamu zařízení

Aby bylo možné mít v seznamu zařízení zobrazenou kostičku s příslušnou barvou, je potřeba tento seznam ručně vykreslit. Ručně vykreslit je slangový výraz pro událost, které se jmenuje **DrawItem** neboli kreslení předmětu..

```
private void lbConnectedDevices_DrawItem(object sender, DrawItemEventArgs e)
{
    const int SPACE = 5;
    if (e.Index < 0)
        return;
    e.DrawBackground();
    Device device = (sender as ListBox).Items[e.Index] as Device;
    Brush myBrush = new SolidBrush(device.Color);
    e.Graphics.FillRectangle(myBrush, new Rectangle(e.Bounds.Left, e.Bounds.Top,
                e.Bounds.Height, e.Bounds.Height));
    e.Graphics.DrawString(((ListBox)sender).Items[e.Index].ToString(), e.Font,
                Brushes.Black, new Rectangle(e.Bounds.Left + e.Bounds.Height + SPACE,
                e.Bounds.Top, e.Bounds.Width - e.Bounds.Height - SPACE, e.Bounds.Height),
                StringFormat.GenericDefault);
    e.DrawFocusRectangle();
}
```

Metoda 29 - Událost vlastní vykreslení seznamu zařízení

Stručně řečeno se zde vykreslují prvky do seznamu tak, že se vykreslí kostička s příslušnou barvou reprezentující zařízení, poté pevně stanovená mezera = 5 a až nakonec informace o zařízení.

5.2.2.1.6 Událost pro uložení nastavení.

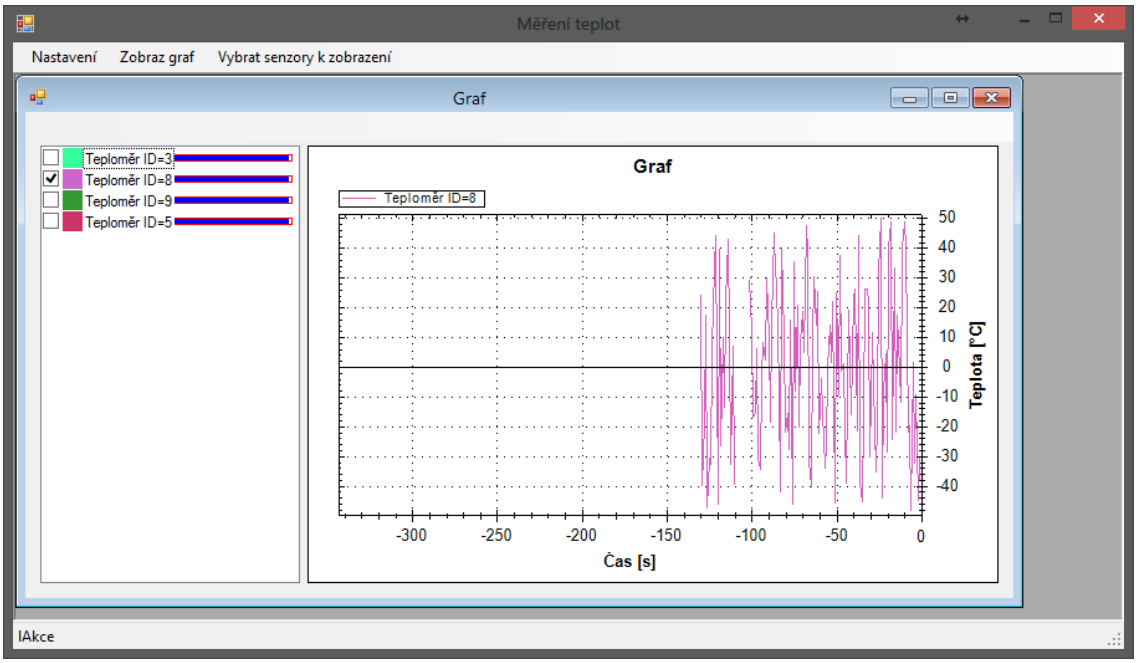
Tato událost je vyvolána při kliknutí na tlačítko „**Uložit**“, které se nachází v pravém dolním rohu okna **Nastavení**. V této fázi programu tlačítko nedělá nic jiného, než že okno nastavení skryje. Kvůli plánovaným změnám je to takto dočasně uděláno.

```
private void btnSettingSaveSetting_Click(object sender, EventArgs e)
{
    this.Visible = false;
}
```

Metoda 30 - Událost pro uložení nastavení

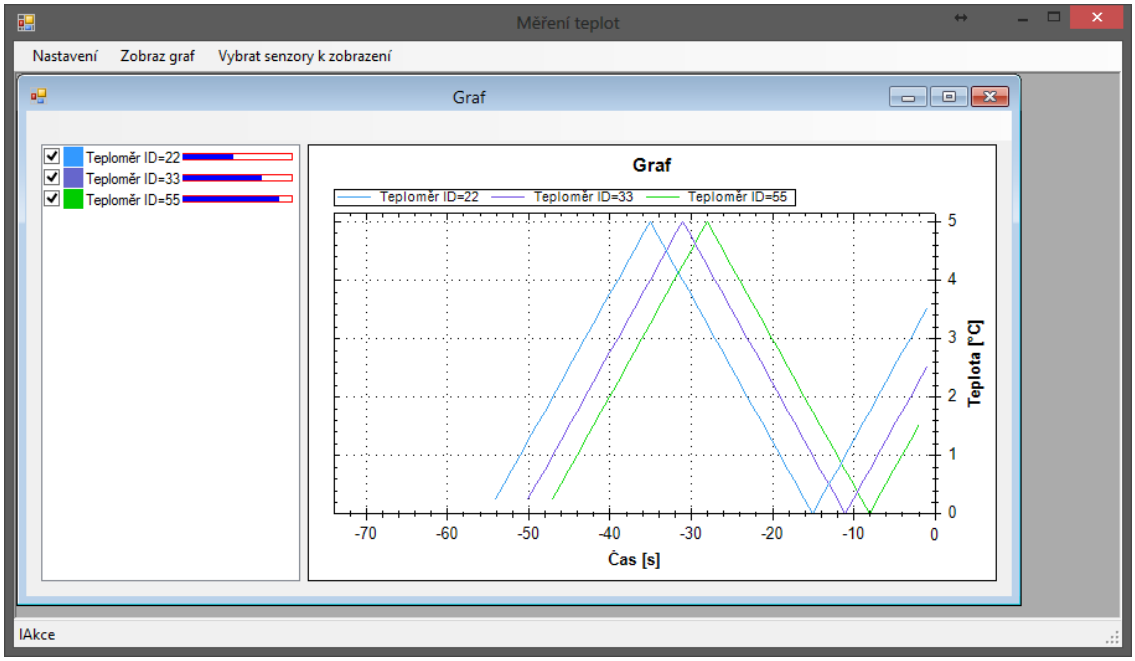
5.2.3 Okno Graf

Toto je cílové okno programu. V tomto okně vykreslujeme hodnoty jednotlivých čidel do grafu.



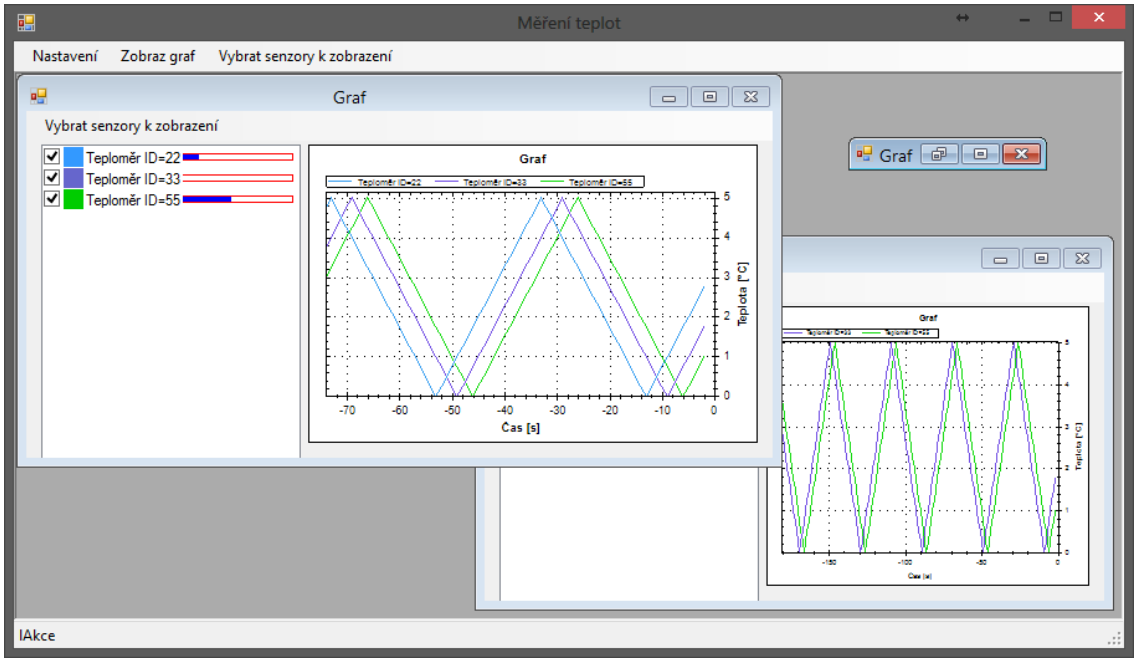
Obrázek 10 - Ukázka okna s grafem pro jedno čidlo

Na tomto obrázku je vyobrazen graf pro teplotní čidlo s ID = 8. V levé horní část je seznam zařízení, který obsahuje barvu zařízený, typ zařízení a stav baterie. V pravé části je graf vyobrazující hodnoty od času T0 do času T-300 a teploty v rozsahu -50°C až +50°C. Takto rozházený graf je díky testovacímu programu, který náhodně generuje teploty v daném rozsahu.



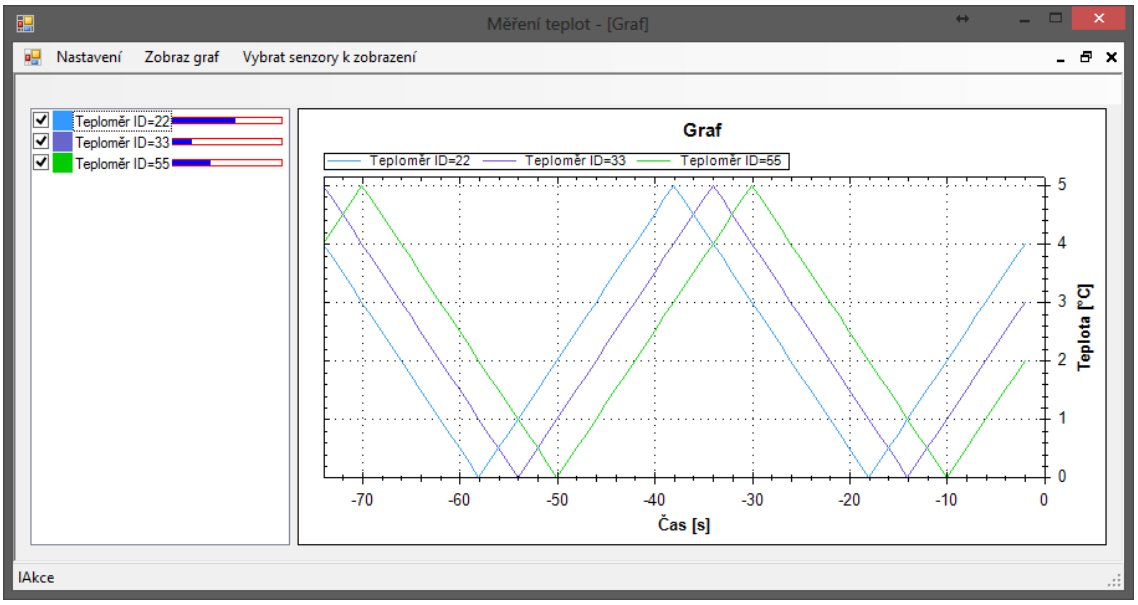
Obrázek 11 - Ukázka okna s grafem pro 3 čidla

Na obrázku jsou vykreslována do grafu průběhy teploty od tří čidel. Simulační program tentokrát simulovat pomalu přirůstající a klesající pilu, která nabývá hodnot od 0 do +5°C.



Obrázek 12 - Ukázka více oken s grafy

Na tomto obrázku je znázorněno, že je možné mít v aplikaci otevřeno více oken s grafy. Levé horní okno je vyobrazeno na předchozím obrázku, pravé dolní okno je nové zobrazující pouze 2 čidla ze tří připojených a třetí okno, nacházející se nad pravým oknem, je minimalizované.



Obrázek 13 - Ukázka okna grafu přes celou obrazovku

Zde je znázorněno, že je možné i roztáhnout okno grafu přes celou obrazovku.

5.2.3.1 Kód okna Graf

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Core;
using ZedGraph;

namespace AP
{
    public partial class fGraf : Form
    {
        EventHandler changeDeviceListEventHandler;

        public fGraf()
        {
            InitializeComponent();
            ListViewHelper.EnableDoubleBuffer(lvConnectedDevices);
```

Třída 25 - Okno graf (1. část)

```
            lvConnectedDevices.Items.Clear();
            timer1.Enabled = true;
            myPane = ygc.GraphPane;
            myPane.Title.Text = "Graf";
            myPane.YAxis.IsVisible = false;
            myPane.Y2Axis.IsVisible = true;
            myPane.XAxis.Title.Text = "Čas [s]";
            myPane.Y2Axis.Title.Text = "Teplota [°C]";
            myPane.XAxis.MajorGrid.IsVisible = true;
            myPane.Y2Axis.MajorGrid.IsVisible = true;
        }

        GraphPane myPane;

        private void timer1_Tick(object sender, EventArgs e)
        {

            myPane.CurveList.Clear();

            foreach (ListViewItem item in lvConnectedDevices.CheckedItems)
            {
                Device device = item.Tag as Device;
                var data = MainManager.Instance.GetHistoryData(device.DeviceID);

                PointPairList list = new PointPairList();
                for (int i = 0; i < data.Length; i++)
                {
                    double x = i - data.Length;
                    double y = data[i];
                    list.Add(x, y);
                }

                LineItem myCurve = myPane.AddCurve(device.ToString(), list, device.Color,
                                   SymbolType.None);
            }

            ygc.AxisChange();
            ygc.Refresh();
            lvConnectedDevices.Invalidate();
        }

        private void fGraf_Load(object sender, EventArgs e)
        {
            changeDeviceListEventHandler = new EventHandler(Instance_ChangeDeviceListEvent);
            MainManager.Instance.ChangeDeviceListEvent += changeDeviceListEventHandler;
            UpdateConnectedDeviceList();
        }

        private void fGraf_FormClosed(object sender, FormClosedEventArgs e)
        {
            MainManager.Instance.ChangeDeviceListEvent -= changeDeviceListEventHandler;
        }

        /// <summary>
        /// Pokud MainManager oznámí změnu seznamu připojených zařízení, aktualizuje seznam
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        void Instance_ChangeDeviceListEvent(object sender, EventArgs e)
        {
            this.Invoke((MethodInvoker)delegate()
            {
                UpdateConnectedDeviceList();
            });
        }
```

Třída 26 - Okno Graf (2. část)

```
        /// <summary>
        /// Aktualizuje seznam pripojenych zarizeni.
        /// </summary>
        private void UpdateConnectedDeviceList()
        {
            //imageList1.Images.Clear();

            var devices = MainManager.Instance.GetConnectedDeviceList();

            // zarizeni jiz zobrazena, resp. jejich ID, ktera se v seznamu zachovaji
            HashSet<byte> visibleDevices = new HashSet<byte>();

            // polozky seznamu k odstraneni (odpojena zarizeni)
            List<ListViewItem> itemsToRemove = new List<ListViewItem>();

            // projdeme soucasny seznam zobrazenych zarizeni a rozhodneme, ktere zarizeni se
            // zachova a ktere je treba smazat
            foreach (ListViewItem item in lvConnectedDevices.Items)
            {
                byte deviceID = (item.Tag as Device).DeviceID;
                if (devices.Any(x => x.DeviceID == deviceID))
                    visibleDevices.Add(deviceID); // zachovame
                else
                    itemsToRemove.Add(item); // smazat
            }

            // seznam pri iteraci)
            foreach (ListViewItem item in itemsToRemove)
            {
                imageList1.Images.RemoveByKey((item.Tag as Device).DeviceID.ToString());
                lvConnectedDevices.Items.Remove(item);
            }

            foreach (Device d in devices)
            {
                // pokud jiz zarizeni v seznamu je, tak jej nebudeme pridavat
                if (visibleDevices.Contains(d.DeviceID))
                    continue;


                ListViewItem item = new ListViewItem();
                item.Text = d.ToString();
                item.Tag = d;

                // vytvoreni barevne ikonky
                Bitmap b = new Bitmap(imageList1.ImageSize.Width,
                                      imageList1.ImageSize.Height);
                using (var g = Graphics.FromImage(b))
                {
                    g.FillRectangle(new SolidBrush(d.Color), new Rectangle(0, 0,
                                    imageList1.ImageSize.Width, image-
                                List1.ImageSize.Height));
                }
                imageList1.Images.Add(d.DeviceID.ToString(), b);
                item.ImageIndex = imageList1.Images.Count - 1;

                lvConnectedDevices.Items.Add(item);
            }
        }
```

Třída 27 - Okno graf (3. část)

```
        private void lvConnectedDevices_DrawItem(object sender, DrawListViewItemEventArgs e)
        {
            try
            {
                e.DrawBackground();
                e.DrawDefault = true;

                double percent = (e.Item.Tag as Device).BatteryLevelPercent / 100d;
                int leftMargin = 100;
                int topMargin = 5;
                int width = 100;
                int height = 5;
                e.Graphics.DrawRectangle(Pens.Red, new Rectangle(e.Bounds.X + leftMargin,
                                         e.Bounds.Y + topMargin, width, height));
                e.Graphics.FillRectangle(Brushes.Blue, new Rectangle(e.Bounds.X +
                        leftMargin + 1, e.Bounds.Y + topMargin + 1, (int)(width * percent) -
                        1, height - 1));
            }
            catch
            {
                // nothing
            }
        }
    }
}
```

Třída 28 - Okno graf (4. část)

5.2.3.1.1 Konstruktor okna graf

Zde se uvádí okno „**graf**“ do úvodního stavu. Například se popíší osy u grafu, rozmístí se komponenty a podobně.

```
public fGraf()
{
    InitializeComponent();
    ListViewHelper.EnableDoubleBuffer(lvConnectedDevices);
    lvConnectedDevices.Items.Clear();
    timer1.Enabled = true;

    myPane = ygc.GraphPane;
    //myPane.XAxis.Type = AxisType.Date;
    myPane.Title.Text = "Graf";
    myPane.YAxis.IsVisible = false;
    myPane.Y2Axis.IsVisible = true;
    myPane.XAxis.Title.Text = "Čas [s]";
    myPane.Y2Axis.Title.Text = "Teplota [°C]";
    myPane.XAxis.MajorGrid.IsVisible = true;
    myPane.Y2Axis.MajorGrid.IsVisible = true;
}
```

Metoda 31 - Konstruktor okna Graf

V této krátké metodě se inicializují komponenty, které jsou na okně rozmístěné, zavolá se třída, která umožňuje komponentě ListView dvojité bufferování. Tato třída se volá proto, aby při změně seznamu komponenta neblikala. Tuto třídu bylo nutné kom-

pletně celou navrhnout. Po zavolání zmíněné třídy se vyčistí seznam a spustí se timer (časovač). Zbývající řádky nastavují osy a popisky ke grafu.

5.2.3.1.2 Událost timer1_tick

Tato událost je vyvolána komponentou Timer (Časovač), která nedělá nic jiného, než, že v pravidelných intervalech, vykonává logiku uvnitř události.

```
private void timer1_Tick(object sender, EventArgs e)
{

    myPane.CurveList.Clear();

    foreach (ListViewItem item in lvConnectedDevices.CheckedItems)
    {
        Device device = item.Tag as Device;
        var data = MainManager.Instance.GetHistoryData(device.DeviceID);

        PointPairList list = new PointPairList();
        for (int i = 0; i < data.Length; i++)
        {
            double x = i - data.Length;
            double y = data[i];
            list.Add(x, y);
        }

        LineItem myCurve = myPane.AddCurve(device.ToString(), list, device.Color,
                           SymbolType.None);
    }

    ygc.AxisChange();
    ygc.Refresh();
    lvConnectedDevices.Invalidate();
}
```

Metoda 32 - Událost timer1_Tick

Tento kód popisuje, že s každým tikem timeru se překreslí hodnoty na grafu pro všechna zaháčkovaná zařízení ze seznamu.

5.2.3.1.3 Událost při načítání okna Graf

Tato událost je vyvolaná v okamžiku kliknutí na tlačítko „**Zobraz graf**“. V této události se provádí podobně jako u konstruktoru okna, inicializace okna. To znamená uvedení některých částí do výchozího stavu.

```
private void fGraf_Load(object sender, EventArgs e)
{
    changeDeviceListEventHandler = new EventHandler(Instance_ChangeDeviceListEvent);
    MainManager.Instance.ChangeDeviceListEvent += changeDeviceListEventHandler;
    UpdateConnectedDeviceList();
}
```

Metoda 33 - Událost při načítání okna Graf

Zde si zaregistrujeme jen události, které budeme potřebovat pro změnu připojených zařízení.

5.2.3.1.4 Událost při zavírání okna Graf

Tato událost se vyvolá při zavírání okna. Vykonává pouze odregistrování obsluhy od události, která nám hlásí změnu seznamu čidel.

```
private void fGraf_FormClosed(object sender, FormClosedEventArgs e)
{
    // pri zavreni okna Graf odregistrujeme obsluhu udalosti ChangeDeviceListEvent
    MainManager.Instance.ChangeDeviceListEvent -= changeDeviceListEventHandler;
}
```

Metoda 34 - Událost při zavírání okna Graf

5.2.3.1.5 Metoda, která naslouchá změnám v připojených zařízeních

Tato metoda identická s metodou 5.2.2.1.2. Proto není podstatní ji zde blíže popisovat.

```
/// <summary>
/// Pokud MainManager oznámí změnu seznamu připojených zařízení, stáhneme seznam znovu...
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void Instance_ChangeDeviceListEvent(object sender, EventArgs e)
{
    this.Invoke((MethodInvoker)delegate()
    {
        UpdateConnectedDeviceList();
    });
}
```

Metoda 35 - Metoda naslouchá změnám v seznamu připojených zařízení

5.2.3.1.6 Metoda aktualizuje seznam s připojenými zařízeními

Tato metoda je volána z předházející metody 5.2.3.1.5. Metoda aktualizuje seznam připojených zaříení v okně Graf.

```
/// <summary>
/// Aktualizuje seznam pripojenych zarizeni.
/// </summary>
private void UpdateConnectedDeviceList()
{
    var devices = MainManager.Instance.GetConnectedDeviceList();
    // zarizeni jiz zobrazena, resp. jejich ID, ktera se v seznamu zachovaji
    HashSet<byte> visibleDevices = new HashSet<byte>();

    // polozky seznamu k odstraneni (odpojena zarizeni)
    List<ListViewItem> itemsToRemove = new List<ListViewItem>();

    // projdeme soucasny seznam zobrazených zařízení a rozhodneme, které zarizení se zachová
    // a která je třeba smazat
    foreach (ListViewItem item in lvConnectedDevices.Items)
    {
        byte deviceID = (item.Tag as Device).DeviceID;
        if (devices.Any(x => x.DeviceID == deviceID))
            visibleDevices.Add(deviceID); // zachovame
        else
            itemsToRemove.Add(item); // smazat
    }
// mazeme, ale nesmime mazat "pod rukou" pri iteraci seznamu(modifikovat seznam pri iteraci)
    foreach (ListViewItem item in itemsToRemove)
    {
        imageList1.Images.RemoveByKey((item.Tag as Device).DeviceID.ToString());
        lvConnectedDevices.Items.Remove(item);
    }

    foreach (Device d in devices)
    {
        // pokud jiz zarizeni v seznamu je, tak jej nebudeme pridavat
        if (visibleDevices.Contains(d.DeviceID))
            continue;


        ListViewItem item = new ListViewItem();
        item.Text = d.ToString();
        item.Tag = d;

        // vytvoreni barevne ikonky
        Bitmap b = new Bitmap(imageList1.ImageSize.Width, imageList1.ImageSize.Height);
        using (var g = Graphics.FromImage(b))
        {
            g.FillRectangle(new SolidBrush(d.Color), new Rectangle(0, 0,
                            imageList1.ImageSize.Width, imageList1.ImageSize.Height));
        }
        imageList1.Images.Add(d.DeviceID.ToString(), b);
        item.ImageIndex = imageList1.Images.Count - 1;

        lvConnectedDevices.Items.Add(item);
    }
}
```

Metoda 36 - Metoda aktualizuje seznam s připojenými zařízeními pro Graf

Tato metoda je klíčová pro správné zobrazování grafů. Bez této metody by se neodstraňovala odpojená čidla a nebylo by možné zobrazovat v grafu nově připojená čidla.

V první části kódu této metody je nejprve zažádáno o nový seznam zařízení. Následně je zachován seznam, který je v současné době zobrazený a pak je vytvořen seznam zařízení, která se odpojila. V dalším kroku se prochází seznam zařízení, odstraňují se v z něho odpojená zařízení a aktualizovaný seznam se zobrazí.

Ve druhé části kódu této metody jsou odtraňována odpojedná zařízení podle klíče z obrázku, kam si ukládáme polohu zařízení. Následně jsou do seznamu přidáváná všechna připojená zařízení. Pokud se již v seznamu připojené zařízení nachází, je vynecháno. Pokud se v seznamu zařízení nenachází, je přídáno i se stavem baterie.

5.2.3.1.7 Událost vlatní vykreslení seznamu zařízení

Tato událost je posledním krokem pro vykreslení zařízení. Pomocí této události se stane seznam se zařízeními vykreselný na ListView. Rozdíl mezi touto událostí a událostí 5.2.2.1.5 je v tom, že zde vykreslujeme i stav baterie.

```
private void lvConnectedDevices_DrawItem(object sender, DrawListViewItemEventArgs e)
{
    try
    {
        e.DrawBackground();
        e.DrawDefault = true;
        double percent = (e.Item.Tag as Device).BatteryLevelPercent / 100d;
        int leftMargin = 100;
        int topMargin = 5;
        int width = 100;
        int height = 5;
        e.Graphics.DrawRectangle(Pens.Red, new Rectangle(e.Bounds.X + leftMargin, e.Bounds.Y
                                 + topMargin, width, height));
        e.Graphics.FillRectangle(Brushes.Blue, new Rectangle(e.Bounds.X + leftMargin + 1,
                                 e.Bounds.Y + topMargin + 1, (int)(width * percent) - 1,
                                 height - 1));
    }
    catch
    {
    }
```

Metoda 37 - Událost vlastní vykreslení seznamu zařízení

Celá logika této události je zasazena do TryCatch bloku. To znamená, že se pokusí vykreslit obsah bloku try. Pokud se to neprovede, zahlasí program výjimku a v tomto případě se nic bezobrazí.

6 Zhodnocení práce

V období psaní této práce jsem byl nucen k mnoha kompromisům, které mají různorodý dopad na výsledek práce. Některé kompromisy mají pozitivní dopad na vylepšení programu a jiné kompromisy negativní dopad. Díky mým zkušenostem s komerčním vývojem SW pro velké firmy jako je Smartdata s.r.o., ŠKODA AUTO a.s., mi zabral vývoj programu cca 10 hodin. Na druhou stranu zde platí pravidlo: „80% práce uděláš za 20% času, zbývajících 20% práce děláš 80% času“. Psaní dokumentace zabralo přibližně 40 hodin. Důležité je, že hlavním cílem této práce není dokumentace, ale výsledný program.

Program je možné vlepšit mnoha způsoby, například dalším rozdělením oken, kde seznam zařízení bude ve vlastním okně a graf také. Takto jsou oba prvky ve společném okně a na malých displejích to nemusí být dobře viditelné. Další aktualizací programu by mohlo být přehazování barev z čidla na čidlo. Takto se barvy čidlům přiřazují již ihned po připojení a jsou neměnné, inovací programu by mohla být možnost, kliknou na barevný čtvereček u čidla a vybrat si libovolnou barvu. Drobný vtípek na závěr pro odlehčení situace:

V průběhu psaní této práce jsem snědl pět dobře živených čuníků a vypil 15 litrů vody. Jinak práce mě vcelku bavila. =oD

7 Literatura a informační zdroje

JAY GLYNN, KARLI WATSON a KOL. (2003), *Programujeme profesionálně*. EAN: 9788025100851.

TREY NASH, (2010), *C#2010*. EAN: 9788025130346

8 Klíčová slova a vybrané pojmy

- **Jazyk C#** - je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework.
- **Protokol TCP** - je jedním ze základních protokolů sady protokolů Internetu, konkrétně představuje transportní vrstvu.
- **Software** – je v informatice program používaný v počítači a vykonávající nějakou činnost.
- **Wi-Fi** – bezdrátová komunikace mezi počítači nebo zařízeními.
- **Core** – z anglického slovíčka jádro, je srdcem aplikace, zde se odehrává veškerá logika programu.
- **GUI** – z angličtiny (Graphic User Interface) je grafické uživatelské rozhraní, tedy to, co uživatel vidí na obrazovce počítače.
- **IP adresa -** je v informatice číslo, které jednoznačně identifikuje síťové rozhraní v počítačové síti, která používá IP (internetový protokol)
- **IDE** – je integrované grafické prostředí (Integrated Development Environment)
- **VS** – Visula Studio
- **Dictionary (slovník)** – Je objekt uchovávající si data podle nějakého klíče
- **MDI** – umožňuje zobrazit více oken současně.

9 Seznam obrázků

10 Seznam tříd

11 Seznam metod

12 Seznam tabulek

13 Přílohy

13.1 Instalační příručka

Instalace tohoto software je velmi snadná. Jak tedy **SW** nainstalovat? Zkopírováním složky **TermometerViewer**, která se se nachází na přiloženém **CD**, do svého počítače provede kompletní instalaci. Složka obsahuje:

- **TermometerViewer.exe**
- **Core.dll**
- **ZedGraph.dll**

Je nutné, aby složka v cílovém počítači obsahovala všechny 3 soubory v jedné složce. V případě ztráty jednoho souboru nebude aplikace funkční.