TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011– Elektronické informační a řídící systémy

Komponenty pro grafický výstup záznamů měření elektrických veličin v prostředí .NET

Components in the .NET environment for graphical output of records from measuring of electrical quantities

Bakalářská práce

Autor: **Lukáš Vančura**
Vedoucí práce: Ing. Jan Kraus
Konzultant: Ing. Tomáš Tobiška

V Liberci 24. 5. 2009

Zde bude vloženo zadání BP

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Abstrakt

Cílem práce je vytvořit počítačovou aplikaci, pomocí které je možné zobrazovat naměřená data v grafech, vypisovat o nich základní informace a umožnit tak uživateli získaná data snadno využít. Proto bylo vyzkoušeno několik programů zabývajících se stejnou problematikou s cílem dodržení standardní metody ovládání, zobrazování apod. Dále bylo nutné vybrat vhodnou komponentu na práci s grafy. Existují různé druhy těchto komponent pro vývojové prostředí Microsoft Visual Studio 2008 a jazyk C#. Musí se proto porovnat podle určitých kritérií a vybrat z nich tu, která se bude do vyvíjené aplikace nejvíce hodit. Dalším krokem je výběr přístupu k datům v databázi. Opět existuje více možností, ale musí se vybrat jen jedna.

Co se týče samotného programu, ten prošel několikrát značnými změnami. Nejčastějším důvodem bylo objevení nových možností komponenty na tvorbu grafů nebo práce s databází. Právě práce s databází se měnila nejčastěji a občas si to vyžádalo rozsáhlé změny v programu. Ve výsledku tak vznikl program, který má potenciál uspět i na poli komerčního softwaru, nicméně to znamená ještě spoustu práce.

V současnosti je program v takovém stavu, že umí zobrazovat několik druhů grafů včetně informování uživatele o zapínání a vypínání různých elektrospotřebičů. Program ještě sám neumí jednotlivé druhy spotřebičů rozpoznat, ale v budoucí verzi programu se s touto funkcí určitě počítá. Jako další se nabízí možnost rozšířit ovládání samotného programu, přidat více způsobů, jak z grafů číst data, nebo zvýšit rychlost aplikace.

Klíčová slova: grafy, vizualizace, databáze, ovládání programu, analýza dat

Abstract

Purpose of this project is to create computer application, which can display measuring data in graphs and write out basic information about them. It also allow user to easily use obtained data. It has been tested several application with same purpose. Goal was to keep standard control, displaying etc. It was also necessary choose right component to work with graphs. There are several components in development environment Microsoft Visual Studio 2008 and programming language C#. All of them must be compare by specific criterion and choose the best one to our application. The next step is the choice of the way of access to data in database. Once again, there are more options, but we must choose the best one.

The program itself was changed few times. The most common reason was the discovery of new possibilities of component for graph control or work with database. Even work with database was changed most frequently. Sometimes were necessary large changes in program. Finally was created program, which have chance be a successful as commercial software. However, it needs a lot of work.

Program can display a few types of graphs and can inform user about turn on and turn off of different electric consumers, in this moment. Program can't recognize different electric consumers. However, program will have this function in future for sure. Next extension could be improvement of controls, addition another way of reading information from graphs and increase speed of program.

Keywords: graphs, visualization, database, controls, data analysis

Obsah

1 Úvod

V posledních několika letech se stále více zdůrazňuje potřeba ochrany životního prostředí. Ač to nemusí být na první pohled zřejmé, tak i šetřením elektrické energie můžeme snížit zátěž pro životní prostředí. Sice existují alternativní zdroje, ze kterých lze elektrickou energii získávat, ale Česká republika využívá především energii z hnědého uhlí a z jádra. O jaderné energii bych zde nerad polemizoval, určitě má své výhody i nevýhody. Naopak fakt, že výroba elektřiny z hnědého uhlí je jasnou zátěží pro životní prostředí, nemůže snad nikdo vyvracet. Náš stále se zrychlující životní styl nebo rozvoj techniky a průmyslu, to vše vede ke zvyšování spotřeby elektrické energie.

Na druhou stranu je zde již zmíněná snaha o šetření energiemi, včetně té elektrické. Vyrábí se proto čím dál úspornější a efektivnější spotřebiče, ale jejich počet se stále navyšuje, a to jak pro osobní účely, tak i ve firmách a podnicích. Právě ve firmách se velké množství počítačů spolu s další kancelářskou technikou podílí výrazným způsobem na spotřebě elektrické energie dané společnosti. Pokud se povede dobře vyhodnotit spotřebu v takovéto společnosti, dala by se navrhnout různá úsporná opatření. Pro samotné vyhodnocení je potřeba nejdříve naměřená data vizualizovat. K tomu je zapotřebí vhodný software. Právě vývoj takovéto aplikace je cílem mé bakalářské práce. Jedná se o program, který chce v budoucnu využívat společnost Seven o. p. s. Ta se zabývá analýzou energetické efektivnosti, úspornými opatřeními a jejich finanční náročností a dalšími činnostmi, souvisejícími s úsporou energie.

K dosažení požadovaných výsledků se využije průběh výkonu a spotřeby. U výkonu však nestačí vyhodnocovat pouze činný výkon. Ten sice umožňuje rozpoznat spotřebu jednotlivých spotřebičů, ale už z něj nepoznáme, o jaký typ spotřebiče se jedná. Proto je nutné měřit i ostatní výkony, zdánlivý, jalový a deformační. Jednotlivé spotřebiče totiž mají různý charakter. Existují čistě odporové spotřebiče, kde se uplatňuje pouze činný výkon. Ale pokud má spotřebič kapacitní charakter, začne se uplatňovat i jalový výkon. Díky tomu by mohlo jít z naměřených hodnot určit, zda byla zapnuta obyčejná žárovka s odporovým charakterem, nebo relé s induktivním charakterem. Měření spotřeby je také důležité, protože pomáhá zjistit, v jaké denní době se odebírá nejvíce elektrické energie.

Základním přístrojem pro měření spotřebované elektřiny je elektroměr. Ten však nenabízí příliš možností, jak přehledně kontrolovat a analyzovat odběr elektřiny. Proto

se v nedávné době začaly na trhu objevovat jednodušší digitální přístroje, které se zapojují před jeden, či více spotřebičů. Většinou umožňují měřit aktuální nebo celkovou spotřebu. Z těchto hodnot je přístroj dále schopen spočítat např. průměrnou spotřebu nebo cenu za spotřebovanou energii. S extrémní přesností se u těchto přístrojů počítat nedá, ale základní přehled o tom, který spotřebič v domácnosti odebírá elektřiny nejvíc, zjistit lze. Pro náročnější měření je použít nelze. Proto se vyrábí daleko složitější přístroje, které měří všechny důležité hodnoty v elektrické síti a ukládají je k dalšímu použití. Pro svůj program využívám data naměřená přístroji SMV 44 s proudovým měřícím transformátorem PEWATRON od společnosti KMB systems. Ty spadají do skupiny průmyslových analyzátorů kvality elektrické energie, protože umožňují měřit nejen spotřebu, ale i aktuální výkon, napětí v síti, odebíraný proud a další parametry a veličiny.

Software je programován v jazyce C# ve vývojovém prostředí Visual Studio 2008. Databáze využívá nejnovějšího SQL 2008. Visual Studio 2008 je rozšířené o knihovny Dxperience. Ty přidají do vývojového prostředí velké množství dalších komponent. Vedle klasických, u kterých je pouze pozměněný grafický vzhled, až po úplně nové komponenty, které v samotné instalaci Visual Studia 2008 nenajdeme. Dalším neméně důležitým rozšířením je komponenta ZedGraph. Ta umožňuje vykreslování nejrůznějších typů grafů od klasických spojitých, přes sloupcové, koláčové, až po speciální grafy. Díky široké možnosti nastavení všech důležitých parametrů je vytvoření grafu rychlé a jednoduché. ZedGraph má LGPL licenci, což znamená, že jej lze volně používat v aplikacích za předpokladu, že se nebude zasahovat do jeho zdrojového kódu. Pokud je to však nezbytné, je povinností tuto změnu zveřejnit.

S aplikacemi zobrazujícími naměřená data v grafech jsem neměl dosud žádné zkušenosti, a proto jsem před začátkem vytváření vlastního programu nainstaloval a vyzkoušel několik obdobných aplikací. Z nich jsem se poučil, jak vhodně zobrazovat jednotlivé průběhy naměřených veličin, jak vypisovat nejdůležitější informace z vybraných úseků apod. Snažil jsem se také vyvarovat nedostatků, na které jsem narazil. Jako příklad mohu uvést zobrazení více průběhů na jednu osu, přestože jejich maximální hodnoty jsou velice rozdílné. Výsledkem je, že ani jeden z průběhů není v grafu zřetelný. Také jsem si všímal, jak snadno se testovaný program ovládá. U některých softwarů bylo ovládání uživatelsky přívětivé a intuitivní, zato jinde bych uvítal rozsáhlý manuál, ale přesto si myslím, že by to ovládání příliš neusnadnilo.

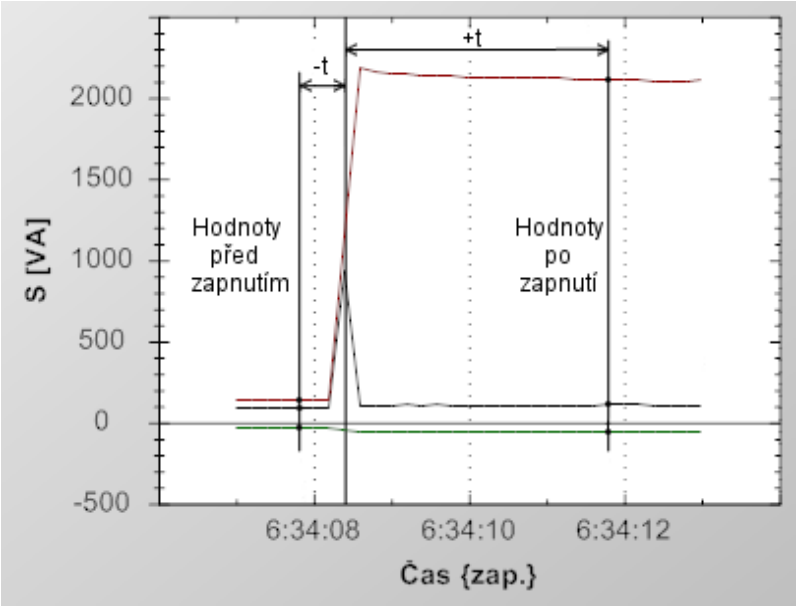
Předpokládané použití v praxi je takové, že se měřící přístroj zapojí do rozvaděče v měřeném objektu. Není tedy zapotřebí pro každou místnost nebo zásuvku pořizovat a zapojovat samostatný přístroj. Měřené hodnoty se budou průběžně ukládat do databáze, ze které je bude možné po skončení měření načíst a vyhodnotit. Základním úkolem při zpracování získaných dat bude určit dobu, po kterou byl daný spotřebič zapnut. K tomu se využije průběh činného výkonu, který má v místech zapnutí a vypnutí zřetelné skoky. Dále se budou využívat i průběhy ostatních typů výkonů, díky nimž by se mělo podařit identifikovat typ zátěže. Ke každému typu zátěže se budou moci dopočítat doplňující informace s využitím všech naměřených průběhů. Další funkcí připravovaného softwaru je zobrazení hodnot do interaktivního grafu. Ten bude sloužit především pro vizualizaci průběhů a časových úseků, kdy byly jednotlivé spotřebiče zapnuty nebo vypnuty. Toto všechno by mělo napomoci k lepší analýze energetické efektivnosti, a tak usnadnit návrh úsporných opatření.

Cílem mé práce je vytvořit software, usnadňující analýzu měření odebírané elektrické energie. V případě správné funkčnosti výsledného algoritmu by měla být možná jeho implementace do elektroměrů, a tak poskytnout doplňkové informace o charakteru odběru. Typickou aplikací mohou být byty, rodinné domy nebo kanceláře.

2 Teoretický rozbor dostupných prostředků

2.1 Metoda zpracování naměřených dat

Zpracování naměřených dat probíhá ve dvou fázích. Nejdříve se určí časové okamžiky zapnutí nebo vypnutí elektrospotřebiče a poté se analyzuje krátký časový úsek v jeho okolí.



Obr. 2.1: Průběh zapnutí elektrospotřebiče

2.1.1 Detekce přechodových jevů

Aby bylo možné naměřená data začít zpracovávat, je potřeba detekovat časové okamžiky, při kterých došlo k zapnutí nebo vypnutí elektrického zařízení. K tomu postačí sledovat skokové změny zdánlivého výkonu. Spolu se zdánlivým výkonem se při zapnutí a vypnutí elektrospotřebičů mění i ostatní výkony, čehož bude možné využít při následném zpracování Drobný problém nastává v případech, kdy skoková změna přechodového jevu trvá delší dobu. To způsobí několikanásobnou detekci jednoho přechodového děje. Nicméně pro prvotní analýzu měřených dat tento nedostatek nevadí. Dalším problémem je současné zapnutí více spotřebičů. S řešením těchto situací se momentálně nepočítá.

2.1.2 Zpracování přechodových jevů

Při zpracování detekovaných přechodových jevů je možné využít všech měřených výkonů. Ze samotného průběhu zdánlivého výkonu v okolí zapnutí či vypnutí spotřebiče není možné s jistotou určit jeho typ. K tomu by měly pomoci zbylé měřené výkony. Ty je možné spolu se zdánlivým výkonem sledovat určitý čas před i po přechodovém jevu. Je tak možné určit jejich rozdíl na začátku a konci sledovaného časového úseku nebo maximální a minimální hodnotu z celého analyzovaného okolí přechodového jevu.

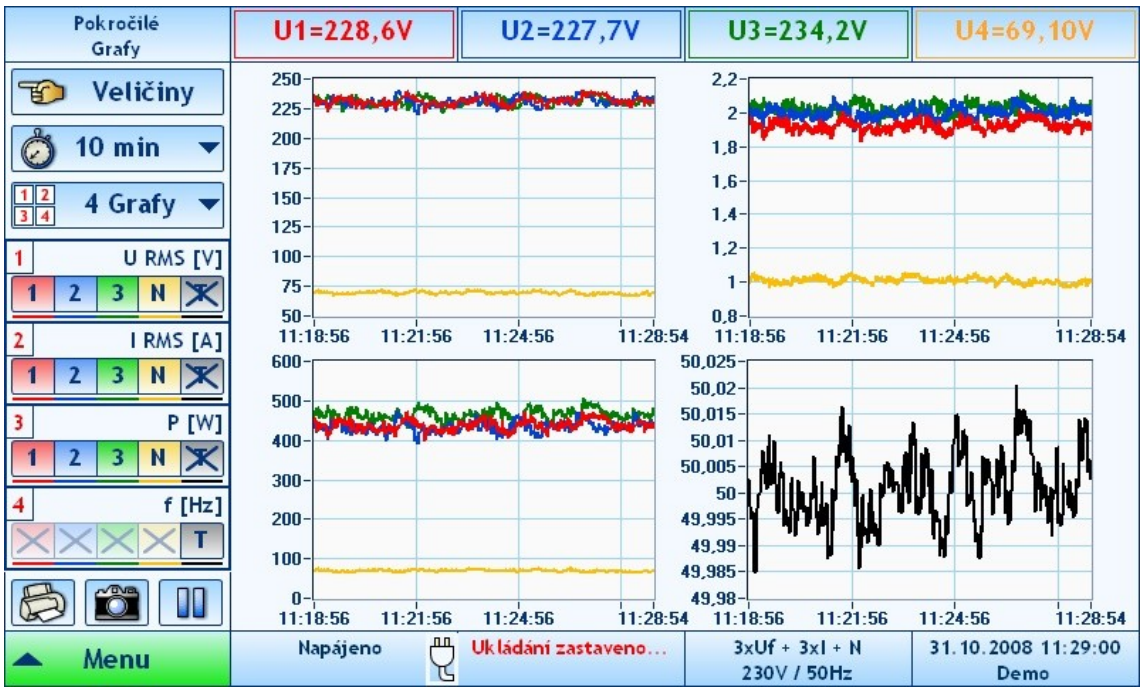
Navzájem tvoří různé druhy výkonů n-rozměrný prostor. Příkladem jedné roviny je závislost deformačního výkonu na zdánlivém výkonu. Předpokládá se, že stejné spotřebiče by měly v n-rozměrném prostoru tvořit určité shluky, což by mělo ulehčit rozpoznávání elektrospotřebičů.

Příklad zapnutí elektrospotřebiče je na obr. 2.1. Spolu se skokovou změnou zdánlivého výkonu (tmavě červený průběh) je vidět výrazná špička u deformačního výkonu (černý průběh). Jalový výkon (zelený průběh) oproti zdánlivému po zapnutí spotřebiče poklesl. Důležité jsou hodnoty výkonů před i po zapnutí. Hodnoty po zapnutí je nutné měřit s větším časovým odstupem, protože se musí počkat na ustálení přechodových dějů.

2.2 Software pro vizualizaci dat

2.2.1 Elcom

Vizualizační software od společnosti Elcom je charakteristický svým velice jednoduchým, intuitivním a uživatelsky přívětivým ovládáním. Nabídka typů grafů je dostatečná. Samozřejmostí je klasický spojitý graf, který umožňuje dva druhy zobrazení. Dále program umí vykreslovat fázorový diagram, časový průběh signálu a spektrum signálu. Na druhou stranu chybí pokročilejší funkce, jak grafy nastavovat nebo ovládat. Svým charakterem je tak tento program zaměřen na zákazníky, kteří nepožadují při měření profesionální výsledky, ale potřebují rychlou, snadnou a názornou interpretaci naměřených hodnot.



Obr. 2.2: Software Elcom

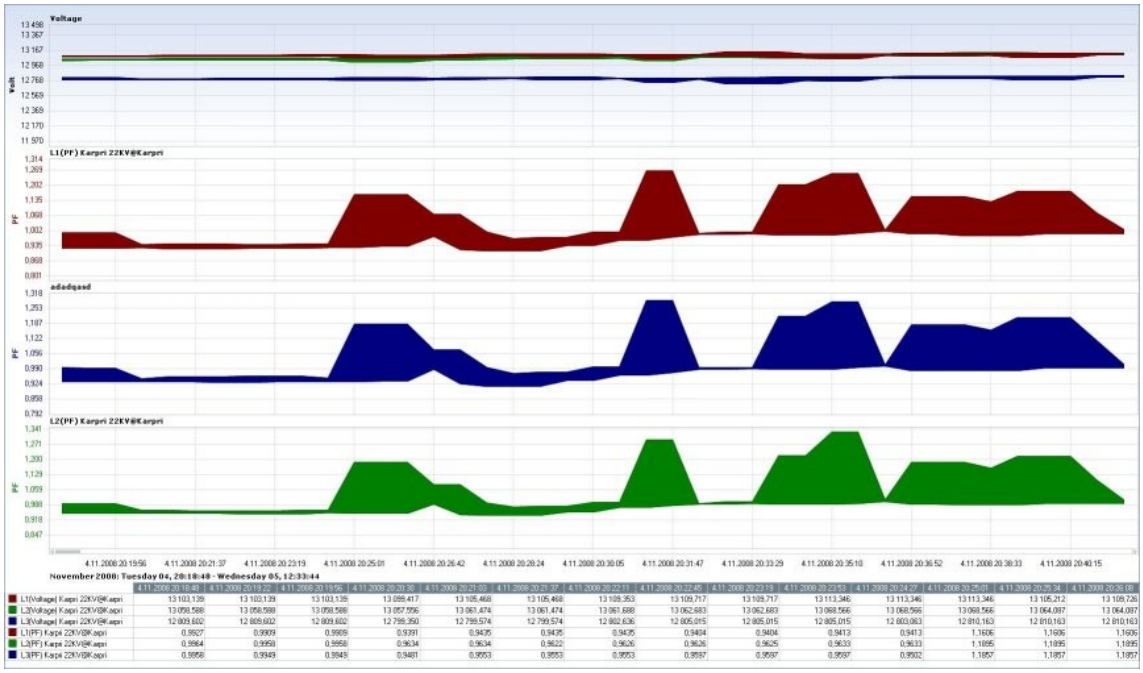
Grafy

Základní graf je možné zobrazit jednak v normálním nebo pokročilém módu. Základní mód umožňuje zobrazit jen jednu veličinu a její průběh v každé fázi v jednom společném grafu. Na výběr je napětí, proud, výkon nebo frekvence. Oproti tomu lze v pokročilém módu zobrazit všechny veličiny společně, a to buď v jednom velkém grafu, nebo v několika malých. V obou případech lze rozsah osy X měnit ručně. Na výběr jsou tři možnosti: jedna minuta, deset minut a jedna hodina. Naproti tomu se osa Y mění plně automaticky podle právě zobrazených průběhů. Na žádné ose však program neumožňuje ani posuv, ani zvětšení, což je alespoň v případě pokročilého zobrazení škoda.

Zobrazení vektorů umožňuje sledování fázorových diagramů napětí a proudu ve třech fázích pro prvních padesát harmonických. Na výběr je společné nebo oddělené zobrazení každé z veličin. Důležité údaje, jako třeba fázový rozdíl mezi napětím a proudem, jsou zobrazeny v tabulce vedle fázorového diagramu. Zobrazení časového průběhu je podobné zobrazení na osciloskopu. Opět lze zobrazit všechny tři fáze, ale není možno nastavovat časovou základnu. Na rozdíl od ostatních grafů jsou v tomto zobrazení osy přehledně popsané. Posledním typem grafu je zobrazení spektra signálu. Lze ho zobrazit pro každou měřenou veličinu v každé fázi dohromady, nebo jednotlivě. Maximální počet spočtených harmonických je padesát.

2.2.2 ELSPEC Investigator 2.0

Software ELSPEC Investigator 2.0 umí zobrazit daleko více informací než software Elcom. Také umožňuje mnohem detailnější nastavení a ovládání grafů. Má však složitější a méně intuitivní ovládání.



Obr. 2.3: Graf v ELSPEC Investigator 2.0

Graf

Graf má přehledně popsané osy, kde osa X je v časovém formátu a na ose Y je zobrazovaná veličina. Samotný graf je interaktivní, což znamená, že po najetí kurzorem na určitou část průběhu se zobrazí různé informace o tomto úseku měření jak v grafu, tak v tabulce pod ním. Myší lze najet i na pozadí grafu a v tomto případě se zobrazí informace o měřené veličině a měřícím přístroji. Graf umožňuje plynulý posuv časové osy, který se realizuje pomocí kolečka myši. Pod pravým tlačítkem jsou uschovaná rozšířená nastavení.

Funkce programu

Dobré zobrazení grafu a možnost z něho vyčíst nejrůznější informace však nejsou jediná kritéria, podle čeho se dá posoudit daný software. Druhým a neméně důležitým faktorem je ovládání programu. ELSPEC Investigator 2.0 má širokou nabídku ovládacích prvků. Prvním z nich je uložení nebo načtení nastavení zobrazování grafů. Zobrazování grafu lze pomocí několika nástrojů přizpůsobit ke své potřebě. Aby

se pokaždé nemuselo všechno nastavovat znovu, lze si uložit pomocí této funkce aktuální nastavení. Při další práci stačí nastavené hodnoty pouze načíst. Funkce rovněž obsahuje možnost vrátit se k základnímu nastavení. Další velmi užitečnou funkcí je exportování dat. Graf lze tak uložit do schránky k následnému použití v jiných programech. Jsou na výběr tři možnosti: bitmapa, metafile, text. Do schránky se ukládá pouze graf, a pokud je zobrazená legenda a tabulky, tak i tyto části.

Další funkce umožňuje výběr typu grafu. Lze volit mezi spojitým a sloupcovým grafem. Spojitý graf je však trochu odlišný od standardního spojitého grafu, protože se nevykresluje jako spojitá křivka, ale jako plocha. Dále je možné volit mezi 2D a 3D grafem. U 2D grafu je nevýhoda, pokud se zobrazuje více průběhů společně, tak se plochy mohou překrývat a graf je nepřehledný. 3D graf se snaží daný nedostatek odstranit, ale podle mého názoru k výraznému zlepšení nedochází. Navíc umožňuje rotování v prostoru po předem daných krocích. Dále software umí zobrazit číselné hodnoty nad průběh grafu. Nevýhodou ale je, že pokud je zobrazené časové rozmezí široké, tak se čísla překrývají a jsou z velké části nečitelná.

Zbývají ještě funkce pro zoomování grafu. Zoomovat lze dvěma způsoby. Jednak v časové oblasti, nebo přímo určitou část signálu. Zoom v časové oblasti se ovládá poměrně snadno a intuitivně. Zoom části signálu se ovládá následovně. Kolem části průběhu se myší vytvoří obdélník. Tento obdélník se však vytváří naprosto nelogickým způsobem, nikoli od rohu do rohu, ale od středu do rohu. Vymezit tedy přesný obdélník vyžaduje cvik. Nejhorší situace nastává s odzoomováním. Neexistuje možnost k navrácení na původní velikost. Ani pomocí myši odzoomování neexistje. Jediný způsob je přes menu, kde lze vybrat odzoomování v předem daném rozmezí.

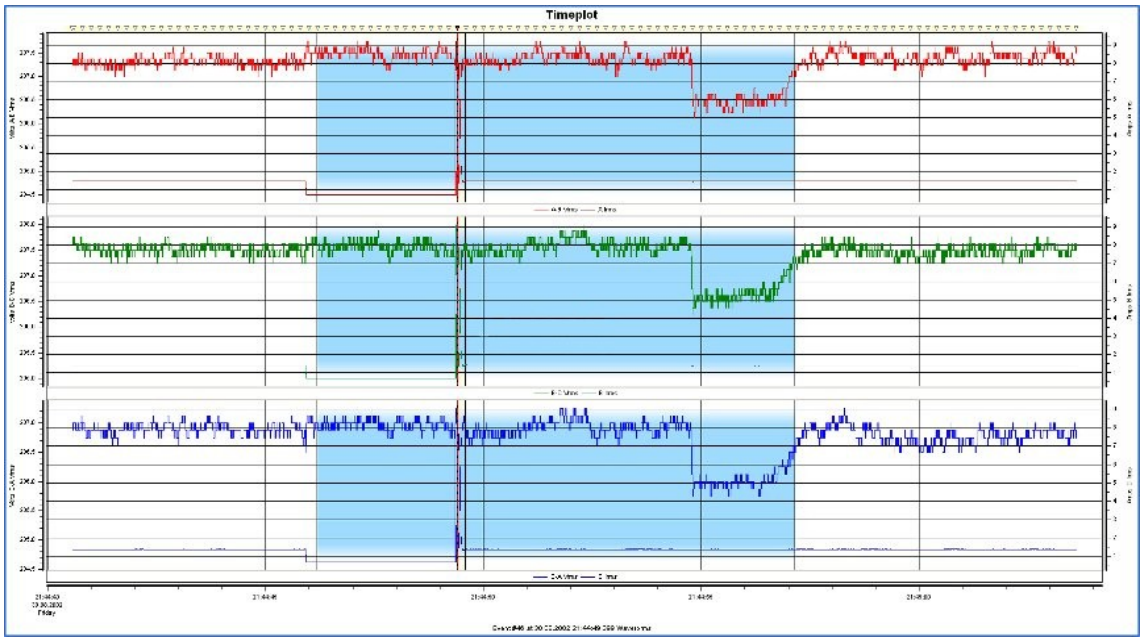
S poslední funkcí, která umožňuje měření času, se pracuje dobře. Je lehce použitelná a intuitivní. Klikne se myší na místo v grafu, odkud se má měřit čas, a táhne se na konec požadovaného časového úseku. Následně se ukáže popisek s datem a časem začátku a konce vybraného úseku a navíc délka jeho trvání.

Zhodnocení

Software ELSPEC Investigator 2.0 je rozporuplný program. Na jednu stranu má bezpočet možností, ale na druhou stranu poskytuje horší ovládání. Pro zkušenější uživatele, používající tento software, nemusí být složitější ovládání problém. Program tak lze považovat za plnohodnotnou aplikaci zobrazující průběhy naměřených veličin.

2.2.3 Trinergi Dran-View 6

Jako nejprofesionálnější software, který jsem testoval, se mi jevil program Trinergi Dran-View 6. Ten lze spustit v režimu Enterprise nebo Professional. Obě verze se liší pouze počtem grafů, které lze zobrazit. Obě verze mají velké množství funkcí k ovládání grafů a získávání potřebných informací. Základní funkce jsou jednoduché, ale pro zbývající je vhodné použít návod.



Obr. 2.4: Graf v Trinergi Dran-View 6

Graf

Po dvojkliku na graf se otevřou jeho vlastnosti, kterých je opravdu nepřeberné množství, od typu a barvy čáry, typu grafu, mřížky, po nastavení kurzorů. Samotný graf má přehledně popsané obě osy a je interaktivní. Po najetí myší na plochu grafu se po chvíli objeví čas a hodnota v místě kurzoru. Po kliknutí levým tlačítkem na průběh se daná křivka zvýrazní. Toho se dá využít při zobrazování více průběhů v jednom grafu. Program také umožňuje exportovat graf ve formě obrázku.

Funkce

První funkcí pro ovládání grafu je zvětšení respektive zmenšení popisek u grafu. Ta umožňuje přepínání mezi dvěma velikostmi popisek. Menší velikost je vhodná nejspíše do tištěných verzí, větší popisky si umím představit například v prezentaci, kde by bylo malé písmo špatně čitelné.

Následuje funkce pro zoomování grafu. Pro tuto činnost jsou k dispozici dva nástroje. Snadnější je zoomování pomocí obdélníku, který se vytváří myší. Druhou možností je za stálého držení levého tlačítka pohybovat kurzorem do stran a podle toho se graf buď zvětšuje nebo zmenšuje. U tohoto způsobu však průběh grafu nepříjemně ujíždí do stran. Na druhou stranu je zvětšování plynulé. Odzoomování je zde provedené velice dobře. Opět jsou na výběr dvě možnosti. Pokud je potřeba zmenšovat graf po krocích, pouze se kliká na příslušné tlačítko. Druhou možností je okamžitý návrat na původní velikost. Spolu s možností zvětšovat nebo zmenšovat graf souvisí i možnost jeho posunu. Posun je v Trinergi Dran-View 6 vytvořen velmi intuitivně. Stačí uchopit graf pomocí levého tlačítka na myši a posunovat ho požadovaným směrem. Jako malý nedostatek bych viděl nekvalitní optimalizaci, protože při posunu průběh problikává.

Po zvětšení části průběhu se může stát, že osa Y nebude využita ve své maximální šířce. Proto je zde funkce Autoscale. Ta po svém spuštění nastaví rozsah Y osy tak, aby byla využita celá. Není však možné zapnout tuto funkci trvale, aby měnila rozsah osy Y po každém zvětšení, zmenšení nebo posuvu. Podobná funkce, ale pro časovou osu X, se jmenuje Autozoom. Nahoře nad grafem je možné pomocí malé nevýrazné šipky vybrat začátek krátkého časového úseku. Poté po kliknutí na Autozoom se zobrazí pouze tento krátký časový úsek.

Další dvě funkce jsou určeny pro vybírání částí grafu. První z nich je označení úseku. Pomocí této funkce je možné zvýraznit pozadí za částí průběhu. Zvýrazní se celý rozsah na ose Y a osu X si volí uživatel. Po zvýraznění se nezobrazí žádné údaje a i způsob, jakým je vybraná část grafu zvýrazněna, napovídá, že toto bude další funkce vhodná pro použití při prezentacích. Naproti tomu funkce kurzorů je určena na zjišťování hodnot z průběhů. Kurzory jsou řešeny velice pěkně. Myší se natáhne odkud kam chceme měřit hodnoty v grafu. Vytvoří se dvě kóty, jedna vodorovná a jedna svislá. Na těchto kótách se zobrazí rozdíl hodnot mezi jejich konci. Malým nedostatkem je, že se nezobrazují absolutní naměřené hodnoty.

Dále je možné pomocí několika tlačítek nastavit zobrazení průběhů naměřených veličin do jednoho či více grafů. V programu jsou odlišná tlačítka na rozdělení různých veličin do více grafů a rozdělení jednotlivých fází do tří samostatných grafů. V tomto případě však software neumožňuje zobrazit např. první a třetí fázi dohromady a druhou fázi samostatně.

Ve verzi Enterprise zbývají už jenom dvě funkce. První z nich umožňuje zobrazit tabulku s naměřenými hodnotami. Tabulka je řešena tím způsobem, že se otevře v nové záložce. Poslední funkce ukládá grafy jako reporty. Verze Professional obsahuje všechny výše uvedené funkce a navíc přidává tabulku se statistikou, fázorový diagram a zobrazení harmonických.

Informace o datech

Tabulka se statistikou zobrazuje maximální, minimální a průměrné hodnoty z grafů. Po dvojkliku se otevře nastavení, kde lze přidat statistické údaje, které chceme zobrazovat. Navíc je možné nastavit další detaily jako barvy nebo font. Fázorový diagram zahrnuje i hodnoty úhlů jednotlivých fázorů. Vše se zobrazí jako poměrně malé okno, které je možné zvětšit. Po dvojkliku se otevře nastavení, kde lze vybírat další zobrazované fázory, jejich barvy apod. S grafem harmonických se zobrazují i hodnoty RMS, FND, DC a THD. Samozřejmostí je volba veličin, u kterých požadujeme zobrazení harmonických.

2.3 Komponenty pro zobrazování grafů

Hlavním úkolem mnou vyvíjeného programu je vykreslování grafů. Proto jsem musel vybrat vhodnou komponentu pro C#, která umí grafy vykreslovat. První kritérium, podle kterého jsem vybíral, byl rozdíl mezi volně šiřitelnou, nebo komerční knihovnou. Mezi open source knihovnami jasně dominuje knihovna ZedGraph, která, až na tvorbu 3D grafů, umí snad vše, co může programátor potřebovat. Z komerčních komponent se nabízí možnost použít DevExpress XtraCharts, nebo Steema Teechart. DevExpress XtraCharts má tu výhodu, že umí vykreslovat i 3D grafy pomocí OpenGL, a Steema Teechart je dostupná i v nekomerční, zato dost omezené verzi. Po zvážení všech kladů a záporů jsem si vybral komponentu ZedGraph. Největší výhodou je široká uživatelská základna, takže k většině otázek lze na internetu najít odpověď.

2.3.1 Všeobecné informace

ZedGraph je knihovna napsaná v programovacím jazyce C#, avšak lze ji používat i v programovacím jazyce C++ nebo Visual Basic. Tato komponenta je určena pro vykreslování grafů a díky své velké volnosti nastavení všech vlastností je vytváření grafů rychlé a snadné. Jedním z nedostatků ZedGraphu je, že neumožňuje tvorbu 3D grafů, proto si při jeho používání musíme vystačit s dvojrozměrným prostorem. Přesto

nabízí velkou škálu typů grafů, od klasických spojitých grafů, přes vertikální a horizontální sloupcové, koláčové, až po speciální grafy, které kombinují všechny předchozí typy. Mohou vypadat jako vrstevnice na mapě, nebo mít podobu analogového měřícího přístroje.

Licence LGPL

ZedGraph je volně šiřitelná knihovna, která spadá pod licenci LGPL (Lesser General Public License). Podle [6] tuto licenci sepsal v roce 1991 Richard Stallman a poslední změny pocházejí z roku 2007. Knihovnu s LGPL licencí lze použít i v aplikacích, které nemají LGPL licenci. Mohou to tedy být programy jak volně šiřitelné nebo komerční. Ty jen za předpokladu, že software není odvozeným dílem dané knihovny. Samotný spustitelný program, který dynamicky linkuje knihovnu, se považuje za dílo, které není odvozené, a proto nespadá pod licenci LGPL.

Historie

Následující informace jsou převzaty a volně přeloženy z [10]. Historie ZedGraphu sahá do listopadu roku 2003, kdy vyšla verze 1.0. Tato verze se vyvíjela do července 2004 a skončila u čísla 1.5. Jako příklad vývoje bych uvedl možnost používání časové osy nebo sloupcového grafu. 1. září 2004 se objevila verze 2.0, u které bylo výraznou změnou ukládání hodnot bodů do PointPairu a celého průběhu do PointPairListu. Jen o dvacet dní později vyšla další verze s pořadovým číslem 3.0. Ta přidala možnost vybarvování ploch plynulými přechody. V prosinci roku 2004 se dočkala vylepšení na verzi 3.5. Zde přibyla například třída okrajů nebo formát času XDate. V březnu 2005 byla vydána verze 4.0 a vyvíjela se až do ledna 2006. Jednou z výrazných změn bylo přidání událostí od tlačítek na myši. V současnosti existuje verze 5.0 určená pro .NET 2.0, ale ve vývoji verze 4 se pokračuje, a ta je určená pro .NET 1.1.

2.3.2 Použití

Používání knihovny ZedGraph není nikterak obtížné. Do projektu stačí přidat referenci na příslušnou .dll knihovnu a může se začít používat. Při vytváření WinForm aplikace se objeví ve vývojovém prostředí nová komponenta ZedGraphControl, přes kterou se ovládají všechny grafy a jejich nastavení. Hlavní plocha, na kterou se umisťují všechny grafy, se jmenuje MasterPane. Ta se nemusí používat v případě, že je zobrazován jen jeden graf. Při nutnosti zobrazit více grafů najednou se bez MasterPanu

neobejdeme. Samotné grafy se vykreslují na GraphPane. Nejedná se však jen o plochu mezi osami X a Y, ale tato plocha zahrnuje i všechny osy, včetně jejich popisků. Zobrazovaná data jsou uložena v PointPairListu, což není nic jiného, než seznam souřadnic X a Y. Pro jeho zobrazení se z něj musí vytvořit vhodný průběh. V případě spojitého grafu se jedná o LineItem a v případě sloupcového grafu o BarItem. Mezi jejich základní nastavení patří barva a šířka čáry a typ zobrazených symbolů. Samozřejmostí je přehledná legenda zobrazených průběhů v jednotlivých GraphPanech nebo všech grafů v MasterPanu.

Další možnosti

ZedGraph nabízí i jiné možnosti, než jen pouhé vykreslování grafů. Jednou z nich je přidávání textových popisků. Ty lze umisťovat k samotným průběhům, nebo jako dodatečnou informaci libovolně do plochy. Také u popisků je velká škála nastavení. Lze měnit font, barvu, velikost, průhlednost nebo úhel natočení. Stejně jako popisky, lze do grafu přidat i barevné Boxy, které mají tvar obdélníku. Použití najdou například pro zvýraznění určité části průběhu, nebo je použít jako podklad pro přidané popisky. Velikou výhodou je, že je možné je pozicovat v závislosti na hodnotách na ose, tak v závislosti na ploše grafu. Dokonce lze tyto metody kombinovat a pro každou osu lze použít jiný způsob.

Osy

Nedílnou součástí každého grafu jsou osy. ZedGraph nabízí hodně typů os. Samozřejmostí je lineární, logaritmická a časová osa. Dále je na výběr textová, exponenciální nebo ordinální osa. Základem je jedna osa X a jedna osa Y. Při zobrazování více průběhů v jednom grafu však může nastat situace, že jedna osa Y nestačí. I na toto ZedGraph pamatuje. Stačí ji pouze zviditelnit a přiřadit k ní patřičný průběh. V případě, že ani dvě Y osy nestačí, lze přidat další, avšak postup už je o něco komplikovanější. Základní nastavení všech os provádí ZedGraph sám v automatickém režimu. Automatický mód lze však vypnout a otevírá se značné množství možností, jak si danou osu přizpůsobit. Mezi nejpoužívanější úpravy bych zařadil ocejchování os a zobrazení mřížky na pozadí grafu. Nejlepší a nejpoužívanější je funkce, která umí nastavit obě osy přesně tak, aby byl viditelný celý zobrazovaný průběh.

2.4 Přístup k datům

Databáze, ze které vyvíjený software načítá data, je relačního typu. Naopak programovací jazyk C# je objektově orientovaný. Nabízí se tak tři způsoby přístupu k datům. Podle [11] je možné vymodelovat databázi v objektově orientovaném programu, nebo vymodelovat aplikaci v databázi, nebo zjednodušit přístup k databázi. Je tedy možné oba způsoby zkombinovat a použít tak objektového přístupu k databázi. K takovému přístupu jsou určeny ADO.NET Entity Framework, nebo DevExpress XPO. Oba typy mají své klady a zápory. Pro ADO.NET Entity Framework mluví fakt, že je rychlejší při práci s velkými databázemi, ale nevýhodou je rychlost prvního načítání. Rychlost DevExpress XPO je oproti tomu přímo úměrná velikosti databáze, ale pokud přesáhne rozsah databáze určitou mez, doba načítání se neadekvátně prodlužuje. Ve své aplikaci jsem použil ADO.NET Entity Framework, protože v KMB systems již s DevExpress XPO pracují, a tak jim můj program poslouží i jako srovnání mezi oběma způsoby přístupu k datům.

2.4.1 Všeobecné informace

Julia Lerman [8] uvádí, že ADO.NET Entity Framework je nový způsob přístupu k datům uložených v databázi při vyvíjení .NET aplikací. Vznik Entity Frameworku si vyžádali sami vývojáři. Ti několik let požadovali změnu přístupu k datům v ADO. Místo původního DAO (Data Access Object) požadovali RDO (Remote Data Objects). První zmínku o Entity Frameworku vypustil Microsoft do světa na konferenci TechEd 2006. Vydán byl o dva roky později, a to v červenci 2008. Byla to součást Visual Studio Service Packu 1 a. NET 3.5 Service Packu 1. Díky Entity Frameworku se při vytváření aplikací mohou vývojáři více zabývat vlastním programem, namísto řešení problémů souvisejících s přístupem k datům. Microsoft mluví o Entity Frameworku, jako o budoucím základním způsobu práce s daty.

Co nového přináší EF

Julia Lerman [8] dále píše, že revoluční je Entity Framework, protože vývojáři nevytváří program přímo vůči databázi, ale pracují s jejím modelem. Tento model se nazývá Entity Data Model (EDM), který byl vyvinut na základě Entity Relationship Modelingu (ERM). ERM definuje schéma entit a jejich vztahy s ostatními entitami. Entity však nejsou to samé co objekt. Entity tvoří v podstatě schéma tabulek v databázi. ERM se roky používalo k transponování strukturovaných tabulek databáze na strukturu

objektů. K získání dat z databáze tak není nutné tyto hodnoty převádět na objekty, protože Entity Framework pracující s Entity Data Modelem je již jako objekty vrací. Při ukládání změn zpět do databáze stačí pouze uložit tyto objekty. Entity Framework udělá nezbytný překlad z objektu na hodnotu v určitém řádku a sloupci databáze sám, takže programátorům ušetří mnoho práce.

Entity Data Model

Entity Data Model je spojnicí mezi daty a aplikací. Při každém čtení či ukládání dat využije program k této činnosti EDM. Právě ze zmíněného modelu vytváří Entity Framework třídy, díky nimž je možné pracovat s daty jako s objekty. Naopak EDM vytváříme my z databáze. Není podmínkou, že EDM mapuje celou databázi. Naopak může být tvořen jedinou tabulkou z rozsáhlé databáze. Existuje tak možnost vytvořit více různých modelů z jedné databáze. Výhodou toho, že do modelu zahrneme jen tabulky, které budou v dané aplikaci potřeba, je možná úspora paměti a zrychlení programu.

Samotný EDM jde rozdělit na tři části: koncepční model, mapovací model a úložný model. Koncepční model je samotná definice entit a je uložen jako XML dokument. Úložný model definuje, jak jsou data uložena. Umožňuje definovat jak jednoduché, tak komplexní rozvržení tabulek a polí. Uložen je také jako XML dokument. Prostředníkem mezi oběma modely je mapovací schéma, které oba modely navzájem propojuje.

2.4.2 Použití

Aby bylo možné Entity Framework vůbec používat, je nutné mít nainstalovaný .NET Framework 3.5 Service Pack 1 a Visual Studio 2008 Service Pack 1. Do projektu ve Visual Studiu se přidá nová položka ADO.NET Entity Data Model. Poté je možné vybrat, jakým způsobem bude EDM vytvořen. Nejpoužívanější je vygenerování modelu přímo z databáze. Vyberou se všechny tabulky a pohledy, které mají být v dané aplikaci používány. Při psaní samotné aplikace není rozdíl v používání tabulek nebo pohledů. Po skončení výběru tabulek do modelu je již EDM vytvořen a objeví se graficky znázorněn. V dané grafice jsou znázorněny jednotlivé tabulky, jejich vzájemné vazby i jednotlivé sloupce. V případě tohoto postupu není vhodné do modelu nijak zasahovat a měnit jeho parametry, protože tyto změny se neprojeví v samotné databázi, model jí tak přestane odpovídat a důsledkem bude nefunkčnost programu.

Možné komplikace

Právě s rychlostí načítání dat z databáze má Entity Framework drobný problém. Ten spočívá v tom, že se zvětšujícím se EDM neúměrně rychle roste doba načtení požadovaných dat. Spolu s tím se i neadekvátně prodlužuje doba kompilace programu, takže pro složitější modely je téměř nemožné program odladit, protože každý překlad trvá řádově několik minut. Tyto dva nedostatky dokáže naštěstí odstranit volně šiřitelný program EdmGen2RL.

3 Realizace programu

3.1 Vizualizace

Vizualizační stránka vyvíjené aplikace je jedna z hlavních náplní mé bakalářské práce. Proto jsem při vyvíjení zadaného programu dbal na přehlednost a názornost. Jsem zastáncem toho, že pokud aplikace nezaujme na první pohled svým vzhledem, uživatel ji mnohdy přejde bez povšimnutí, i kdyby byla po programové stránce dokonalá.

3.1.1 Základní graf

Způsob vykreslení průběhů

Aby bylo vůbec možné nějaké grafy vykreslovat, je zapotřebí vložit do aplikace komponentu ZedGraphControl. Ta může být na každém Formu pouze jednou a stará se o kompletní správu všech vytvářených grafů. Jako první, o co se musí postarat, je vytvoření ploch, na které se budou průběhy vykreslovat. Hlavní plocha se nazývá MasterPane. Umisťují se na ní jednotlivé GraphPany, což jsou právě plochy, na kterých se zobrazují jednotlivé průběhy. Výhodou tohoto konceptu je fakt, že lze vytvořit velké množství GraphPanů s různými průběhy a poté je jen přidávat nebo odebírat z MasterPanu bez nutnosti cokoli přenastavovat.

GraphPane je skutečná plocha, na kterou se vykreslují průběhy. Umožňuje nespočet nastavení. Mezi nejpoužívanější určitě patří nastavení os, mřížky, legendy, nadpisu apod. Postupem času jsem se dopracoval k podrobným nastavením, díky kterým lze výsledné zobrazení vyladit do posledního detailu. Jako příklad bych jmenoval možnost zneviditelnit první nebo poslední popisek na některé z os. To se hodí v případech, kdy má graf nastavené malé okraje a právě krajní popisky by přesahovaly ven.

Vykreslování průběhů na již vytvořené GraphPany lze rozdělit do dvou kroků. Nejprve se musí data uložit do PointPairListu, který je součástí knihovny ZedGraph. Jedná se o dynamické pole se sloupci X, Y, Z. Práce s ním je stejná jako s běžnými dynamickými poli. Následně se z PointPairListu vytvoří křivka. Při jejím vytváření se musí zadat, na kterém GraphPanu bude umístěna, její jméno, jakou bude mít barvu a z jakého PointPairListu bude vykreslena. Lze určit i další parametry křivky, ale není to podmínkou. Já osobně hned po vytvoření dané křivky nastavím její tloušťku a symboly.

Změny na vykreslené křivce se dají realizovat opravdu snadno. Každá křivka je pevně provázána s jedním PointPairListem, takže stačí změnit hodnoty v něm a křivka se automaticky překreslí. Osobně toto využívám ve svém programu tak, že po startu programu přiřadím všem křivkám předem připravené PointPairListy a následně je plním daty. Nastavení veškeré grafiky související s vykreslováním průběhů tak proběhne na začátku v jediném kroku.

V mé aplikaci do základního grafu vykresluji průběhy výkonů P, S, Q, D. Velikost hodnot P, S jsou podobné, stejně jako hodnoty Q, D. Navzájem jsou tyto dvojice svými hodnotami dost vzdálené, proto při zobrazování všech průběhů na jednu osu Y byly průběhy Q a D vykresleny téměř jako rovné vodorovné čáry. Bylo tedy nutné vytvořit pro ně druhou samostatnou osu Y. Se dvěma osami Y ZedGraph umí pracovat a stačilo tedy přiřadit zmíněné průběhy k druhé ose.

Časová osa a její posun

ZedGraph umožňuje vybírat z několika typů os. Protože v mém programu zobrazuji hodnoty naměřené v časové posloupnosti, používám pro většinu grafů časovou osu. Způsob použití je obdobný jako u klasické lineární osy, liší se ale formát hodnot. Časová osa využívá strukturu XDate, která určuje ve formátu double počet dní od 30. prosince 1899. Vytvářet ji však lze způsobem rok, měsíc, den, hodina, minuta, sekunda, milisekunda a z těchto údajů se sama převede na typ double. Výhodou XDate je, že dokáže převést např. datum 32. března na 1. dubna, což není u formátu DateTime snadné.

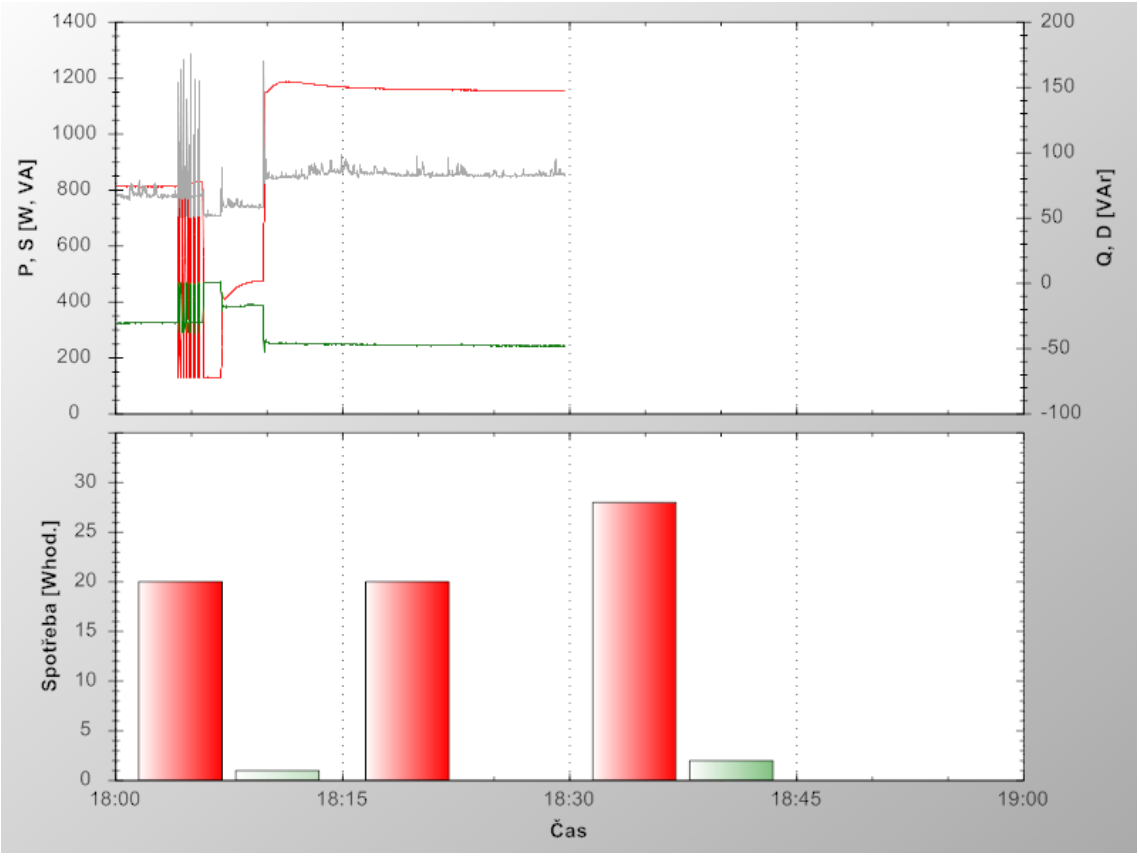
Po vykreslení grafu umí ZedGraph sám nastavit osy tak, aby byl zobrazen celý průběh. Já však měl za úkol udělat přesně definované časové intervaly, ve kterých by se zobrazovala vždy jen část grafu a po kterých by se s grafem posunovalo. Z několika měření vyšly nejlépe tyto časové úseky: týden, den, hodina a pět minut. V tomto případě už není možné používat automatický rozsah obou os, proto jsem vytvořil funkci, která výše zmíněné požadavky zajišťuje. Jedná se především o rozpoznání, zda uživatel požaduje posun po časové ose vpřed, nebo vzad a jaký je vybraný časový interval. Z těchto údajů se vypočítají konkrétní hodnoty pro začátek a konec časové osy. Dále se nastaví vhodné intervaly na ose.

Výpočet začátku a konce časové osy probíhá následujícím způsobem. Z hodnoty double, která určuje právě zobrazený začátek na ose X, se zpětně získají hodnoty rok,

měsíc, den, hodina, minuta, sekunda, milisekunda. Následně se vytvoří dvě hodnoty typu XDate pro nový začátek a konec. Ty se použijí pro nastavení nově zobrazené časové osy. Na konci se zkontroluje, zda je možné dále graf posunout dopředu či dozadu, a případně se zablokuje příslušné tlačítko, takže není možné zobrazit prázdné časové úseky. Vždy tak bude zobrazena alespoň část průběhu.

Synchronizace s grafem elektroměru

S postupem času přibyl požadavek, aby moje aplikace nezobrazovala jen jeden graf a v něm průběhy elektrických výkonů, ale aby přibyl ještě jeden graf zobrazující hodnoty z elektroměru. O grafu samotném je psáno dále. Na tomto místě bych se rád zmínil o tom, jak nový graf ovlivnil ten původní. Oba grafy mají časovou osu a oba zobrazované průběhy jsou ve stejných časových intervalech. Naskytlo se tak řešení umístit oba grafy pod sebe a vytvořit jim jednu společnou časovou osu.



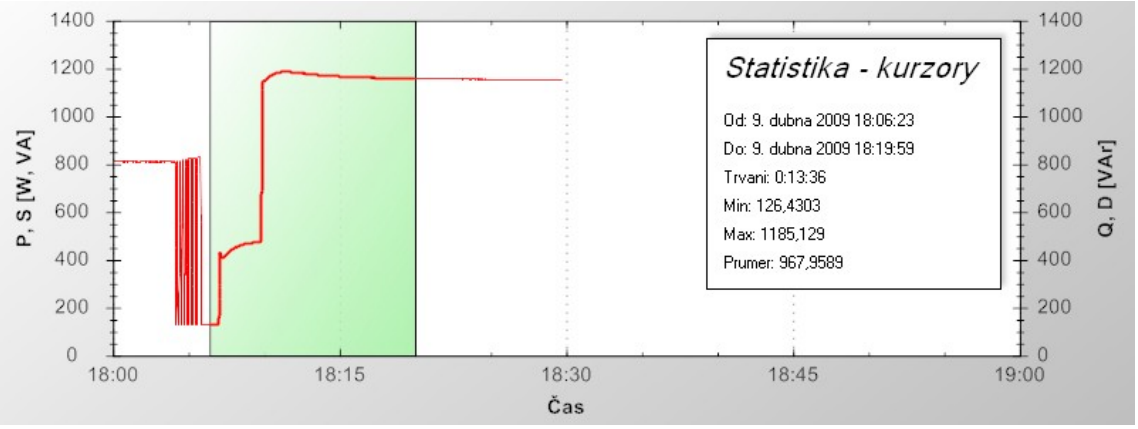
Obr. 3.1: Společná osa X pro dva grafy

Nejdříve jsem schoval popisky a název časové osy u horního grafu. Dále doplnil výpočet rozsahu časové osy i pro druhý graf. Nyní sice měly oba grafy stejný rozsah osy X, ale nebyly zobrazené přesně pod sebou. Graf pro zobrazení jednotlivých výkonů

má totiž i sekundární osu Y a je tím pádem užší. Naštěstí ZedGraph umožňuje nastavit minimální vzdálenost osy od okraje a po nastavení stejných hodnot pro oba grafy se tak tento problém vyřešil.

Kurzory

První požadovanou interakcí s grafem byly kurzory. Pomocí nich se vybere určitá část průběhu a k ní se zobrazí základní informace: začátek a konec vybraného úseku, jeho trvání, maximální, minimální hodnota a průměr. Aby bylo ovládání co nejvíce uživatelsky přívětivé, rozhodl jsem se udělat kurzory způsobem „táhni a pusť“. K tomu bylo zapotřebí zajistit určení pozice kurzoru na grafu a navrhnout vhodný způsob zobrazení kurzorů.



Obr. 3.2: Ukázka kurzorů a vypisovaných informací

S určením pozice kurzoru, který se nachází nad plochou grafu, mi opět pomohl ZedGraph, který je na něco podobného připraven. Pozici dokonce nevrací v pixelech, ale přímo jako hodnoty X, Y z grafu. Díky tomu není nutné nic přepočítávat, ale stačí si uložit počáteční a konečnou pozici kurzoru a následně provést příslušné matematické operace s vybraným úsekem. Počátek kurzorů se uloží při stisknutí levého tlačítka myši. Následně při jeho držení se neustále mění hodnota konce kurzorů. Přestane se měnit po uvolnění levého tlačítka. V případě, že se kurzory vyberou způsobem zprava doleva, nastává problém s tím, že konečný kurzor má menší hodnotu než počáteční. Je tedy nutné, před předáním těchto hodnot další funkci, obě proměnné zaměnit.

Grafické znázornění vybraného úseku jsem vyřešil pomocí barevného, poloprůhledného obdélníku, zobrazeným za vykreslenými průběhy. Jedná se objekt BoxObj, který se po nastavení vloží přímo na některý GraphPane. Velikou výhodou

tohoto objektu je fakt, že se jeho rozměry dají určit jednak podle měřítka jednotlivých os, nebo procentuálně. Já jsem tyto dva způsoby zkombinoval. Velikost obdélníku určuji na ose X přímo hodnotami XDate a na ose Y využívám druhý způsob. Obdélník je totiž vykreslen stále přes celý rozsah osy Y. Kurzory označující vybranou oblast jsou ještě doplněny zvýrazněním průběhu tlustou čarou.

3.1.2 Ostatní grafy

Graf elektroměru

Vedle hlavního grafu je v mé aplikaci ještě několik dalších doplňkových grafů, mezi kterými lze přepínat. Jedním z nich je graf pro zobrazení průběhu z elektroměru. Tento graf má časovou osu synchronizovanou s hlavním grafem, o čemž je psáno výše. Zobrazení hodnot z elektroměru je realizováno pomocí sloupcového grafu. Sloupce jsou dva, respektive tři, s tím, že jeden je samostatně a zbývající dva sloupce jsou umístěny nad sebou. Zobrazuje se tak průběh tří hodnot.

Hodnoty z elektroměru se ukládají do databáze každých patnáct minut, takže zobrazení je vhodné provést po čtvrt hodinách. Dále byl požadavek na to, aby se nezobrazovaly hodnoty přímo z elektroměru, ale spotřeba elektrické energie v daných patnácti minutách. Proto se před vlastním vykreslením musí provést matematická operace, která spočítá odpovídající hodnoty.

Samotné zobrazení bylo trochu komplikované, a to z toho důvodu, že ZedGraph neumožňuje kombinaci umístění sloupců vedle sebe a nad sebe zároveň. Zvolil jsem proto nastavení zobrazení nad sebou a samostatný sloupec ručně v programu posunuji o malý časový úsek. Tím se dosáhne požadovaného výsledku. Po této úpravě však stále nejsou umístěny sloupce přesně v daných časových úsecích. Proto se musí ještě jednou posunout po časové ose tak, aby byly umístěny přesně v té čtvrthodině, ve které se naměřila spotřeba elektřiny.

Grafy SQ, SD

Jako další zobrazované funkce jsou v mé aplikaci umístěny dva grafy. Ty ukazují závislost jalového výkonu Q na zdánlivém výkonu S a deformačního výkonu D na zdánlivém výkonu S. Grafy jsou umístěny opět pod hlavním grafem vedle sebe. Jsou to v podstatě spojité grafy a také se tak vytváří. Odlišují se jen tím, že není vykreslená celá křivka, ale pouze její body. Do grafu se zobrazují pouze hodnoty z okolí

detekované události, takže buď zapnutí nebo vypnutí některého elektrospotřebiče v jedné ze třech fází.

Drobný problém nastal s rozmístěním těchto grafů. Pokud jsem zvolil pro oba grafy klasické rozvržení os X a Y, nastal problém s tím, že druhý graf byl posunut hodně doprava a nebyl tak zarovnán s hlavním grafem (Obr. B.1). Proto jsem zvolil řešení, že levý graf má osu Y nalevo, ale pravý graf ji má umístěnou napravo (Obr. B.2). Tím jsem docílil toho, že uprostřed mezi grafy je malá mezera a vpravo i vlevo jsou oba grafy zarovnány s vrchním hlavním grafem.

Grafy zapnutí, vypnutí

Grafy zapnutí a vypnutí jsou poslední z nabídky zobrazitelných grafů. Umístění a vyřešení problému s osou Y je shodné jako u grafů SQ a SD. Grafy jsou v aplikaci za účelem detailního pohledu na události zapnutí a vypnutí elektrospotřebičů. Ty v průběhu tvoří strmý přechod z jedné hodnoty na druhou. Ten je nutné analyzovat pro budoucí rozpoznávání jednotlivých spotřebičů. V grafech se zobrazují stejné průběhy jako v hlavním grafu, což jsou všechny výkony ve třech fázích. Rozdíl je ale v tom, že v tomto detailním pohledu se data načtou z nejpodrobnější tabulky. O způsobu načítání bude napsáno podrobněji dále.

Samotné vyhodnocení události zapnutí nebo vypnutí elektrospotřebiče probíhá programově v SQL. Zde se vytvoří tři tabulky pro zapnutí a tři tabulky pro vypnutí v jednotlivých fázích. Do nich se uloží pouze časy, kdy k těmto událostem došlo. Ve své aplikaci pak tyto časy načtu a zobrazím v ListBoxu. Po vybrání některé události z vypsaného seznamu se načtou detailní hodnoty kolem tohoto času a zobrazí se v grafu. Zobrazený interval jsem zvolil v intervalu -1 sekunda až +5 sekund. Je tak možné sledovat první okamžiky po zapnutí, které jsou pro další identifikaci spotřebiče nejdůležitější.

Společně s grafy se do TextBoxu vypisují důležité informace ze zobrazeného intervalu. Jedná se o hodnoty všech výkonů před a po zapnutí nebo vypnutí spotřebiče. Počet kroků vpřed a vzad od samotné události se dá nezávisle nastavit. Nastavení je ošetřeno tak, aby v případě, kdy zadaná hodnota dosáhne mimo graf, program sám zvolil maximální možnou hodnotu pro danou volbu. Dále se vypíší maximální a minimální hodnoty všech výkonů. Seznam zobrazovaných údajů nemusí být konečný, ale záleží na tom, zda bude v budoucnu potřeba z průběhů zjišťovat další informace.

3.2 Načítání z databáze

Aby bylo vůbec možné zobrazovat nějaká data, průběhy, statistiky a další, je předem nutné získat potřebná data. Měřící přístroje, ze kterých výsledky zobrazuji, ukládají data do databáze SQL. Z této databáze tak musí můj program umět načítat data, což lze provést dvěma způsoby. Každý z nich má určité výhody, ale i nějaká úskalí. V mém případě byl požadavek na přístup do databáze pomocí Entity Frameworku, se kterým společnost KMB systems doposud nepracovala a tudíž jim moje aplikace slouží zároveň k testování tohoto přístupu k datům.

3.2.1 Entity Data Model

Entity Data Model je základním prvkem, přes který lze s daty v databázi pracovat. S postupem času, jak se při psaní programu objevovaly nové možnosti využití Entity Frameworku, se tento model mnohokrát změnil. Od počátečního testování, kdy obsahoval jednu tabulku, přes kompletní schéma celé databáze, až k podobě, kterou má nyní. Každá změna s sebou přinesla určité vylepšení a myslím si, že v budoucnu se může i nadále vyvíjet.

Způsob vytvoření

Entity Data Model se nejjednodušeji vytvoří přímo z databáze. Do aplikace je nutné přidat soubor ADO.NET Entity Data Model. Následuje zvolení databáze, ze které má být model vytvořen. Poté se přejde k vybrání tabulek, pohledů a procedur, které mají být ve výsledném modelu zahrnuty. Aby byl model funkční, stačí vybrat jen jednu tabulku. Po potvrzení vybraných částí se vytvoří požadovaný model, který se zobrazí ve Visual Studiu. V modelu jsou zobrazeny všechny vybrané položky a jejich vzájemné vazby. V jednotlivých tabulkách jsou znázorněny jednotlivé sloupce a případně jejich atributy. Cokoli měnit v tomto modelu přímo ve Visual Studiu však není dobré, protože model přestane odpovídat databázi a program se stane nefunkčním.

Použitý EDM v mé aplikaci

Model databáze, který používám ve své aplikaci, prošel mnohými změnami. Jednak kvůli zrychlení programu, nebo z důvodu, že bylo potřeba načítat nové údaje. Kvůli tomu se několikrát změnila struktura databáze, nebo byly přidány nové tabulky. Jelikož nedisponuji připojením k žádnému měřícímu přístroji, pokaždé musím stáhnout zálohu databáze z internetu.

První model, který jsem použil, zahrnoval tabulky s naměřeným napětím, proudem a výkonem. V programu jsem vyhodnocoval události zapnutí a vypnutí. Následně se vypisovaly hodnoty v okolí těchto událostí z použitých tabulek. Tento model však měl ten nedostatek, že první načtení dat trvalo značnou dobu. Vyvíjený program ale byl na počátku svého vzniku a jeho optimalizace tak nebyla prioritní. Místo toho se analyzovala získaná data a dospělo se k názoru, že tabulky s napětím a proudem nejsou potřeba.

Následující model tedy obsahoval tabulku jen s výkonem, která ale sama o sobě je značně rozsáhlá a žádného zrychlení načítání dat se nedosáhlo. Naopak se objevily další komplikace. Jednak se zjistilo, že měřící přístroj nemusí ukládat data do databáze chronologicky za sebou. To znamenalo závažný problém, protože ZedGraph neumí sám o sobě srovnat data podle času, ale spojuje vždy sousední dvojice bodů. Tím vznikl naprosto nesmyslný průběh. Rovnat data v programu se zkoušelo, ale bylo to další zbytečné zpomalení programu. Druhý problém nastal v případě, kdy databáze obsahovala větší množství záznamů. Program se pak stal téměř neovladatelným kvůli své pomalosti a také detekce událostí zapnutí a vypnutí trvala značně dlouho.

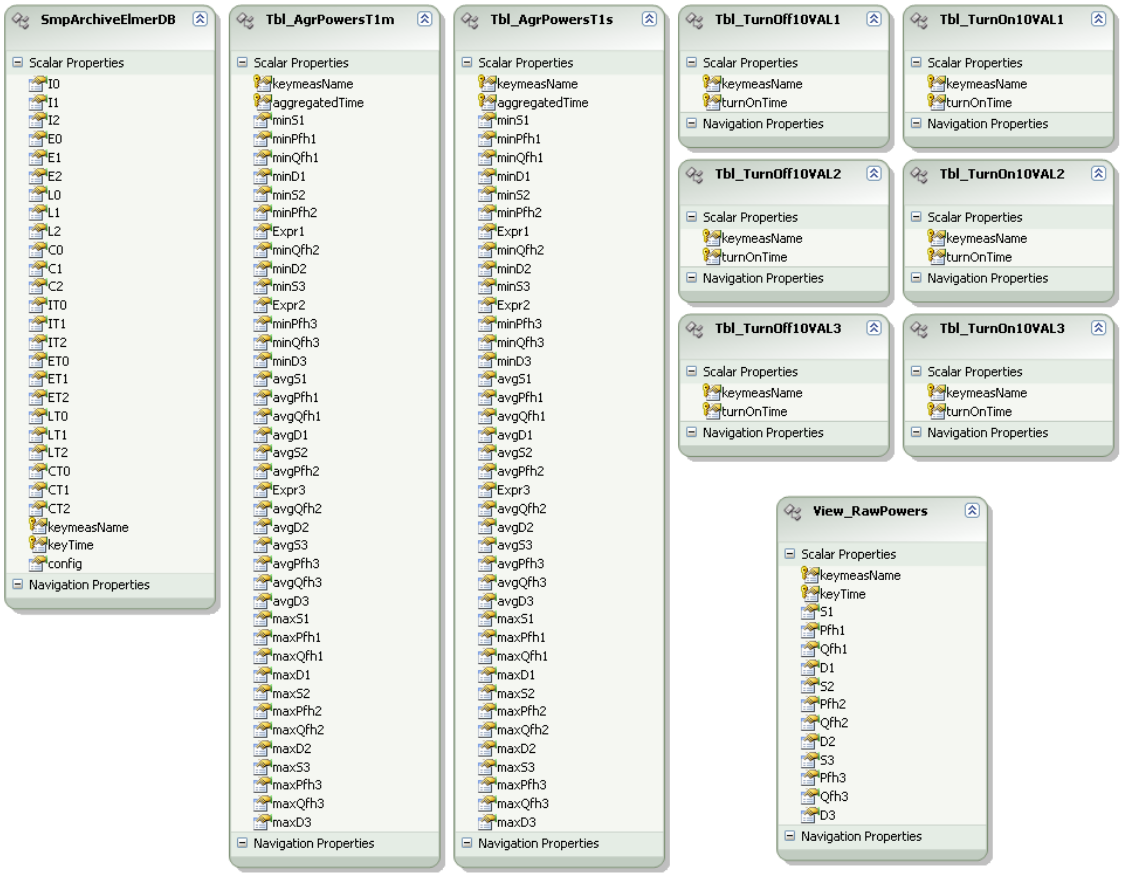
Kvůli složitosti tabulky s výkonem, která obsahuje pro můj program značnou část nepoužívaných dat, se došlo k závěru, že se z dané tabulky vytvoří nová a jednodušší. Ta bude obsahovat pouze potřebné údaje, které v aplikaci využiji. Vznikla tak tabulka obsahující všechny naměřené výkony ve třech fázích. Tato tabulka se vytváří přímo v SQL. Zároveň se hodnoty seřadí podle času, což řeší jeden z výše zmiňovaných problémů. Rychlost programu se tak zvýšila, ale stále přetrvával problém s rozsáhlejší databází a detekováním velkého množství událostí zapnutí a vypnutí.

Vznikl tedy nápad, přidat do databáze další tabulku, která bude obsahovat pouze časy těchto událostí. Tabulka se opět vytváří přímo v SQL. Tím pádem došlo v programu ke zrušení funkce na detekci zapnutí a vypnutí. Nahradila ji však funkce načítající časy z nové tabulky. Výhodou tohoto řešení je fakt, že se události nemusí detekovat v programu stále znovu, ale jejich zjištění proběhne jen jednou v SQL. Doba vytvoření takovéto tabulky se u rozsáhlých databází pohybuje v řádu několika minut. Po odzkoušení tohoto způsobu se ještě doplnily tabulky zapnutí pro ostatní fáze a zároveň i tabulky pro vypnutí.

Posledním problémem byly rozsáhlé databáze, a tím pádem dlouhé načítání dat. První částí řešení bylo vytvoření agregovaných tabulek z důvodu, že měřící přístroje

standardně měří po 200ms. Zobrazovat však naměřené hodnoty v rozsahu jednoho týdne, dne, nebo hodiny s takto velkým rozlišením je naprosto zbytečné. Agregované tabulky tak obsahují zprůměrovaná data po jedné minutě a po jedné vteřině, ze kterých program načítá data při zvoleném větším časovém úseku. Druhou částí řešení je načítání dat po částech, což je popsáno dále.

Výsledný model obsahuje jeden pohled se všemi výkony ve třech fázích, dvě tabulky agregovaných dat po jedné minutě a po jedné sekundě, tabulku s hodnotami z elektroměru a šest tabulek pro události zapnutí a vypnutí v jednotlivých fázích.



Obr. 3.3: Použitý EDM

3.2.2 Problémy a jejich řešení

Entity Framework je poměrně novou záležitostí a nejsou s ním moc velké zkušenosti, proto se objevilo při vyvíjení softwaru několik problémů. Jednalo se o rychlost načítání, o počet načítaných hodnot apod. S postupem času se podařilo všechny závažné problémy odstranit, nicméně to neznamená, že by se výsledný program nemohl dále vyvíjet.

Rychlost Entity Frameworku

Jedním z hlavních kritérií všech programů je jejich rychlost. U mé aplikace je to stejné, protože žádný uživatel nechce čekat u počítače několik minut, zatímco se na pozadí načítají data. Něco málo o způsobu řešení již bylo nastíněno, ale problém s rychlostí Entity Frameworku je rozsáhlejší problematika, která bude popsána na následujících řádcích.

První problémy se začaly objevovat při použití složitějšího modelu, konkrétně při použití rozsáhlé tabulky s výkonem. Nejen načítání, ale i rychlost překladu programu se neúměrně zpomalily. Zajímavé v tomto případě bylo, že výsledná rychlost nezáležela na počtu načítaných veličin. Program byl stejně pomalý při načítání samotného napětí jako při načítání napětí, proudu a výkonu. Rychlost závisela pouze na tabulkách v EDM. Vhodným řešením se ukázal volně šiřitelný program EdmGen2RL , který je dostupný na webové adrese: ⟨http://code.msdn.microsoft.com/EdmGen2⟩. Pomocí něho se do složky s modelem databáze vytvořil ještě jeden soubor, a tím se značně urychlilo načítání za běhu programu. Aby se zrychlil i překlad programu, bylo nutné ve Visual Studiu vypnout automatickou validaci modelu. Nakonec se u tohoto řešení nezůstalo a použila se nová, daleko jednodušší tabulka naměřených výkonů.

Načtení velkého množství hodnot

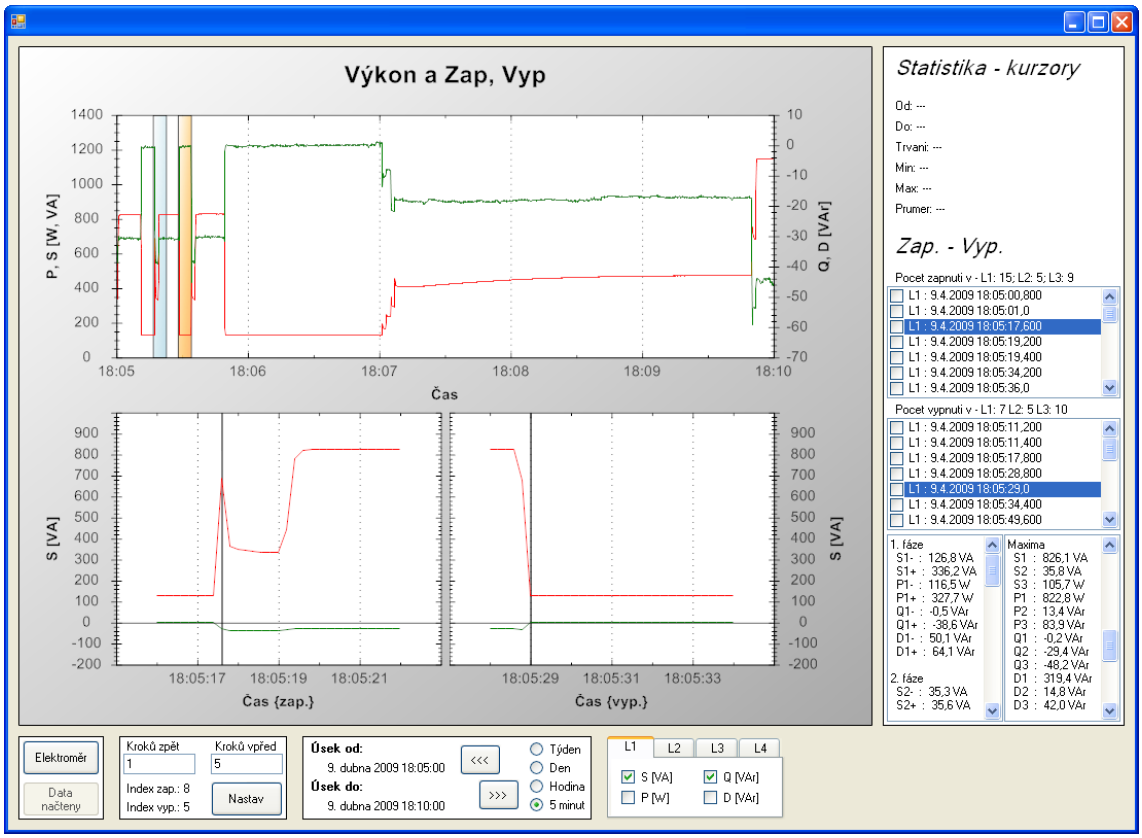
Zároveň se vyskytl problém při použití rozsáhlé databáze. Nejenže se prodloužila doba, za kterou se načetla všechna data do programu, ale jejich velké množství i po načtení program natolik zpomalilo, že ho nešlo téměř používat. Drobné zlepšení nastalo při použití agregovaných tabulek, o kterých jsem již psal, nicméně stále byl program pomalý. Nezbylo tak nic jiného, než načítat do programu jen ty hodnoty, které uživatel uvidí zobrazené v danou chvíli. To vyžadovalo výrazně pozměnit funkce, starající se o načítání dat, a zároveň použít LINQ to SQL, aby se mohl specifikovat časový úsek, který bude z databáze načten. Ve výsledku se tak docílilo toho, že v závislosti na vybraném časovém úseku je v jednu chvíli v programu uloženo maximálně několik tisíc hodnot. Naproti tomu při načítání dat z celé databáze najednou musel program pracovat až s milióny dat najednou.

V tuto chvíli se tímto způsobem nenačítá průběh z elektroměru, který má sám o sobě mnohonásobně méně hodnot, protože se měří jen každých patnáct minut. Nicméně při rozsáhlé databázi trvá první načtení relativně dlouho, což je jedna

z nevýhod Entity Frameworku. Každé další načítání, které se provádí při posunu časové osy, ale trvá maximálně několik vteřin i pro rozsáhlé databáze.

4 Výsledná aplikace

4.1 Finální verze programu



Obr. 4.1: Výsledný program

4.1.1 Grafy

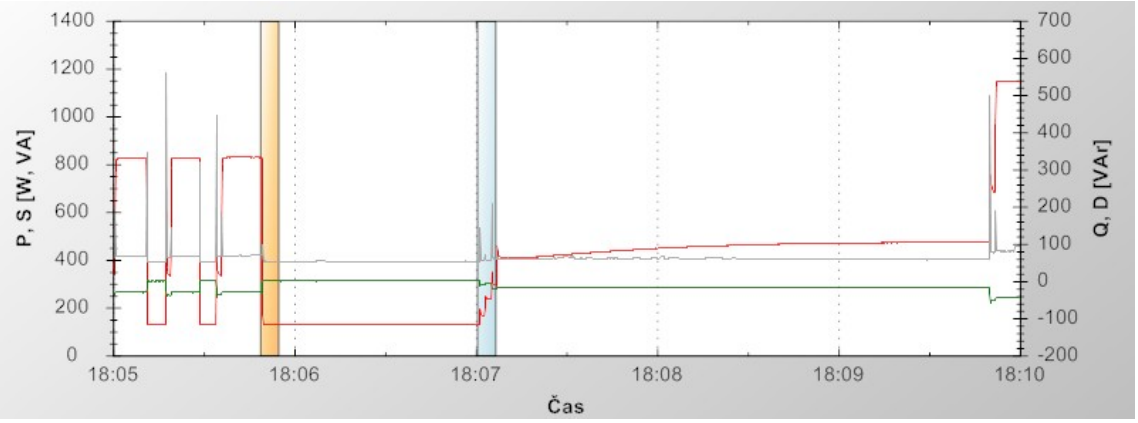
Prostor pro grafy je vymezen v levé horní části aplikace. O většinu grafiky se stará sama komponenta ZedGraphControl. Pomocí této komponenty jsou nastaveny všechny nadpisy, barevné přechody na pozadí nebo okraje grafů. I barevné obdélníky, označující kurzory, nebo jiné významné časové úseky v grafech jsou součástí funkcí ZedGraphu.

Jediné, co jsem do prostoru grafů musel přidat sám, byl panel označující načítání dat. Na panelu je textový popisek, oznamující, že program načítá data z databáze. K vizualizaci tohoto procesu jsem vytvořil animovaný obrázek gif, který má podobu běžícího ProgressBaru.

Základní graf

Základní graf se vykreslí po načtení dat z databáze a na svém místě zůstává až do ukončení aplikace. Zobrazují se v něm spojité průběhy z naměřených hodnot čtyř

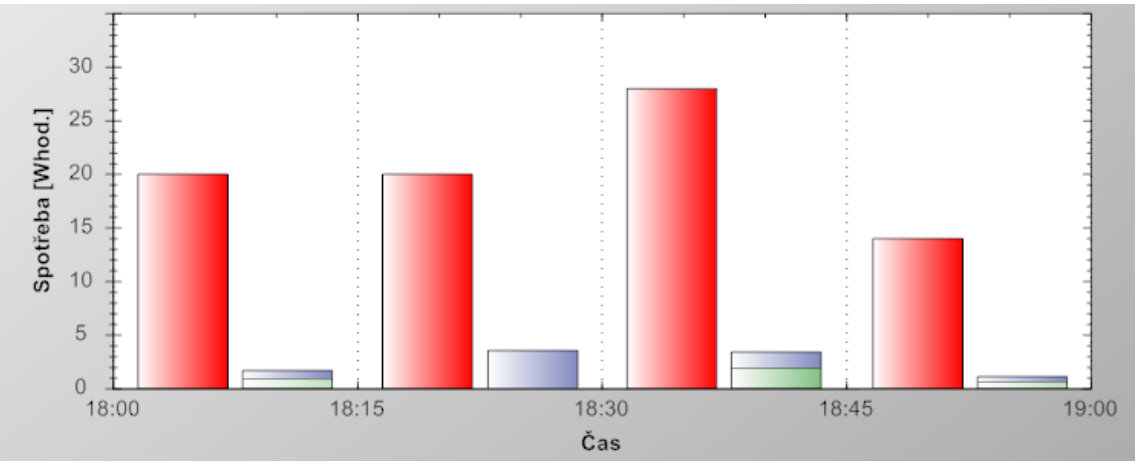
různých typů výkonů, a to ve třech fázích. Najednou je tedy možné zobrazit až dvanáct průběhů v jenom grafu. Každý průběh je pro lepší orientaci vykreslen jinou barvou. Také jsem se snažil, aby stejné výkony v různých fázích byly kresleny stejným odstínem. Proto je v první fázi vykreslen zdánlivý výkon červeně, ve druhé fázi tmavě červeně a ve třetí světle červeně. S ostatními výkony je to podobné.



Obr. 4.2: Hlavní graf

Graf z elektroměru

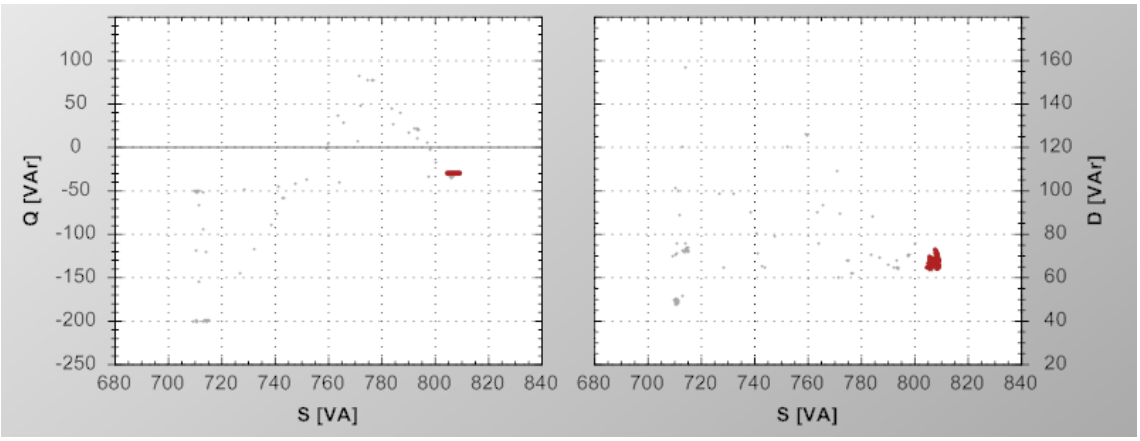
Graf tvořený hodnotami z elektroměru se objeví také hned po načtení dat, nicméně lze místo něj zobrazit další typy grafů. Jedná se o sloupcový graf tvořený dvěma sloupci, přičemž jeden ze sloupců zobrazuje dvě veličiny. Ve výsledku tak tento graf zobrazuje tři průběhy najednou. Každá veličina má svoji barvu sloupce pro lepší orientaci. Tento graf nelze nijak ovládat, nebo z něj získávat podrobnější informace, je to spíše doplňkový průběh.



Obr. 4.3: Graf elektroměru

Grafy SQ, SD

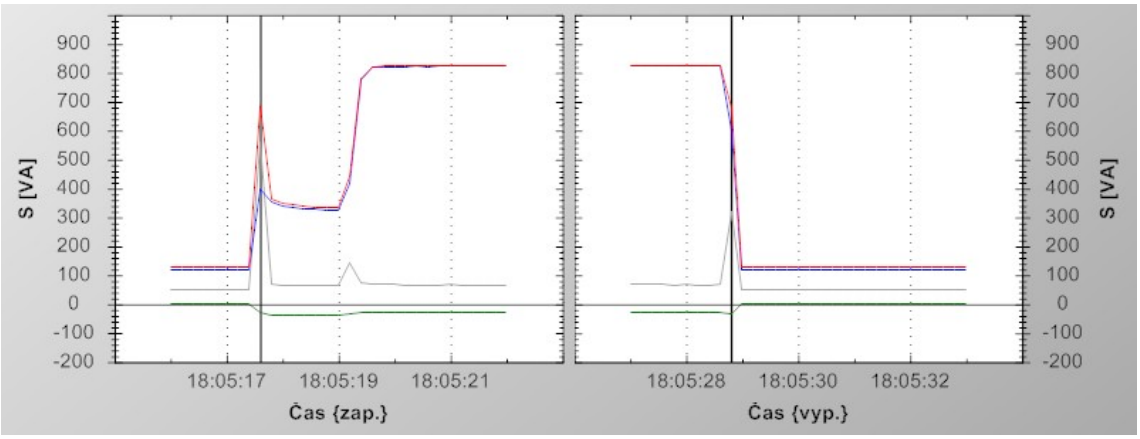
Po přepnutí na průběhy SQ a SD se nahradí průběh z elektroměru dvěma bodovými grafy. Jedná se v podstatě o spojité průběhy, ale vykreslují se pouze body. Data pro tyto grafy se berou z okolí událostí zapnutí a vypnutí. Ve výsledku tak je v grafu několik skupin bodů. Každá skupina odpovídá zapnutí nebo vypnutí nějakého elektrospotřebiče. Dále jsou grafy provázány s kurzory, takže při výběru určitého časového úseku v hlavním grafu se příslušné body zvýrazní.



Obr. 4.4: SQ, SD grafy

Grafy přechodových dějů

Jedná se o další typ grafů. Jsou to spojité průběhy, zobrazující vždy jednu událost zapnutí a vypnutí vybranou ze seznamu těchto událostí. V podstatě by se dalo říct, že se jedná o detailní pohled na konkrétní časový úsek v hlavním grafu. Z tohoto důvodu jsou hlavní průběhy spolu s průběhy přechodových dějů silně provázané. To znamená, že v grafech přechodových dějů jsou zobrazeny vždy jen ty průběhy, které jsou i v hlavním grafu.



Obr. 4.5: Grafy přechodových jevů

4.1.2 Ovládací prvky a zobrazování informací

Panel s informacemi o datech

Vedle grafů je napravo umístěn panel, který slouží pro vypisování nejrůznějších informací ze zobrazených hodnot. Nahoře je několik textových popisků, které patří k funkci kurzorů. Zde se vypisuje začátek a konec vybraného úseku, délka jeho trvání a maximální, minimální a průměrná hodnota v tomto úseku. Pod těmito popisky jsou umístěny dva ListBoxy, ve kterých se zobrazují události zapnutí a vypnutí z právě zobrazeného časového úseku. ListBoxy jsou dva, aby bylo odděleno zapínání a vypínání. K těmto ListBoxům patří dva TextBoxy, ve kterých se zobrazují informace o vybrané události zapnutí či vypnutí.

Ovládací prvky

Spodní část okna programu patří ovládacím prvkům. Úplně vlevo na kraji je panel s tlačítkem na načtení dat z databáze a tlačítko pro přepínání různých typů grafů. Vedle tohoto panelu se nachází skupina ovládacích prvků pro detailnější analýzu událostí zapnutí a vypnutí. Dá se zde nastavit, v kterém čase, před a po vybrané události, se budou vypisovat požadované informace.

Následuje panel pro ovládání posuvu po časové ose. K dispozici jsou dvě tlačítka pro posun dopředu a dozadu. Každé tlačítko je ošetřeno tak, aby bylo nefunkční v případě, že se uživatel dostane na začátek nebo konec měření. Není tak možné dostat se na takový časový úsek, kde nejsou žádná data. Dále jsou zde dva textové popisky, informující uživatele o časech začátku a konce právě zobrazeného časového úseku.

Jako poslední je ovládací prvek, sloužící pro nastavení zobrazení jednotlivých průběhů v jednotlivých fázích. Pro přepínání fází slouží záložky a pro vybírání průběhů jsou použity klasické CheckBoxy.

Obr. 4.6: Ovládací prvky

Zmenšování a zvětšování okna programu

Samozřejmostí je, že se dá okno aplikace zvětšovat nebo zmenšovat. Podle velikosti okna se zvětšuje nebo zmenšuje i plocha s grafy. Panel na zobrazování informací nemění svoji velikost, ale drží si stále stejný odstup od plochy s grafy a pravého okraje okna. To samé platí i pro všechny panely s ovládacími prvky, které ale udržují konstantní odstup od spodního okraje. Extrémní zmenšení, kdy dojde k překrytí ovládacích panelů s informačním panelem, zatím není řešeno.

4.2 Přechodové jevy

Přechodovými ději nazývám ve své aplikaci události zapnutí a vypnutí různých elektrospotřebičů. Po jejich zapnutí totiž nenastává strmá hrana v průběhu výkonu, ale vznikají různé špičky, mění se i jiné výkony, než jen ten činný apod. Aby bylo možné správně spotřebiče rozpoznat a následně spárovat jejich vzájemné zapnutí a vypnutí, musí se tyto přechodové jevy analyzovat. K tomuto účelu jsem právě vyvíjel svoji aplikaci.

4.2.1 Zpracování přechodových dějů

Detekce přechodových dějů

Na počátku vývoje jsem zpracovával naměřená data přímo ve svém programu. Funkce, kterou jsem k tomuto účelu vyvinul, nebyla špatná. Uměla rozpoznat přechodový jev, který byl naměřen v několika vzorcích po sobě a utvořit z něj jeden samostatný. Dále ukládala detekované události do souboru. Nevýhodou bylo, že při větším počtu dat jejich analýza trvala zbytečně dlouho.

Proto se přešlo k řešení, že přechodové děje se detekují přímo v SQL. Vzniklo tak šest nových tabulek, ze kterých jsou tři pro zapínání ve třech fázích a tři pro vypínání, taktéž ve třech fázích. Do těchto tabulek se ukládá pouze čas, kdy k přechodovému ději došlo, o všechno ostatní se stará sám program. Jedná se především o načtení dat v okolí detekované události.

Práce s přechodovými ději v programu

Zpracování probíhá následujícím způsobem. V časovém rozmezí, které je zobrazeno v hlavním grafu, se načtou všechny časy přechodových jevů. Následně se vypíšou do dvou ListBoxů, z čehož je jeden pro události zapnutí a druhý pro události

vypnutí. V jaké fázi byl přechodový jev detekován je vypsáno spolu s časem přímo v ListBoxu.

Po kliknutí na některou událost zapnutí nebo vypnutí se v hlavním grafu zvýrazní pomocí barevného obdélníku časový úsek dlouhý šest sekund. Tento úsek označuje tu část průběhu, která se zobrazí v grafech přechodových jevů. Dále se vypíší informace z okolí vybraného přechodového děje do příslušného TextBoxu. Události zapnutí a vypnutí mají nejen různé ListBoxy, ale i TextBoxy, do kterých se vypisují různé informace.

4.2.2 Vizualizace přechodových jevů

Jak již bylo naznačeno v minulém odstavci, spolu s výpisem dat se provede i zobrazení krátkého časového úseku v okolí přechodového jevu do příslušného grafu. Protože data v hlavním grafu mohou být načtena z tabulky, kde jsou hodnoty uložené po jedné minutě nebo jedné sekundě, musí se do detailních grafů načíst nová data z nejpodrobnější tabulky. Data se načítají v rozsahu jedna vteřina před časem přechodového jevu, až pět vteřin za něj. Musí se načíst průběhy všech výkonů ve všech fázích. Do grafu se zároveň vyznačí svislá čára, která značí přesný čas, ve kterém byla událost zapnutí nebo vypnutí detekována.

4.3 Rychlost programu

4.3.1 Rychlost ZedGraphu

Maximální počet zobrazovaných hodnot

Samozřejmě že ZedGraph nemůže zobrazovat ohromné množství hodnot najednou. Sice je umí ZedGraph zobrazit, ale program se natolik zpomalí, že je téměř nepoužitelný. Během vývoje programu jsem došel k číslu kolem 15000 hodnot v každém průběhu, kdy byl program ještě rozumně použitelný. Program využívá dvanáct průběhů výkonů, jeden průběh z elektroměru a dva SQ, SD grafy. Z toho usuzuji, že maximální počet hodnot, které lze mít v programu uložené, se může pohybovat kolem 200 000.

Toto číslo bude nejspíše závislé i na výkonu samotného počítače, velikosti operační paměti apod. Proto se metoda postupného načítání dat k tomuto číslu ani nepřibližuje, aby byla dostatečná rezerva pro běh programu na slabších počítačích.

Optimalizace zobrazování

S postupem času, jak jsem vytvářel svůj program, jsem několikrát předělal princip zobrazování grafů a průběhů. Nejdříve jsem se domníval, že kvůli každé změně v jakémkoli průběhu je nutné překreslit celý graf. To mělo za následek, že se mi jednotlivé průběhy vrstvily na sebe a ve výsledku tak zpomalovaly postupně celý program.

Nakonec jsem pochopil, že žádné překreslování grafu není nutné. Stačí po změně zobrazovaných hodnot zavolat funkci na opětovné překreslení buď celého Formu nebo ZedGraphu a změny se okamžitě projeví. Z tohoto důvodu můj program sice obsahuje funkci, která měla zajišťovat překreslování grafů, ale nakonec je z ní funkce, která inicializuje a nastaví všechny průběhy na začátku a poté už v programu není potřeba.

4.3.2 Rychlost načítání dat

Načítání dat ve finální verzi

Nevýhodou Entity Frameworku je, že první načítání hodnot trvá déle, než následující načítání. To je jeden z důvodů, proč první načtení hodnot po spuštění programu trvá déle, než když se načítají data při posuvu časové osy. U menších databází se čas prvního načtení pohybuje v řádu sekund až desítek sekund. U větších databází tento proces trvá déle, řádově minuty.

Z tohoto důvodu načítání běží v samostatném threadu, což zabraňuje zamrznutí programu do doby, než budou všechny hodnoty načteny. Dále jsem program doplnil informačním panelem s grafickým znázorněním a popisem, že program pracuje. Dále je možnost načítání přerušit, nicméně samotné přerušení se provede až po dokončení načítání z některé tabulky. Snaha o okamžité přerušení byla, ale řešení by bylo velice pracné. Navíc samotná rychlost načítání není hlavní prioritou mého programu, proto se tato problematika dále neřešila.

4.4 Možné rozšíření aplikace

4.4.1 Další ovládací prvky

Vylepšení kurzorů

Kurzory by bylo možné rozšířit v několika oblastech. Možná by nebylo na škodu, aby se kurzory daly používat jen v případech, kdy je uživatel chce použít. To znamená

přidat tlačítko, které by funkci kurzorů zpřístupňovalo. Dále by bylo dobré, aby se dal kurzor po jeho vyznačení v grafu dodatečně roztáhnout. To proto, aby uživatel mohl korigovat jen jednu stranu časového úseku a druhá se neměnila. Dále by bylo možné doplnit další informace, které se z úseku mezi grafy vypisují. Zabývat se touto problematikou dřív nemělo příliš smysl, protože se často měnily načítané veličiny. V současnosti je program v takovém stavu, že se načítaná data již příliš nemění a je tak reálné doplnit další zobrazované údaje.

SQ, SD grafy

Průběhy SQ a SD samozřejmě nejsou jedinými možnými kombinacemi výkonů, proto se nabízí možnost přidat i další varianty těchto bodových grafů. O tomto problému jsem již uvažoval. Nejlepší by bylo stále zobrazovat pouze dva grafy vedle sebe, ale přidat možnost výběru grafu přes kontextové menu. Otázkou ale zůstává, které kombinace výkonů má smysl zobrazovat a které ne.

Posuv časové osy

Posuv časové osy sice můj program obsahuje, ale tento posuv není plynulý. Proto se nabízí možnost posouvat časovou osu plynule. Nicméně to by byl asi největší problém, protože data program načítá postupně podle zvoleného časového úseku. Jak zajistit posouvání časové osy v reálném čase spolu s načítáním dat a zároveň i jejich vykreslováním si momentálně nedokážu představit. Možností by bylo načítat o něco více dat, než je rozsah časové osy a rozdělit tak posun a načítání na dvě samostatné funkce.

4.4.2 Načítání dat z databáze

Entity Data Model

Protože může kdykoliv přibýt požadavek na načítání nových hodnot z nové tabulky, první věc, která by se v programu změnila, by byl EDM. Začlenění nového modelu do programu není velký problém. Pro nově přidanou tabulku by se však musela napsat nová funkce na načítání dat z této tabulky.

Metoda načítání dat

V současnosti program umožňuje díky postupnému načítání dat práci i s rozsáhlými databázemi. Může však nastat případ, že by načítání dat někomu přišlo

příliš pomalé a bylo tak nutné optimalizovat v programu ty funkce, které se o načítání dat starají. První z možností by nejspíše bylo načítat i data z elektroměru postupně. Je také možnost, že se objeví nové vlastnosti Entity Frameworku a bude vhodné předělal celý mechanismus načítání dat z databáze.

5 Závěr

Výsledný software, který odevzdávám spolu s bakalářskou prací umí zobrazovat několik druhů grafů, vypisovat z průběhů důležité informace a umožňuje ovládání grafu pomocí několika funkcí. Tím splňuje zadání bakalářské práce. Není to však finální verze, kterou si objednala společnost Seven o. p. s. Během vývoje byl program několikrát představen právě v této společnosti a pokaždé na něj byly kladné ohlasy. Také vždy byly vzneseny nové požadavky, podle kterých se program vyvíjel.

Právě proto, že byl software několikrát představován, docházelo postupně k upřesňování konkrétního zadání. V době, kdy o program ještě nebyl žádný zájem a měl sloužit hlavně pro bakalářskou práci, jsem se snažil dosáhnout přímo cíle, a to programu, který by byl schopen detekovat časy zapnutí a vypnutí elektrospotřebičů a vypisovat důležité informace z okolí těchto časů. Vyvíjením takového programu jsem se několik týdnů zabýval, ale po tom, co byl o program projeven zájem, jsem se začal soustředit na nové funkce.

Doplňoval jsem do programu ty prvky, na které byly kladeny požadavky, a opustil jsem přímou cestu k finální verzi. Hlavním požadavkem je přesné určení času, kdy byl některý elektrospotřebič zapnut nebo vypnut. Poté jsem doplnil do programu několik grafů, ze kterých může uživatel analyzovat různá elektrická zařízení. Po důkladné analýze budu moci s vývojem svého programu pokračovat. Začnu nejspíše s tvorbou funkce, která na základě získaných poznatků sama zanalyzuje průběh a přiřadí k jednotlivým přechodovým jevům příslušné spotřebiče.

Během vytváření bakalářské práce jsem se dostal k několika novým prostředkům, které jsem doposud neznal. Nejvýraznější z nich je programovací jazyk C#, ve kterém svoji aplikaci píšu. Do té doby jsem se s tímto programovacím jazykem příliš nesetkal. Jedinou výjimku tvořil jeden semestr, kdy jsme pracovali v příbuzném C++. Dále jsem se seznámil s prací v SQL a používáním databází nejenom při vyvíjení aplikace. Z tohoto pohledu považuji bakalářskou práci pro mě jako přínosnou.

Případný budoucí rozvoj programu je možný a také se s ním počítá. Já osobně nechci po odevzdání bakalářské práce s vývojem tohoto programu skončit, ale budu pokračovat dál. Doufám proto, že zájem, který byl doposud o program projeven, přetrvá, a dostane se mi tak příležitosti dokončit program třeba jako diplomovou práci.

Seznam použité literatury

[1] *.NET Framework Developer's Guide* [online].

URL: ⟨http://msdn.microsoft.com/⟩.

[2] *Devart: Entity Framework Tutorial* [online].

URL: ⟨http://www.devart.com/dotconnect/mysql/articles/tutorial_ef.html⟩.

[3] *Dokumentace ke knihovnám DXperience* [online].

URL: ⟨http://www.devexpress.com/⟩.

[4] *Dokumentace k měřícím přístrojům a programu RETIS* [online].

URL: ⟨http://www.kmb.cz/⟩.

[5] *Dokumentace knihovny ZedGraph* [online].

URL: ⟨http://www.zedgraph.org/⟩.

[6] *Internetová encyklopedie Wapedia: Licence GNU* [online].

URL: ⟨http://wapedia.mobi/cs/GNU_Lesser_General_Public_License⟩.

[7] Jiří Činčura. *Vyvojar.cz: Entity Framework vs. LINQ to SQL* [online].

URL: ⟨http://www.vyvojar.cz/Articles/601-entity-framework-vs-linq-to-sql.aspx⟩.

[8] Julia Lerman. *Programming Entity Framework* [online].

URL: ⟨http://oreilly.com/catalog/9780596520281/preview.htmlx⟩.

[9] Miroslav Virius. *C# – Hotová řešení*, Computer Press, ISBN 8025110842.

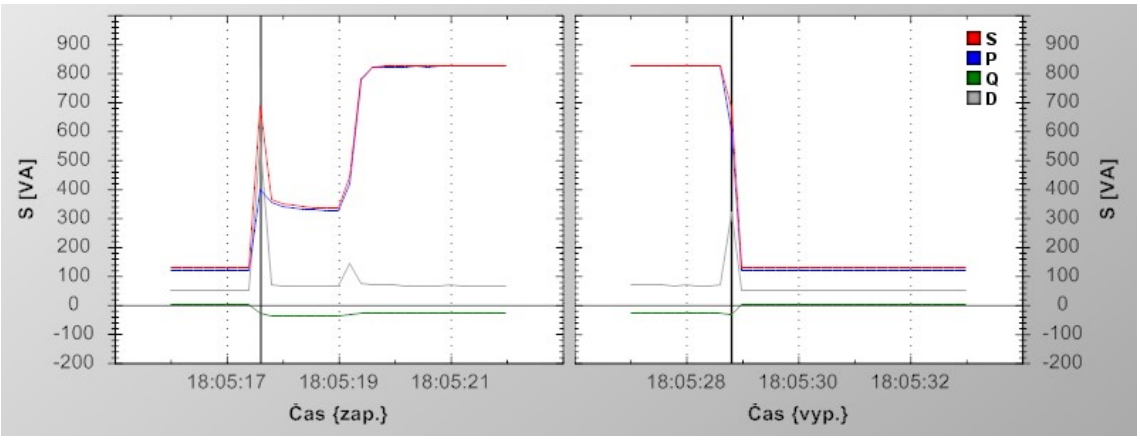
[10] *SourceForge.net: ZedGraph: Historie* [online].

URL: ⟨http://zedgraph.sourceforge.net/revision.html⟩.

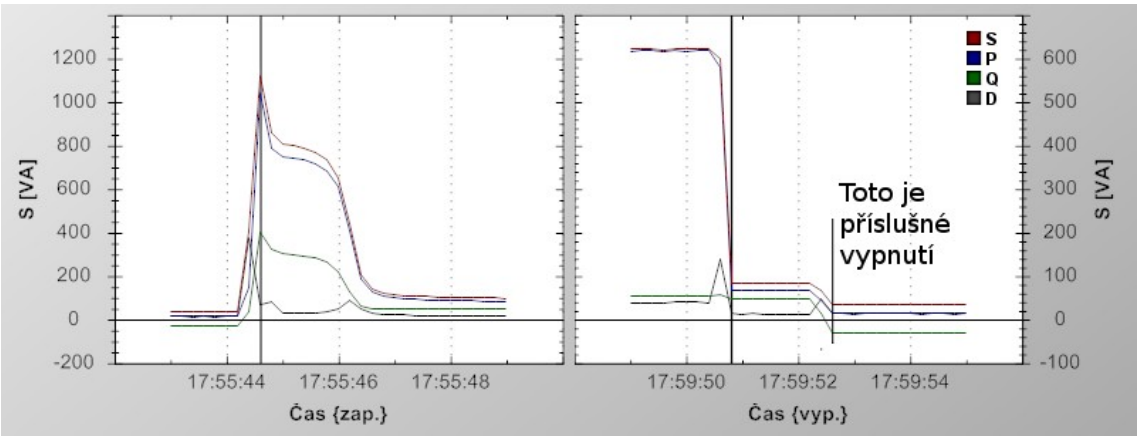
[11] *Způsoby přístupu k databázím* [online].

URL: ⟨http://www.fi.muni.cz/~xbatko/oracle/compare.html⟩.

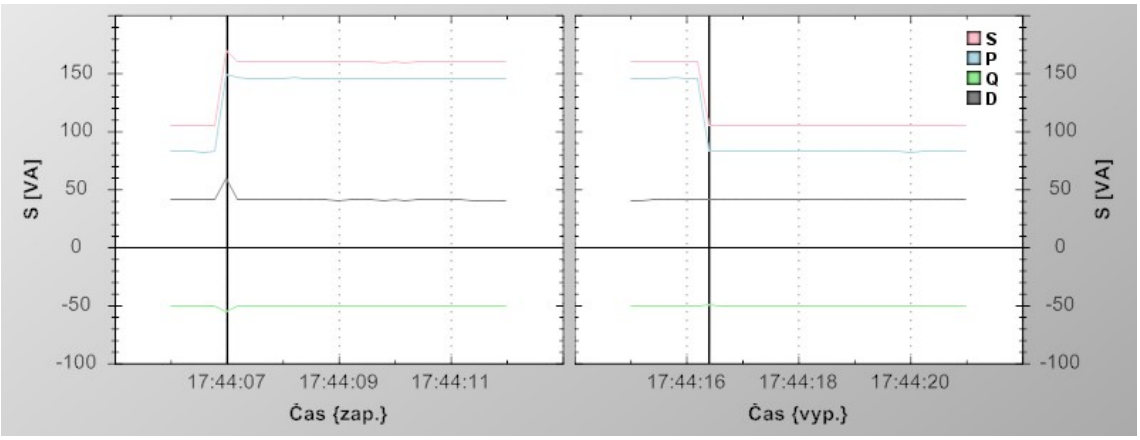
Příloha A – Ukázka přechodových jevů některých elektrospotřebičů



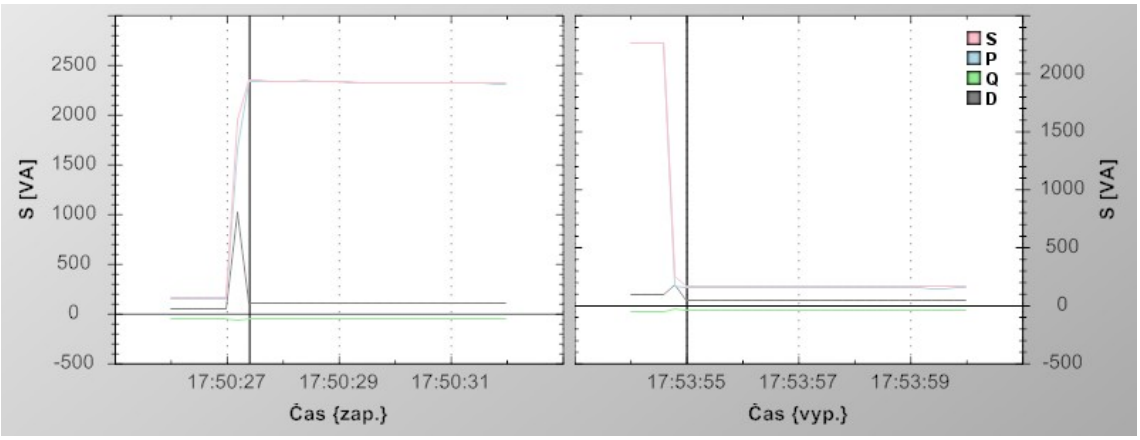
Obr. A.1: Zapnutí a vypnutí zářivky



Obr. A.2: Zapnutí a vypnutí PC Via Epia

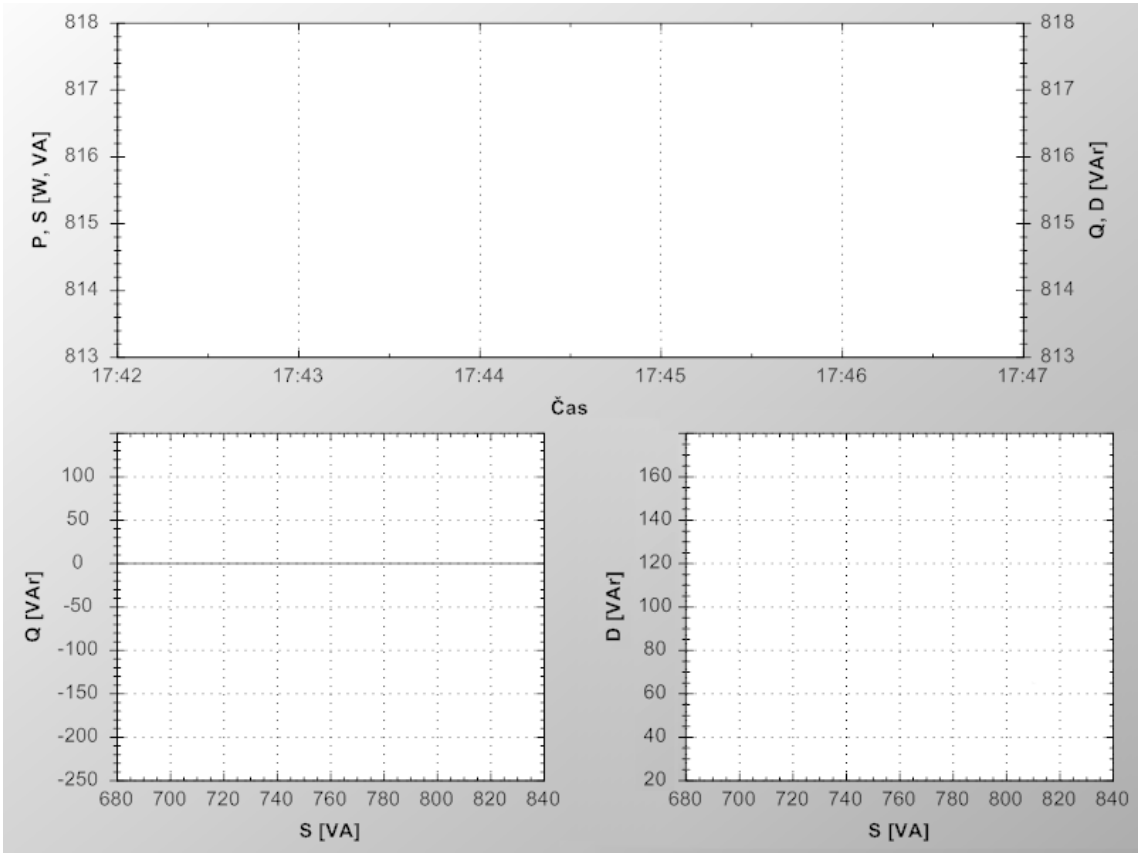


Obr. A.3: Zapnutí a vypnutí neznámého spotřebiče

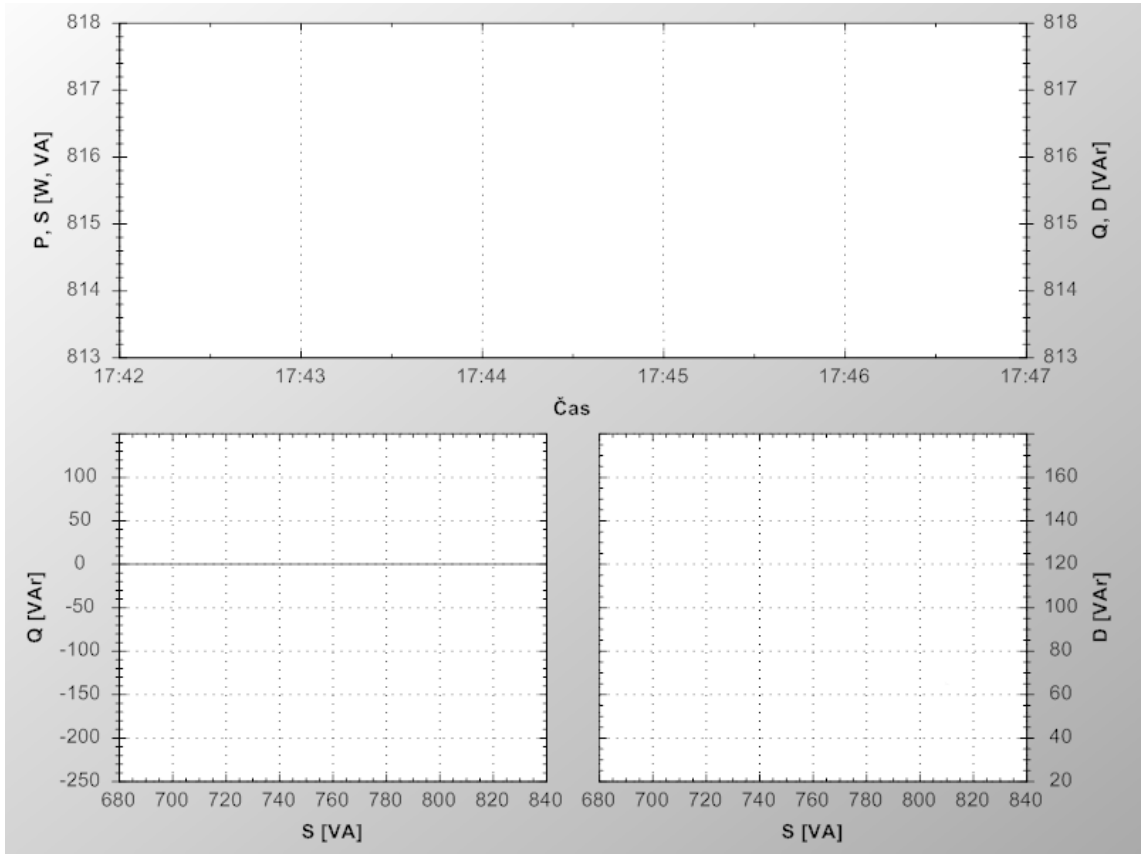


Obr. A.4: Zapnutí a vypnutí varné konvice

Příloha B – Ukázka zarovnání SQ a SD grafů s hlavním grafem



Obr. B.1: Nezarovnané grafy SQ a SD s hlavním grafem



Obr. B.2: Zarovnané grafy SQ a SD s hlavním grafem