Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Zobrazovací systém pro výukové účely

Bakalářská práce

Lukáš Širhal

vedoucí práce: Ing. Václav Novák, Csc.

České Budějovice 2012

Bibliografické údaje

Širhal L.,2012: Zobrazovací systém pro výukové účely.[Bc. Thesis, in Czech.] - 30 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Abstract:

This work deals with presentation and work with the measurement values. The main objective of this work was to design an imaging system for PC (Windows or Mac), which will display the measured values in real time. The created application allows you to design your own charts in spreading.

Abstrakt:

Tato práce se zabývá zobrazením a prácí s měřením hodnot. Hlavním cílem práce bylo navrhnout zobrazovací systém pro PC (OS Windows nebo Mac), který bude zobrazovat měřené hodnoty v reálném čase. Vytvořená aplikace dovoluje navrhnout si vlastní rozprostření grafů v aplikaci.

Keywords:

Graf, real-time, element, drag, stretch

Klíčová slova:

Graf, real-time, element, drag, stretch

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě, elektronickou cestou ve veřejně přístupné části databáze STAG, provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů..

V Českých Budějovicích, dne 12.12. 2012

Podpis: .....

Poděkování: Děkuji panu Ing. Václavu Novákovi, Csc. za odborné vedení při zpracování samotného projektu i při vypracování odborného textu.

Obsah

1 Úvod **7**

2 Cíle práce **8**

3 Základní principy systémů reálného času **9**

4 Real-time aplikace **9**

4.1 Real-time systémy se dělí na Hard a Soft systémy 11

4.1.1 Hard 11

4.1.2 Soft 11

5 Real-time OS **11**

5.1 Windows RTX 11

5.2 RTLinux 11

5.3 PikeOS 12

5.4 VxWorks 12

6 Real-time přístup k databázi a datům **12**

6.1 Charakteristika RTBDS 13

7 Windows Presentation Foundation **14**

7.1 Úvod do WPF 14

7.2 Úvod do WPF 15

7.3 Stylizace WPF (jazyk XAML) 16

8 Volba knihovny pro vytváření grafů **17**

8.1 Nástroje použité při vytváření aplikace 18

9 Samotný návrh řešení **18**

9.1 Schéma funkčnosti projektu 18

9.2 UseCase diagram 19

9.3 Sequence diagram hlavní aplikace 20

9.4 Sequence diagram Editoru 21

9.5 Class diagram 22
9.6 MainWindow.cs 22
9.6.1 MakeChart.cs 24
9.6.2 MultiChart.cs 25
9.6.3 ReadF.cs 26
9.6.4 Number.cs a NowTm.cs 27
9.7 Editor.xaml.cs 28
9.7.1 Element.cs 32
9.7.2 ListRec.cs 33
9.7.3 Seznam.cs 33
9.7.4 SaveList.cs 34

10 Testování aplikace **35**

11 Závěr **36**

1 Úvod

Tato práce se zabývá zobrazováním naměřených hodnot, např. teploměrem, které se nacházejí v textovém souboru uloženém na lokálním disku. Tato data výsledná aplikace načítá a vykresluje pomocí lineárních grafů nebo jako konstantní hodnoty v textovém okénku. Data jsou na disk zapisována po řádcích do textového souboru a aplikace si při každé změně souboru tyto řádky přečte a začne postupně vypisovat na obrazovku.

Součástí této práce je také Editor, ve kterém si rozvrhneme umístění grafů po pracovní ploše. Grafům můžeme měnit také některé parametry: polohu, rozměr, barvu. V Editoru můžeme také popsat názvy měřených konstantních hodnot. Návrhy vytvořené v editoru si ukládáme v podobě binárního souboru na disk, je tedy možno při příštím načtení aplikace znovu použít známý vzhled a známé rozestavení grafů.

2 Cíle práce

1. Analyzovat a navrhnout zobrazovací systém pro PC (OS Windows nebo Mac OSX), který umožní vizualizovat měřené veličiny v reálném čase mikropočítačovými systémy sloužícími pro výuku v učebně hardware.

2. Program musí umožňovat vizualizaci jak spojitých, tak diskrétních stavů.

3. Program bude přebírat data na hard-disku a zobrazovat. Součástí bakalářské práce je návrh i datového modelu vhodného pro tyto účely.

4. Student vypracuje podrobnou softwarovou dokumentaci, popíše funkce a interface zobrazovacího systému v manuálu pro použití ve výuce.

5. Součástí práce jsou i akceptační testy provedené třetí osobou.

3 Základní principy systémů reálného času

Tyto systémy zpravidla řeší metodu souběžného zpracování procesů. Systém musí vždy dát odpověď v předem stanovené lhůtě. To znamená, že musí být deterministický, tedy konečný v čase. Takové systémy mají reakční dobu odezvy menší než 10mils, a používají zpravidla synchronizační a komunikační nástroje definované v normě POSIX.

POSIX je označení standardu definující rozhraní přenositelnosti zejména u Linuxových systémů. Standard má zajistit přenositelnost aplikací mezi různými HW platformami.

4 Real-time aplikace

Tyto systémy kladou požadavky nejen na logickou správnost informace, ale také na časovou správnost. Logická správnost znamená dostávat správné výsledky například $10+1 = 11$. A časová správnost znamená, abychom tento výsledek dostali ve správném čase. Tímto je myšleno, aby požadovaný výsledek byl připraven v pravou chvíli, jako třeba vyhodnocování výšky při přistávání letadla.

Tyto aplikace musí reagovat predikovaně na nepredikovatelné události, které když splní do svého ultimátního času, dá se říci že chování systému je předvídatelné. A také musí mít deterministické vlastnosti, na které je kladen důraz.

Tyto systémy potřebují také zdroje, které chápeme jako tyto: procesor, paměť, disk... . Jsou zařazeny do skupin:

- Dostatečné – Téměř každý návrh může být použit pro naplnění časových požadavků.
- Nedostatečné – Požadavky na časovou správnost jsou tak velké, že přesahují současnou technologickou hranici. Proto nemůže být žádný návrh použit k naplnění časových požadavků.
- Dostatečné – Časové požadavky jsou splnitelné s pečlivě uzpůsobeným nakládáním se zdroji.

Příklad popíši pseudokódem:

Nastav přerušení T, které je periodické. V každém přerušení T nastane kontrola délky souboru. Pokud je délka souboru stejná, pokračuj v periodách, pokud se délka liší, načti nová data se souboru do paměti.

Takto zjednodušeně funguje jednoduchý systém reálného času. Nastavení periody T může být méně jak sekundové např. 50x za sekundu zkontroluj hodnotu dat z teplotního senzoru. Klasifikaci aplikací reálného času popisujeme takto:

- Čistě cyklické - úlohy jsou spouštěny peridiodicky s neměnnými požadavky na prostředky
- Převážně cyklické- úlohy se většinou spouští periodicky s očekáváním reakce na externí asynchronní události
- Asynchronní a predikovatelné- úlohy nejsou periodické a jejich požadavky na prostředky a zdroje se během spuštění aplikace mohou značně lišit, avšak s danými mezemi a rozložením
- Asynchronní a nepredikovatelné- reagují na asynchronní události, úlohy mají vysokou složitost

U real-time aplikace se můžeme setkat s těmito názvy:

- Úloha : úsek kódu, množina dávek spouštěná pro vykonání určité funkce.
- Dávka: instance úlohy dávky potřebují k běhu zdroje: CPU, síť, disk, ...
- Čas uvolnění dávky: Hodnota, kdy je dávka připravena k běhu, znázorněna časem
- Termín dokončení dávky: okamžik, do kdy dávka musí skončit, vyjádřený časem
- Relativní termín dokončení dávky: Délka intervalu mezi časem uvolnění a dokončení
- Doba odezvy: skutečná hodnota dokončení, vyjádřená časem

4.1 Real-time systémy se dělí na Hard a Soft systémy

4.1.1 Hard

Tyto systémy mají daný termín dokončení, který je striktně dodržován a musí být bezpodmínečně dodržen. Tyto systémy mají při nedodržení této striktní hranice dokončení fatální až katastrofální následky (vysunutí podvozku letadla).

4.1.2 Soft

Soft systémy nemají striktně dané termíny dokončení úlohy, tento termín může být někdy porušen. Avšak ne příliš často.

5 Real-time OS

5.1 Windows RTX

Úloha windows byla a je prioritní využití pro domácí a kancelářské potřeby, ale Microsoft zkoumal i možnosti nasazení windows jako real-time OS. Toto rozšíření založené na speciálních ovladačích se nazývá RTX a umožňuje Windows být realtime OS, při nezměněném pracovním prostředí. RTX dokáže snížit timelatency z 5 ms na 5 μs. Bohužel se Windows ani v tomto rozšíření nehodí jako plnohodnotný systém reálného času, protože mají příliš málo úrovní priorit vláken, nedostatečně deterministické rozhodování plánovače.

5.2 RTLinux

Je to malý operační systém, který je plně preemptivní. Obsahuje navíc vrstvu, která mezi hardwarem a jádrem Linuxu vytváří virtuální vrstvu. Vlákna RTLinuxu jsou zpracována s pevnou prioritou. Systém byl navržen pro méně výkonné pc, třeba pro řízení robotů a časově citlivých nástrojů a aplikací. Systém byl navržen tak, že sdílí zdroje mezi realtime a nonreal-time procesy, s tím že nonreal-time procesy nemůžou blokovat běžící realtime procesy a snadno sdílí data. Proto RTLinux poskytuje možnost spouštět procesy na stejném stroji jako běží normální Linux. Přerušení nastane od události do 15 μs. Systém by měl být transparentní, modulární a snadně rozšiřitelný.

5.3 PikeOS

Tento systém je založený na správě bezpečnosti a ochrany kritických systémů. Systém je založený na dělené podpoře pro více OS s různými cíli, bezpečnostními požadavky a ochrany sloučené v jednom stroji. Systém musí zajistit správné chování několika systémů postavených vedle sebe v jednom stroji, které mají lišící se úrovně kritičnosti. Systémy spouštěné v této architektuře běží i s aplikacemi ve virtuálních strojích. Tyto stroje mají každý vlastní sadu zdrojů, a tak aplikace spuštěné v jednom stroji nejsou závislé na druhém. Vzhledem k oddělování časových zdrojů můžou na PikeOS také fungovat Hard real-time systémy.

5.4 VxWorks

Nejvíce rozšířený systém pro řízení v reálném čase. Ze začátku byl používán pouze jako vylepšení pro real-time OS VRTX. VRTX byl jednoduchý a postrádal potřebné funkce. Po zavedení VxWorks se VRTX dostalo rozšíření o souborový systém a vývojové prostředí. V roce 1987 bylo vyvinuto samostatné jádro VxWorks. VxWorks je vyvinut jako čistě vestavěný systém pro řízení procesů.

VxWorks je multitasking os s preemptivním plánováním a s možností spouštět neomezený počet procesů. Díky 256 úrovní priorit je jasně dáno, který proces má prioritní právo na přidělení zdrojů.

VxWorks se díky jeho účinosti také podíval do vesmíru, jako třeba v 90.letech na palubě kosmické sondy Clementine. Nebo na palubě sondy Phoenix, která dokonce přistála na povrchu Marsu, aby získala informace, jestli je tam možný mikroskopický život.

6 Real-time přístup k databázi a datům

Vysoké nároky jsou kladeny nejen na real-time softwary, ale také při velkém objemu dat na databáze. Tradičně se data v real-time systémech zpracovávají ve strukturách zcela závislých na aplikaci. Kvůli neustálému rozšiřování aplikací se vyžaduje efektivnější přístup ke stále většímu množství dat. Takže potřebujeme uchovávat data v organizované

podobě. Systémy, které mimo schopností databází mají i vlastnosti systémů s reálným časem, se nazývají databázové systémy reálného času(RTDBS).

Tyto databáze ukládají data efektivními algoritmy a dokážou vyhledávat data a manipulovat s nimi. Jako příklad lze znovu využít databázové systémy v letectví, ve výrobních provozech a v systémech řízení provozu. V porovnání s konvenčními systémy ty real-time při práci data neuzamykají při čtení, aby se dala měnit. Dá se tedy předvídat délku zpoždění při blokování transakcí, jelikož konvenční databázové systémy při blokování transakcí mají podobu kaskády.

6.1 Charakteristika RTBDS

Přístup k datům v časovém omezení nebo k datům, která mají dočasnou platnost. Tato data jsou získávána v diskrétních intervalech. Při stálém získávání dat se stávají méně přesné, až se dosáhne bodu, kdy hodnoty neodráží skutečný stav. Proto rozlišujeme tyto pojmy:

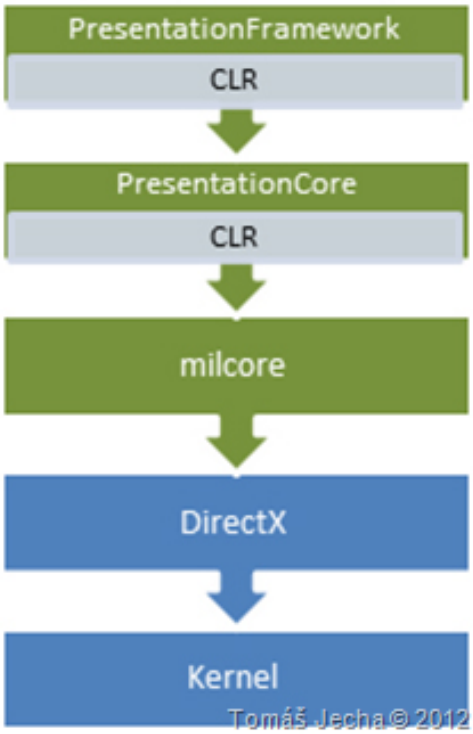
- Časová značka – čas, kdy byla data získána.
- Absolutní interval platnosti – interval, kdy je získaná hodnota považována za platnou.
- Relativní interval platnosti – takový interval je mezi daty pořízenými v čase za sebou, například radarem změřená rychlost letadla a jeho poslední poziční údaj

Transakce v reálném čase se nazývají transakce, které mají explicitně definovány časové požadavky. Tyto transakce kladou časové požadavky vynucené systémovým prostředím a platností dat. Korektnost real-time transakcí tedy nezáleží na rychlosti výpočtů, ale také na čase vyřízení výsledku transakce.

Dá se tedy říci, že RTDBS jsou vylepšené databázové systémy podporující transakce v reálném čase. Tyto transakce musí být vyřízeny do kritického termínu, aby byly maximálně přínosné pro aplikaci. Tyto požadavky lze ovšem obtížně splnit, jak již víme z real-time OS. Ve RTDBS může dojít k takzvané zpožděné transakci. Taková transakce může mít pozitivní, nulový nebo negativní přínos.

7 Windows Presentation Foundation

7.1 Úvod do WPF



Obrázek 1: Architektura WPF

Windows Presentation Foundation (WPF) je produktem firmy Microsoft dříve vyvíjené pod názvem Avalon. WPF je podmnožinou .Net Frameworku 3.0, jako nástupce staršího Windows Forms, které začaly narážet na některá omezení, spojená převážně s tím, že Windows Forms jsou především určena pro Windows. Díky novému modelu a použití DirectX, které slouží jako přímá komunikace s hardwarem především grafickou kartou, je tedy snadné vykreslovat 2D a 3D modely. Nejnovější verze přišla s .Net Frameworkem 4.0 čili WPF 4.

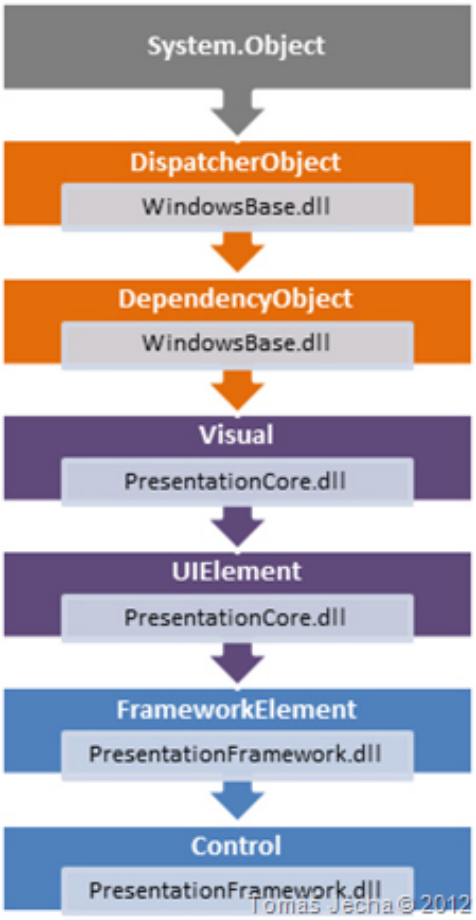
Překlenutí mezi DirectX a WPF rozhraním, se kterým komunikuje C# se nazývá milcore. Tato vrstva běží mimo .NET, je zavedená kvůli zvýšení výkonu. S vrstvou milcore komunikuje PresentationCore, které nám nabídne základní rozhraní WPF. A nakonec PresentationFramework rozšiřuje prostředky a funkce.

Nesmíme také opomenout windowsové knihovny, nejdůležitější je WindowsBase. Tato knihovna nepomáhá s vykreslováním, má však jiné důležité třídy a funkce. Třeba jako management běhu jednoho vlákna, popsané na další stránce.

Úkoly vykonávající se v průběhu aplikace jsou řízeny pomocí jedno hlavního vlákna. Takový přístup se nazývá jedno-vláknový. Díky tomuto se nemusíme starat o synchronizaci, pouze přidáme úkol do vlákna a ten se v pravý čas provede. Mezi hlavní operace patří vykreslování události uživatelského vstupu a změny vlastností.

7.2 Úvod do WPF

Na dalším obrázku jsou znázorněny abstraktní typy, od kterých se dědí ovládací prvky a velká část nevizuálních komponent ve WPF.



Obrázek 2: Objektový model WPF

7.3 Stylizace WPF (jazyk XAML)

Stylování vzhledu WPF probíhá za pomoci jazyka XAML, který je založen na jazyku XML, má tedy i podobný zápis kódu a patří do skupiny značkovacích jazyků. Pomocí jazyka XAML lze jednoduše definovat uživatelské rozhraní ve srozumitelné struktuře použitím XAML elementů pro vytvoření prvků UI. Prvky viditelné v aplikaci můžeme vytvořit zápisem XAML a definovat vlastnosti buď jiným XAML prvkem nebo atributy jednotlivých elementů. XAML elementy jsou reprezentovány stejnojmennou třídou ve stejnojmenném prostoru.

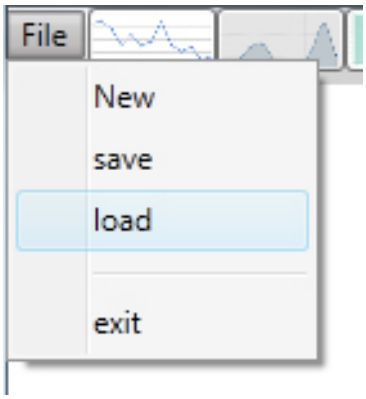
Tím se dá říci, že jazyk XAML pracuje s komponenty definovanými a dodávanými s .Net Frameworkem a můžeme tedy celý kód XAML zapsat pomocí jazyka C# díky oddělení definice vzhledu uživatelského rozhraní a programové funkčnosti. Toto umožňuje týmům pracovat zároveň na vzhledu aplikace a i na její funkčnosti. Jazyk XAML dokáže také jednoduše pracovat s jakoukoliv přiřazenou knihovnou. Na následujícím obrázku syntaxi popíši.

```
1  <Window x:Class="amChart.Editor"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4          Title="MainWindow" Height="800" Width="1200"
5          WindowStartupLocation="CenterScreen" WindowState="Maximized">
6      <Grid>
7          <Grid.ColumnDefinitions...>
11         <Grid.RowDefinitions...>
15         <WrapPanel Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="2" Background="Li
16             <Menu>
17                 <MenuItem Header="_File">
18                     <MenuItem Header="_New" Click="MenuItem_New"/>
19                     <MenuItem Header="_save" Click="MenuItem_Save"/>
20                     <MenuItem Header="_load" Click="MenuItem_Load"/>
21                     <Separator/>
22                     <MenuItem Header="_exit" Click="MenuItem_Exit"/>
```

Obrázek 3: Sintaxe XAML

Zápis jazyka XAML vždy začíná tagem <Window>, ve kterém jsou jako jeho atributy nastavené namespaci, tedy vlastnosti celého okna, v tomto případě například atribut WindowState=“Maximized“, který nám říká, že při spuštění programu bude okno aplikace maximalizováno. Dále může následovat některý s kontejnerů, které se dají lehce stylovat jako třeba Grid nebo WrapPanel.

Jednotlivé prvky také mohou sestavovat kolekci prvků. Jak si můžeme všimnout výše, prvek MenuItem obsahuje další prvky MenuItem, které se potom zobrazují jako podmnožina tohoto prvku. Každému prvku, jak jsme již řekli, může být přiřazen atribut a také událost.



Obrázek 4: Kolekce prvků znázorněné jako menu (XAML)

8 Volba knihovny pro vytváření grafů

K vykreslování grafů bylo nejdříve vybíráno z nástrojů určených pouze pro prostředí Windows Forms. Pro toto prostředí jsme vybrali knihovnu s názvem ZedGraph, která je programována v jazyku C# a zobrazuje jak 2D lineární grafy, tak také grafy sloupcové a různé měřiče v podobě kruhů nebo polokruhů. O poznání visuálně hezčí a s více možnostmi se jevila volba v podobě knihovny Nevron Chart, která bohužel nenabízí rozsáhlejší OpenSorce knihovny, se kterými by se dalo pracovat. Naštěstí tyto neúspěchy vedly k možnosti zobrazovat grafy v prostředí Windows Presentation Foundation.

Ve Windows Presentation Foundation lze snadno navrhnout podobu uživatelského rozhraní a lze také jednodušeji manipulovat s elementy na obrazovce než ve Windows Forms.

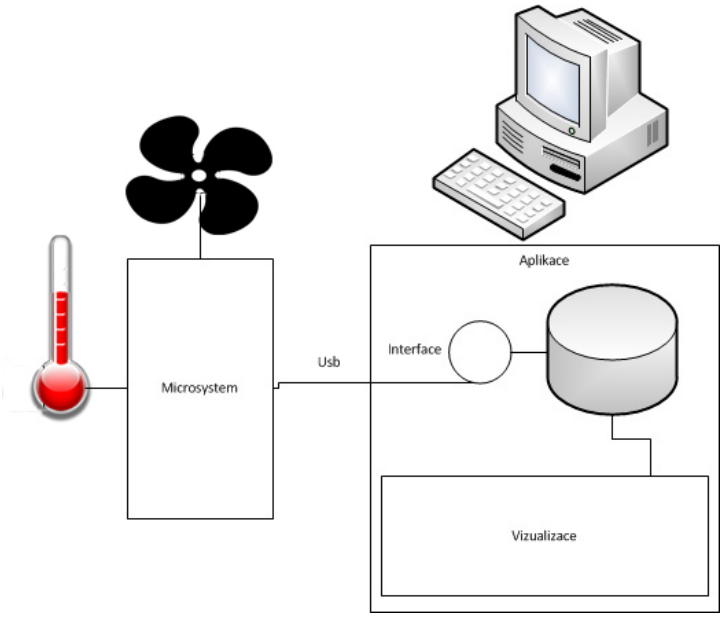
Pro zobrazování grafů byla použita knihovna WPFamCharts, které byla dostupná s rozsáhlou dokumentací a popsanými funkcemi.

8.1 Nástroje použité při vytváření aplikace

Celá aplikace byla vytvořena ve VisualStudiu 2010 verze 10.0.30319.1. Na platformě .NET Framework 4 zejména jeho podmnožiny Windows Presentation Foundation 4. Zobrazené diagramy byly vytvořeny v UML Visual Paradigm v 10.0. a schéma bylo vytvořeno ve Microsoft Visio 2010.

9 Samotný návrh řešení

9.1 Schéma funkčnosti projektu



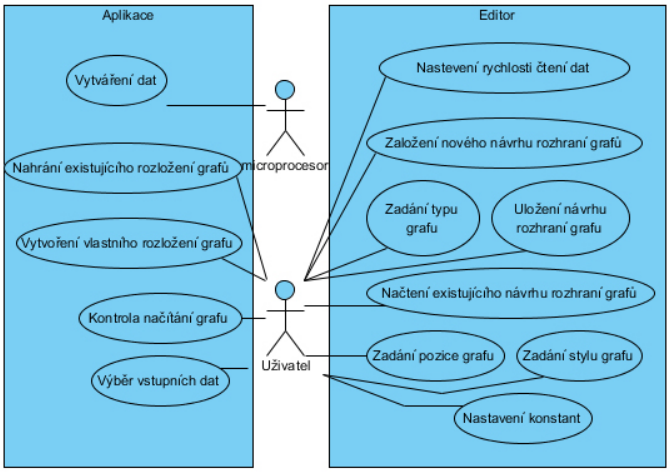
Obrázek 5: Schema

Jak bylo napsáno v úvodu, cílem této práce je vytvořit aplikaci, která bude schopná číst uložená data ze souboru, uloženém na disku. Jak je možné poznat se schéma, microsystém zpracovává údaje o naměřených hodnotách například o teplotě nebo rychlosti větru.

Tyto hodnoty předává pomocí rozhraní USB nebo Sériového rozhraní do PC, kde se tato data ukládají do textového souboru na disk. Když dojde k zápisu na disk, vytvořená aplikace zobrazí uložené hodnoty pomocí lineárního grafu nebo jako konstantní hodnoty.

9.2 UseCase diagram

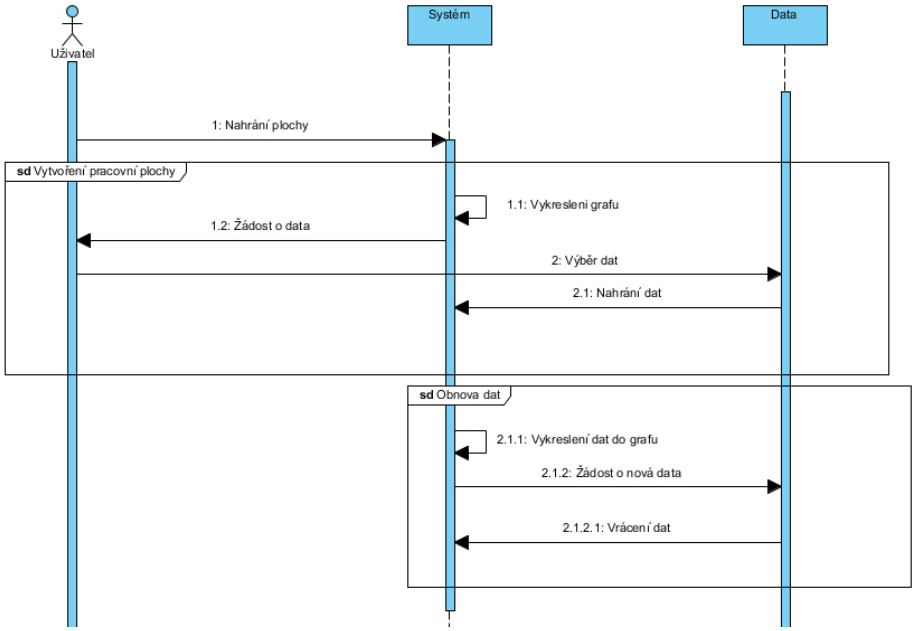
Před samotným začátkem programování jsme si vytvořili některé diagramy, podle kterých by měla výsledná aplikace fungovat. Abychom mohli začít měřit hodnoty uložené



Obrázek 6: UseCase diagram

v souboru na lokálním disku, musíme nejdřív umět hodnoty zobrazovat, k tomu slouží Editor. Také složí pro návrh vlastního uspořádání grafů na obrazovce zobrazující měřené hodnoty, které získáme od microprocesoru zastoupeném např. teploměrem, slouží. Díky Editoru můžeme napozicovat jednotlivé grafy na obrazovku jak chceme a v jakém chceme měřítku. Těmto grafům můžeme také nastavit některé atributy, které rozliší jednotlivé grafy stejného typu. Grafům, které se obnovují za určitý časový úsek, lze také nastavit právě tuto obnovovací periodu. Pojmenování měřených konstantních hodnot najdeme v pravé části editoru. Návrh grafů si můžeme uložit na disk a později znovu přepozicovat.

9.3 Sequence diagram hlavní aplikace

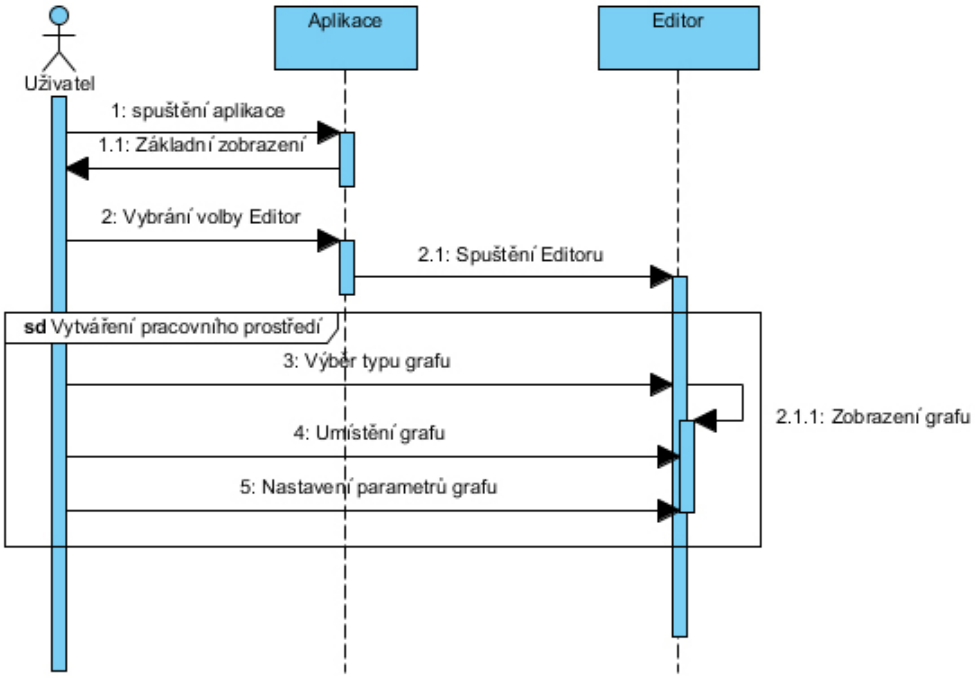


Obrázek 7: Seqvence diagram aplikace

Při spuštění aplikace je hlavní okno čistě bílé, tedy prázdné. Jedinou funkční oblast najdeme v levém horním rohu. Při otevření nabídky menu můžeme otevřít Editor popsaný sequence diagramem níže. Nebo si můžeme načíst již připravené návrhy z minula. Po načtení návrhu z minula se nás bude aplikace při nahrávání každého grafu ptát na zdrojový soubor, ze kterého bude číst data zobrazovaná na obrazovku v podobě grafů. Data jsou po řádcích neustále ukládaná na disk do souboru z měřící jednotky, v našem případě teploměru.

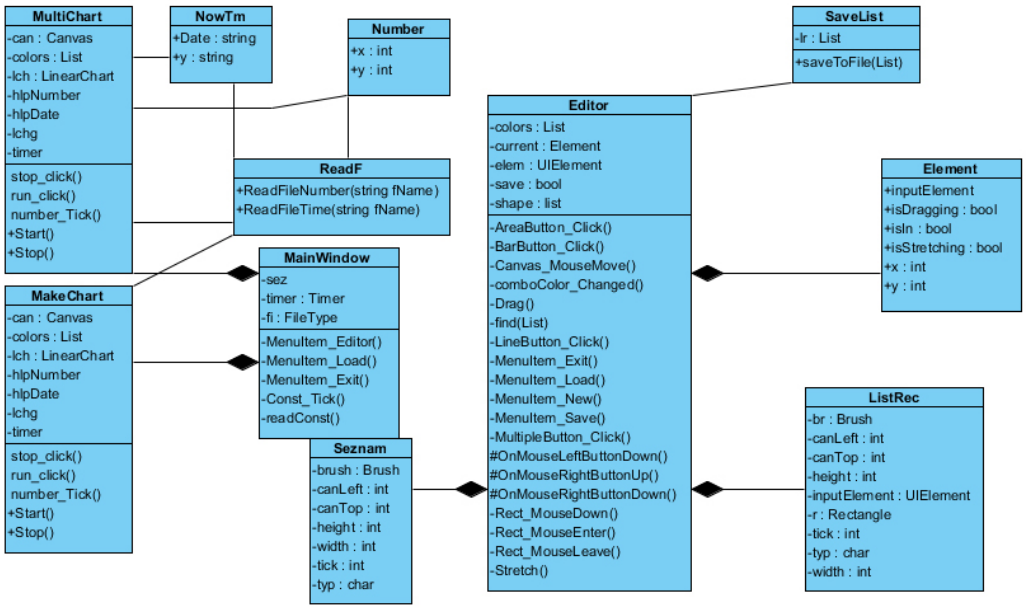
9.4 Sequence diagram Editoru

Editor spustí uživatel přístupem skrze menu v hlavní aplikaci. Po spuštění Editoru může začít pozicovat grafy, které bude pro svou práci potřebovat a měnit jejich vlastnosti. Po rozestavení grafů může návrh uložit do binárního souboru.



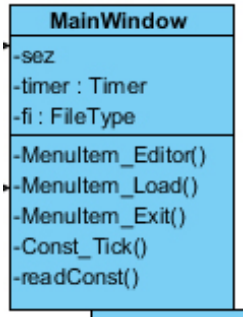
Obrázek 8: Seqvence diagram editoru

9.5 Class diagram



Obrázek 9: Class diagram

9.6 MainWindow.cs



Obrázek 10: Třída MainWindow

Hlavní třída aplikace, která spouští uživatelské prostředí. Toto uživatelské prostředí je popsané jazykem XAML, díky kterému si můžeme navrhnout vzhled a potom vytvářet funkce programovacím jazykem C#. Při spuštění aplikace vidíme pouze prázdnou pracovní plochu a v levém horním rohu tlačítko menu. V nabídce menu nalezneme možnost spustit Editor, který je popsaný níže nebo nahrání vytvořeného návrhu rozestavení grafů připraveném právě v Editoru.A posledním tlačítkem exit vypneme aplikaci.

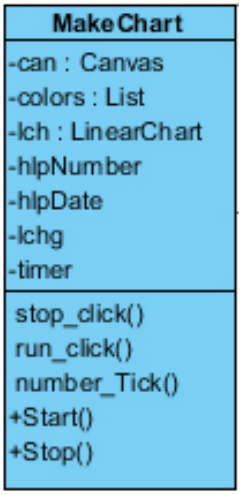
Při stlačení volby menu load spustíme stejnojmennou metodu. Tato metoda spustí OpenDialog, kde najdeme binární soubor, ve kterém máme uložené naše připravené návrhy grafů vytvořené v Editoru. Následně pomocí třídy Stream otevřeme binární soubor, jehož obsah díky deserializaci ze třídy BinaryFormatter načteme do kolekce objektů třídy seznam, o těchto objektech se dozvíme více níže. Tuto kolekci projdeme v cyklu a podle typu zavádíme grafy s parametry, které máme uložené v kolekci seznam.

```
using (Stream str = File.Open(fileName, FileMode.Open))
{
    BinaryFormatter bin = new BinaryFormatter();
    sez = (List<Seznam>)bin.Deserialize(str);
}
if (sez.Count != 0)
{
    foreach (Seznam se in sez)
    {
        if (se.Typ.Equals('l') || (se.Typ.Equals('a')))
        {
            MakeChart mch = new MakeChart(se, canvas);
        }
        else if (se.Typ.Equals('m'))
        {
            MultChart ml = new MultChart(se, canvas);
        }
        else if (se.Typ.Equals('b'))
        {
            BarCh b = new BarCh(se, canvas);
        }
```

Obrázek 11: funkce load

V kolekci seznam máme také uložené názvy konstant. Tyto názvy nahrajeme při průchodu cyklu a zobrazíme je v pravé části okna. Pokud se v kolekci vyskytují některé tyto návrhy konstant, spustí se automaticky časovač. Tento časovač při každém průchodu kontroluje délku souboru nacházející se na disku, do kterého se ukládají data. Pokud jsou do tohoto souboru přidána některá data, zobrazí se do příslušného okénka pro zobrazování konstant.

V aplikaci zobrazujeme lineární data 2 různými druhy grafů. Lineárním, který ještě můžeme rozdělit na čistě lineární, lineární s vybarvenou dolní části grafu a grafem zobrazujícím více hodnot ve stejném čase, a sloupcovým který není ideální pro reálné zobrazování dat, ale zobrazuje se s malým zpožděním.



Obrázek 12: Třída MakeChart

9.6.1 MakeChart.cs

Díky třídě MakeChart vytváříme lineární grafy. Při vytváření objektu graf musíme nejdříve zadat soubor s daty, ze kterého budeme získávat data. K načtení znovu použijeme OpenDialog, ve kterém lze snadno najít ty soubory, ze kterých budeme data číst. Po nahrání souboru ověříme, jaká data čteme. O ověřování a podobě datového souboru se více dozvíme ve třídě ReadF.cs. Po zjištění podoby dat si zavedeme objekt časovače, pomocí kterého zobrazujeme postupně data v grafu. Následuje samotné sestavení grafu z parametrů předaných v kolekci objektů typu seznam.

V předešlé kapitole jsem zmínil, že lineárních grafů je více typů. Lineární grafy však pracují na stejném principu pouze mají malinko odlišný vzhled.

Do grafu také přidám dvě tlačítka s metodami stop a run. Těmito tlačítky buď spustíme čtení dat nebo toto čtení můžeme pozastavit a zkoumat vychýlení linky. Tlačítkem run spustíme časovač, který podle typu dat spouští buď metody number_Tick nebo date_Tick.

Při každém průchodu časovače kontrolujeme ve zmíněných metodách délku souboru, ze kterého čteme data. Pokud byla do souboru přidána některá data, spustí se metoda readFile ze třídy ReadF.cs. Tato načtená data se přidají do kolekce objektů tříd number nebo nowTm. V grafu může být pro dobrou čitelnost zobrazeno pouze 60 hodnot, pokud

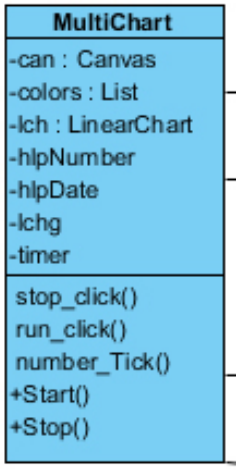
```
fi = new FileInfo(fileName);
long fl = fi.Length;
if (fl > fileLength)
{
    int c = hlpNumber.Count + 1;
    ObservableCollection<Number> n = new ObservableCollection<Number>();
    n = rf.ReadFileNumber(fileName);
    foreach (Number nu in n)
    {
        hlpNumber.Add(new Number() { x=c,y=nu.y });
        c++;
    }
    fileLength = fl;

}
```

Obrázek 13: Kontrola souboru pomocí FileInfo

zobrazujeme více dat, začne se graf posouvat směrem zprava doleva. Pokud v zobrazované kolekci dojdeme nakonec, zobrazí se popisek, který nám říká, že nemohou být další zobrazovaná, protože do souboru s daty nebyla žádná data přidána.

9.6.2 MultiChart.cs

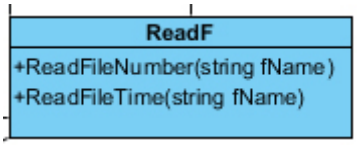


Obrázek 14: Třída MultiChart

Tato třída zobrazuje grafy s více linkami zobrazující jiná data čtených souborů. Způsob hledání textových souborů, jejich kontroly a následného zobrazení je totožný jako u třídy MakeChart.cs, liší se ovšem v sestavení samotných grafů. Načtené soubory musí být číselného typu, jinak se zobrazený graf nenačte a bude stále zobrazovat vyhledání a načtení souborů.

Dále se při průběhu časovače kontroluje vždy pouze první soubor a podle toho se zobrazují data.

9.6.3 ReadF.cs



Obrázek 15: Třída ReadF

Tato třída obsahuje metody pro čtení dat z textových souborů. Data uložená v textovém souboru můžou být číselného typu, která zobrazujeme na ose Y a naose X pouze přičítáme jednotky. Nebo mohou být data ukládaná s časem pořízení. Tento čas je stringového typu a zobrazuje se vždy k příslušné hodnotě na ose X.

Odlišnost čtených dat poznáme podle prvního znaku na začátku každé řádky. Tento znak je buďto n pro pouze číselná data, nebo d pro data s časem pořízení. Znak se kontroluje hned po vyhledání souboru. Podle tohoto znaku spouštíme metody pro čtení dat ze souboru.

Tyto dvě metody se liší jak ve způsobu čtení textového souboru, tak i ve způsobu zápisu do kolekce, mohou být buď objekty tříd Number.cs nebo NowTm.cs.

```
public ObservableCollection<Number> ReadFileNumber(string fName)
{
    ObservableCollection<Number> n = new ObservableCollection<Number>();
    string line;
    using (StreamReader read = new StreamReader(fName))
    {
        line = read.ReadLine();
        string[] words = line.Split('$');
        for (int i = 1; i < words.Length;i++ )
        {
                n.Add(new Number() { x = i, y = Convert.ToInt32(words[i]) });

        }
        read.Close();
    }
    xml read
    return n;
}
```

Obrázek 16: Užití parsování pro načtení hodnot do pole

Metoda ReadFileNumber otevře soubor pomocí třídy StreamReader. Ze souboru nám stačí přečíst pouze první řádku, protože při nahrávání do souboru zapisujeme nová data vždy na začátek. Tuto řádku přečteme metodou readLine ze třídy StreamReader.

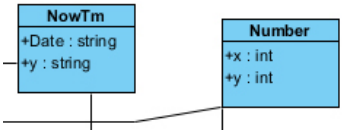
Řádka v souboru bude vypadat nějak takto:

n$53$42$11$17$22$27$32$85$91$32$1$6$59$64$70$7.

Na začátku každé řádky je vždy znak, o kterém jsme mluvili výše, potom následují čísla oddělená znakem $. Díky tomuto znaku můžeme každý řádek rozparsovat do pole stringů. Z tohoto pole dále v cyklu čteme data do kolekce. Tato kolekce se předává k vykreslení grafů.

Metoda ReadFileTime se chová podobně. Protože, ale musí číst ze souboru i čas pořízení dat, načítá vždy dva první řádky, které začínají znakem d a data jsou znovu rozdělena znakem $. $.

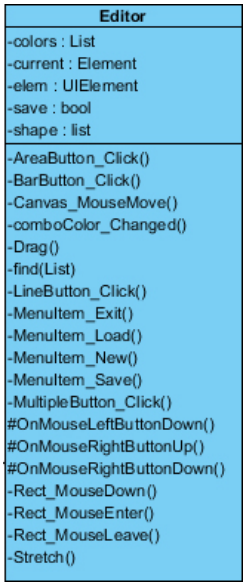
9.6.4 Number.cs a NowTm.cs



Obrázek 17: Třídy Number a NowTm

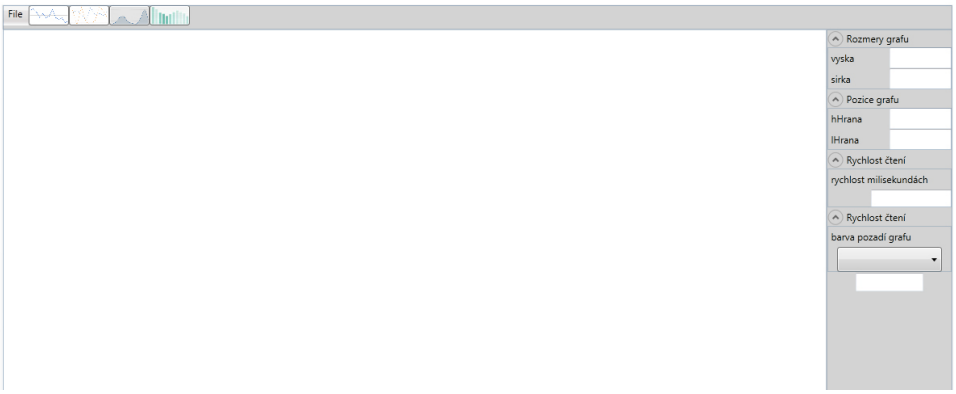
Tyto dvě třídy slouží jako objekty kolekcí, do nichž se ukládají data naměřená teploměrem, které jsme přečetli ze souboru. Tyto třídy tedy obsahují getry a setry pro osu x a osu y. Třídy jsou také připraveny na ukládání do xml souboru, které bylo implementováno dříve, ale upustili jsme od něj kvůli komplikovanému znovupřepisování souboru s daty.

9.7 Editor.xaml.cs



Obrázek 18: Třída editor

Tato třída skrývá veškerou funkčnost návrhového editoru a také inicializuje a vykreslí prostředí navržené XAML jazykem, které vypadá takto.



Obrázek 19: Vizuální podoba editoru

Nejdříve bychom měli začít v levém horním rohu, kde se nacházejí tlačítka znázorňující modely grafů. Po kliknutí na některé z tlačítek vyvoláme příslušnou metodu, která vykreslí do pracovní plochy element typu rectangle, jenž zastupuje podobu grafu. Při vytváření tohoto elementu je mu nastavena defaultní výška, šířka a poloha zastoupená souřadnicovým systémem x, y. Tyto hodnoty se samozřejmě mění společně s pohybem elementu po pracovní ploše nebo jeho transformacemi.

Vytvořením elementu také vytváříme nový objekt typu ListRec do kolekce shape. V této kolekci uchováváme veškeré elementy na pracovní ploše a s nimi také informace o jejich vlastnostech. Takto se chovají všechna tlačítka vytvářející grafy.

Dále se přesuneme k samotnému ovládání a pohybu elementů po pracovní ploše. Stlačením levého tlačítka myši voláme metodu OnMouseLeftButtonDown, díky které přiřadíme element do proměnné elem. Tato metoda je typu UIElement. UIElement je základní třída WPF, definuje základní subsystémy, včetně rozložení vstupů a událostí. Díky tomu lze elementu přidat nebo od něj získat vlastnosti. Vytvořené elementy na pracovní ploše jsou také zastoupené jako objekty třídy Element.cs, která povoluje elementy hýbat, když je tlačítko stlačené atd. Puštěním levého tlačítka uvolníme element.

Stlačením pravého tlačítka mažeme nepotřebné elementy na pracovní ploše. Touto událostí vybereme příslušný element a vymažeme ho z pracovní plochy. Vymazaný element musíme také vymazat z naší kolekce uchovávající informace o elementech shape. Po celou

```
protected override void OnMouseRightButtonDown(MouseButtonEventArgs e)
{
    base.OnMouseRightButtonDown(e);
    elem = (UIElement)this.current.InputElement;
    shape.Remove(find(shape));
    canvas.Children.Remove(elem);

}
```

Obrázek 20: Metoda pro mazání elementů

dobu pohybu myši po pracovní ploše funguje metoda Canvas_MouseMove. Tato metoda, pokud stiskneme levé tlačítko myši, získá informace o transformaci elementu ze třídy Element.cs. a zavolá buď metodu Drag po pohyb po pracovní ploše nebo metodu Stretch pro roztahování elementu na výšku a šířku.

Metoda Drag umožňuje pohyb elementu po pracovní ploše. Při volání této metody měníme kurzor myši na ruku a také hodnoty její pozice. A dále díky vlastnosti TranslateTransform z TransformGroup můžeme hýbat elementem, tedy pokud držíme levé tlačítko myši. Při zvednutí tlačítka nastavíme konečnou pozici elementu příslušnému záznamu v kolekci shape.

```
Rectangle nRec = new Rectangle();
nRec.Height = 250;
nRec.Width = 350;
Canvas.SetTop(nRec, 150);
Canvas.SetLeft(nRec, 150);
nRec.MouseLeftButtonDown += Rect_MouseDown;
nRec.MouseEnter += Rect_MouseEnter;
nRec.MouseLeave += Rect_MouseLeave;
nRec.Name = "Rec0";
TransformGroup grp = new TransformGroup();
nRec.RenderTransform = grp;
grp.Children.Add(new TranslateTransform());
grp.Children.Add(new ScaleTransform());
```

Obrázek 21: Přidání TranslateGroup do vlastností rectanglu

Metodou Stretch modifikujeme výšku a šířku elementu. Tyto modifikace mají jistá omezení ve zmenšování elementu, kdy by se s výslednou podobu grafu nedalo pracovat. Směr roztahování elementu poznáme podle podoby kurzoru myši, měnící se po vyjetí z oblasti elementu na příslušné hraně. Při používání těchto metod zobrazujeme informace o elementech v pravém informačním panelu.

```
if (this.Cursor == Cursors.SizeWE)
{
    ((Rectangle)this.current.InputElement).Width += xDiff;
}
else if (this.Cursor == Cursors.SizeNS)
{
    ((Rectangle)this.current.InputElement).Height += yDiff;
}
else
{
    this.Cursor = Cursors.Arrow;
    this.current.IsStretching = false;
}

this.current.X = mouseX;
this.current.Y = mouseY;
```

Obrázek 22: Rozhodnutí směru roztahování na základě podoby kurzoru

Dále máme funkce myši spojené přímo s elementem jako je najetí na element a vyjetí z něj. Při najetí na element měníme kurzor myši. Znovu získáme polohy kurzoru a aktuální element, ve kterém se nacházíme a pokud je poloha jinde než jsou strany elementu, změníme kurzor znovu na šipku.

Pokud opustíme element spodní nebo levou hranou, voláme metodu MouseLeave, která v oblasti těsně u hrany elementu nastavuje podobu kurzoru myši. Tyto kurzory mají podobu vodorovné nebo svislé roztahovací šipky. Podle těchto kurzorů poznáme v metodě Stretch, jestli budeme element modifikovat na výšku nebo šířku. Meto Stretch zavoláme když budou změněné kurzory myši.

Zmíněna byla také několikrát metoda find. Touto metodou vracíme objekt z kolekce shape typu ListRec. Tuto metodu potřebujeme zejména při modifikování elementu, se kterým momentálně pracujeme. Metodu také využijeme pro vymazání objektu z kolekce při mazání elementu. Metoda find zobrazuje i hodnoty elementu v pravém informačním panelu.

Jako poslední vysvětlím metody spojené s menu. V menu se nachází čtyři volby, a to volba New, Save, Load a exit.

Při volbě new nás systém vyzve ,pokud nemáme dosavadní návrh uložený, k jeho uložení. Ať už návrh uložíme nebo ne, volba new vymaže stávající elementy z pracovní plochy a vyčistí také kolekci shape, ve které elementy uchováváme.

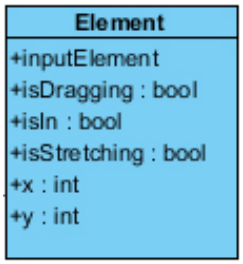
Pokud chceme návrh uložit, ať už při zmáčknutí samotného tlačítka save nebo při vyzvání aplikace, vytvoříme novou kolekci objektů z třídy Seznam.cs. Do této kolekce přepíšeme v cyklu všechny objekty uložené v kolekci shape. Toto přepisování děláme kvůli proměnným ve třídě ListRec.cs, které nejdou serializovat do binárního souboru. Zavoláme metodu saveToFile ze třídy SaveList a jako parametr použijeme právě námi nově vytvořenou kolekci.

Když budeme chtít něco změnit na uloženém návrhu, můžeme si ho jednoduše nahrát tlačítkem load. Zobrazí se OpenDialog pro otevírání souboru, kde můžeme najít chtěný soubor. K otevření znovu slouží Stream a BinárníFormátování, kterým si do kolekce objektů třídy Seznam.cs načteme uložené elementy. A dále je v cyklu vykreslíme na pracovní plochu.

A poslední tlačítko Exit upozorní na neuložený návrh a vypne editor.

Třída Editor obsahuje ještě dvě metody, jedna z nich hlídá změnu políčka s hodnotou frekvence kontroly změny délky souboru, ze kterého čteme data. A druhá metoda mění pozadí grafů. Obě metody mění tyto hodnoty i v kolekci shape.

9.7.1 Element.cs

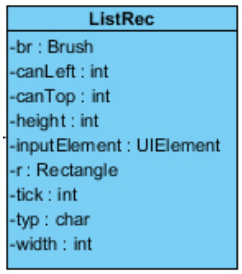


Obrázek 23: Třída element

Třída Element.cs je základem Editoru. V této třídě se uchovávají informace o elementech, kterými můžeme hýbat nebo je roztahovat na pracovní ploše, třeba isDragging nám při vrácení false zabraňuje hýbat s objektem, když není stisknuté tlačítko myši. Tato třída se přiřazuje každému vytvořenému elementu na pracovní ploše. Ve třídě najdeme pouze getry a setry, díky kterým uchováváme polohu elementů.

V této třídě najdeme také IInputElement. Rozhraní tohoto typu zavádí běžné události jako právě zmíněné získání hodnot a přidává události prvku a zavádí vlastnosti a metody související s WPF.

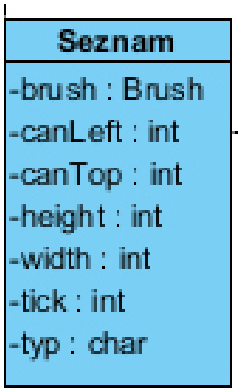
9.7.2 ListRec.cs



Obrázek 24: Třída ListRec

ListRec zastupuje objekty v kolekci shape. Tyto objekty uchovávají veškeré informace o elementech na pracovní ploše.

9.7.3 Seznam.cs



Obrázek 25: Třída Seznam

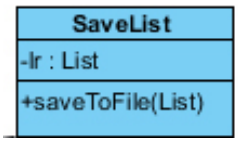
Třída má atribut Serialiazable, díky kterému můžeme celou třídu ukládat do binárních souborů. Tato třída slouží jako zástupce objektu v seznamu při ukládání návrhu do souboru.

```
 8   {   [Serializable]
 9       public class Seznam
10       {
11           private double width;
12           public double Width...
17
18           private double height;
19           public double Height...
24
```

Obrázek 26: Serializace kolekce

Ve třídě jsou getry a setry polohy, typu a ostatních vlastností elementů pro vytvoření grafu. Do této třídy se přepisují vlastnosti elementu z kolekce shape typu ListRec, protože ve třídě ListRec uchováváme informace o celém Rectanglu a IInputElementu a tyto dvě položky nejdou serializovat do binárního souboru, navíc je ve výsledném vykreslení grafů ani nepotřebujeme.

9.7.4 SaveList.cs



Obrázek 27: Třída SaveList

Tato třída má pouze jednu metodu a to ukládání navržené pracovní plochy do binárního souboru, buď s příponou .bin nebo .dat. Metoda saveToFile s parametrem kolekce typu seznam vyvolá ukládací dialog známý z prostředí windows, kde si můžeme zvolit kam soubor uložíme a také jeho pojmenování včetně přípony.
Pro ukládání používáme Stream, který pomocí BinárníhoFormátování zapíše do souboru a Stream potom uzavře.

10 Testování aplikace

Pro testování aplikace jsme si napsali jednoduchou konzolovou aplikaci vytvářející textové soubory s řádky, jako jsme mohli vidět v popisu třídy ReadF.cs

V metodě main zavádíme časovač, který běží v nekonečné smyčce. V každém průchodu časovače uchováváme string, do kterého zapisujeme náhodná čísla vytvářená ze třídy Random. Také vytváříme přesný čas pořízení dat v milisekundách. Hodnoty vždy zapíšeme za sebe do dlouhého stringu oddělené znakem $.

Po 30 průchodech celý string uložíme do textového souboru jako jeden řádek vždy na úplný začátek souboru. To proto, abychom nemuseli při čtení dat procházet celý soubor až na poslední řádku. String potom vymažeme a na začátek celého stringu dáme příslušný znak pro typ uložených dat.

Zápis do textového souboru provádíme StreamWriterem. Nejdříve StreamReader přečte celý soubor a jeho obsah uchová, potom tento soubor vymaže a vytvoří nový jako první řádek, zapíše nová data a za tento řádek pokračuje se zbytkem, který si uchoval.

11 Závěr

Při vytváření aplikace bylo hlavním cílem dosáhnout co nejpohodlnější možnosti přípravy zobrazení měřených veličin tzn. Editoru. Posléze bylo obtížné dosáhnout reálného načítání hodnot, tak aby se data načítala při změně délky souboru s daty.

1. Aplikace byla vytvořena pro operační systém Windows. Byla tvořena na platformě .NET s podmnožinou WPF. Díky knihovně pro usnadnění zobrazování hodnot bylo docíleno dynamckého vytváření grafů s možností si grafy samostaně nastylovat a určit jejich polohu.

2. Spojité stavy se v aplikaci zobrazují pomocí zmíněných grafů, které se ve smyčce stále obnovují. Diskrétní hodnoty jsou zobrazeny na pravé straně obrazovky hlavního okna aplikace. Názvy diskrétních hodnot lze nastavit v editoru.

3. Aplikace načítá data z textového souboru, který musí být formátován v dané podobě, aby aplikace věděla, která data načítá.

4. Popis tříd a jejich funkčnost je zdokumentována v podobě bakalářské práce.

5. Akceptační testy byly provedeny podle předešlé kapitoly. Tedy nahráváním dat z konzolové aplikace do souboru na lokálním disku.

Seznam obrázků

1 Architektura WPF 14
2 Objektový model WPF 15
3 Sintaxe XAML 16
4 Kolekce prvků znázorněné jako menu (XAML) 17
5 Schema 18
6 UseCase diagram 19
7 Seqvence diagram aplikace 20
8 Seqvence diagram editoru 21
9 Class diagram 22
10 Třída MainWindow 22
11 funkce load 23
12 Třída MakeChart 24
13 Kontrola souboru pomocí FileInfo 25
14 Třída MultiChart 25
15 Třída ReadF 26
16 Užití parsování pro načtení hodnot do pole 26
17 Třídy Number a NowTm 27
18 Třída editor 28
19 Vizuální podoba editoru 28
20 Metoda pro mazání elementů 29
21 Přidání TranslateGroup do vlastností rectanglu 30
22 Rozhodnutí směru roztahování na základě podoby kurzoru 30
23 Třída element 32
24 Třída ListRec 33
25 Třída Seznam 33
26 Serializace kolekce 33
27 Třída SaveList 34

Reference

[1] *stackoverflow.com*[online]. [cit. 2012-12-12]. Dostupné z: http://stackoverflow.com/questions/tagged/wpf

[2] *www.wpftutorial.net*[online].2011[cit. 2012-12-12] Dostupné z: http://wpftutorial.net/Home.html

[3] *http://www.codeproject.com*[online]. [cit. 2012-12-12] Dostupné z: http://www.codeproject.com/KB/cs/

[4] *amCharts*,[online].[cit. 2012-12-12] Dostupné z: https://github.com/ailon/amCharts-Quick-Charts?id=4

[5] *www.wikipedia.org*,[online].[cit. 2012-12-12]. Dostupné z: http://cs.wikipedia.org/wiki/Hlavni_strana

[6] KRÓL, Václav. *Odbornecasopisy.: ATOMA* [online]. Slezská univerzita, 27. 4. 2001 [cit. 2012-12-12]. Dostupné z: http://www.odbornecasopisy.cz/index.php?id_document=33758