TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Uživatelské rozhraní pro vícekanálové měřicí systémy

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Daniel Varnuška**
Vedoucí práce: Ing. Tomáš Bedrník
Konzultant: Ing. Jan Kraus, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

User interface for multi-channel measuring systems

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information technology

Author: **Daniel Varnuška**
Supervisor: Ing. Tomáš Bedrník
Consultant: Ing. Jan Kraus, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Uživatelské rozhraní pro vícekanálové měřicí systémy

Jméno a příjmení: **Daniel Varnuška**
Osobní číslo: M15000055
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2018/2019**

Zásady pro vypracování:

1. Proveďte rešerši existujících řešení GUI aplikací pro vícekanálové měřicí systémy dostupné na trhu v oblasti měření spotřeby elektrické, hydraulické a pneumatické energie (např. HMGWin, SensoWin).
2. Navrhněte a naprogramujte GUI aplikaci pro vícekanálový měřící systém sestavený z měřicích modulů firmy KMB systems s.r.o.
3. GUI aplikace musí umožňovat zejména synchronní online sledování měřených hodnot, nahlédnout do konfigurace měřicích modulů, rychlou kontrolu správného zapojení sond, libovolně upravovat topologii systému, vytvářet dopočítávané kanály a ukládání měřených hodnot v počítači.
4. Dále musí vytvářet jednoduché čarové grafy aktuálního příkonu a dalších parametrů včetně možnosti základní práce s grafem (např. vytváření ořezů, přidávání a ubírání zobrazených křivek) a vytvářet jednoduché koláčové grafy spotřeby energie.
5. Proveďte praktické ověření aplikace s vícekanálovým měřicím systémem, zhodnoťte její uživatelskou přívětivost a funkční možnosti a případně navrhněte budoucí možné směřování vývoje vytvořeného GUI.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická

Seznam odborné literatury:

[1] LAZAR, Guillaume a Robin PENEA, 2017. Mastering Qt 5. Birmingham Mumbai: Packt Publishing – ebooks Account. ISBN 978-1-78646-712-6.

[2] QML Applications. Qt Documentation [online]. Finland: The Qt Company, 2017 [cit. 2017-10-10]. Dostupné z: http://doc.qt.io/qt-5/qmlapplications.html.

[3] JINHUI, Q., L. D. HUI a Y. JUNCHAO, 2012. The Application of Qt/Embedded on Embedded Linux. In: 2012 International Conference on Industrial Control and Electronics Engineering [online]. s. 1304–1307.

[4] Learning Linux for embedded systems. Embedded [online]. 2015 [cit. 2017-10-10]. Dostupné z: https://www.embedded.com/electronics-blogs/open-mike/4420567/Learning-Linux-for-embedded-systems.

Vedoucí práce:	Ing. Tomáš Bedrník Ústav mechatroniky a technické informatiky
Konzultant práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	10. října 2018
Předpokládaný termín odevzdání:	30. dubna 2019

L. S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci 10. října 2018

Prohlášení

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

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

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

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

30. 4. 2019

Daniel Varnuška

Abstrakt

Práce se zabývá návrhem grafického uživatelského rozhraní ve frameworku Qt. Hlavním tématem je návrh prototypu softwaru, jak daná aplikace může vypadat v praxi. Rešerše obsahuje porovnání frameworků a ukázky aplikací již nasazených v provozu. Cílem práce je návrh a následné vytvoření aplikace, která se spojí s vybranými měřicími přístroji a měřená data promítne do křivek grafů. Aplikace umožňuje náhled do nastavení přístroje a přístup k jeho základním údajům. Závěrem jsou poznatky získané z použitých prostředků pro správnou funkcionalitu a návrhu prototypu aplikace. Dále jsou uvedeny jednotlivé doporučení pro další rozvoj softwaru.

Klíčová slova: Framework Qt, fázorový diagram, data reprezentována v grafech, realtimové měření, REST API.

Abstract

The thesis is about design of graphical user interface in Qt framework. The main theme is the design of the software prototype, how the application could look in practice. The research includes comparison of possible frameworks and demonstrations of applications already deployed in service. The aim of the thesis is to design and create an application that connects with selected measuring instruments and projects the measured data into graph curves. The application allows you to preview the device settings and access its basic data. The conclusion is the knowledge gained from the tools used for the correct functionality and design of the prototype application presented. Thesis includes recommendations for further software development.

Keywords: Framework Qt, phasor diagram, data represented in graphs, realtime measurement, REST API.

Poděkování

Rád bych poděkoval Ing. Tomáši Bedrníkovi za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce. Dále bych chtěl poděkovat rodině za jejich podporu při tvorbě práce.

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam zkratek

QML	Značkovací jazyk pro uživatelské rozhraní
SDK	Softwarový vývojový balíček
GUI	Grafické uživatelské rozhraní
REST	Representational state transfer
API	Rozhraní pro programování aplikací
FPS	Obrazové snímky za sekundu
HTML	Hypertextový značkovací jazyk
CSS	Kaskádové styly
JSON	JavaScriptový objektový zápis
USB	Univerzální sériová sběrnice
UTC	Koordinovaný světový čas
PC	Osobní počítač

Úvod

Tématem bakalářské práce je vytvoření vlastního návrhu softwarové aplikace pro grafickou reprezentaci dat vybraných měřicích přístrojů. Tento návrh je pouze jednou z možností, jak by daná aplikace mohla vypadat v nasazení. Reprezentace dat do grafů umožňuje konzumentům nahlédnout například do správy svého domu, kde mohou zjistit v jaké místnosti je největší odběr energie a podle toho třeba zařídit domácnost, tak aby byla nízkoenergetická. Pokud se odvážeme pouze od elektrických měřicích přístrojů a budeme zaznamenávat jednotlivé teploty v pokojích, tak můžeme z křivek měřených teplot zjistit, kde je například únik tepelné energie.

V bakalářské práci je seznámení se s již funkčními aplikacemi. Zásadní položkou u těchto aplikací je na jakém operačním systému jsou spustitelné. Aplikace jsou totiž zaměřené pro platformu Microsoft Windows a není dostupná žádná multiplatformní aplikace pro naše účely. Text pojednává o výběru vhodného frameworku a knihoven pro navržení multiplatformního softwaru s hlavním zaměřením spustitelnosti na různých distribucích systému Linux. Tato aplikace umožňuje zpracování, kde data jsou zobrazeny do grafů, ze kterých následně uživatel může provést nějaký závěr měření.

1 Výběr frameworku

Framework nebo softwarový framework je platformou pro vývoj softwarových aplikací. Poskytuje základ, na kterém mohou vývojáři softwaru vytvářet programy pro konkrétní platformu. Framework může například zahrnovat předdefinované třídy a funkce, které lze použít při zpracování vstupu, správu hardwarových zařízení a interakci se systémovým softwarem. To zjednodušuje proces vývoje, protože programátoři nemusejí znovu vytvářet tyto funkce při každém vývoji nové aplikace[15].

1.1 Qt

Qt toolkit byl vytvořen v roce 1999 společností Trolltech, která ho v roce 2008 prodala firmě Nokia. V roce 2011 Nokia ohlásila prodej práv na provoz podpůrných služeb a prodej licencí pro komerční projekty vytvořené pomocí Qt společnosti Digia. Zároveň však Nokia ujišťuje, že po transakci zůstane hlavním vývojářem tohoto toolkitu.

Qt je knihovna programovacího jazyka C++, ale existuje i pro další jazyky jako je Python, Ruby (QtRuby), C, Perl, Pascal, C#, Java a další. Podporuje SQL, zpracování XML, správu vláken, přístup k souborům, práci s grafikou a multimédii.

Qt je multiplatformní framework. Podporované platformy jsou - Windows od verze XP výše, Linux, Mac OS X, Android (a další mobilní a embedded platformy).

Qt bylo od počátku vyvíjeno v programovacím jazyku C++, ale využívá svou rozšířenou verzi jazyka, tzn. rozšíření o komunikaci mezi objekty, pro události a jejich obsluhy. Aplikace napsané ve frameworku Qt nelze napřímo zavolat do gcc kompilátoru. Framework využívá vlastního překladače, meta-objektového kompilátoru (nebo-li moc).

Moc je program, který zpracovává C++ rozšíření přidané knihovnou Qt. Moc čte hlavičkové C++ soubory, pokud nalezne jeden nebo více deklarací tříd, které obsahují makro Q_OBJECT, tak vytvoří C++ zdrojový kód obsahující meta-objektový kód pro tyto třídy. Mimo jiné, meta-objektový kód je vyžadovaný pro mechanismus signálů a slotů, runtime a dynamický systém vlastností.

C++ zdrojový soubor vygenerovaný pomocí moc musí být zkompilován a propojen s implementací třídy[14].

Při kompilaci Qt aplikace je aplikace propojena s knihovnami operačního systému a výsledkem je, že aplikace bude spustitelná pouze na daném systému.

Qt umožňuje vývoj pro několik platforem najednou, jelikož kód aplikace může být kompilován pro všechny platformy, které Qt podporuje, a výsledky budou po-

dobné ne-li stejné. Říkám podobné, protože Qt se snaží emulovat nativní widgety operačního systému. Je nutné aplikace otestovat a popřípadě mírně upravit podle cílového operačního systému. Mobilní uživatelská rozhraní jsou velmi odlišná od desktopů, takže grafické rozhraní se musí implementovat zvlášť pro mobilní aplikace.

Velkou výhodou Qt je velmi přehledně zpracovaná dokumentace a také vývojové programy Qt Creator nebo Qt Designer.

Qt představuje celý softwarový balík, ve kterém je mimo jiné zahrnuto:

- sada knihoven pro různé účely,
- kolekce překladačů,
- nástroje pro lokalizaci,
- vývojové prostředí Qt Creator.

Qt pravděpodobně používá téměř každý uživatel stolního počítače, aniž by o tom věděl. Řadí se totiž mezi nejrozšířenější a nejvíce využívané multiplatformní knihovny a používá ho mnoho obecně známých aplikací:

- linuxové desktopové prostředí KDE,
- matematický softwarový balík Mathematica,
- aplikace pro internetovou komunikaci Skype (starší verze),
- Google Earth, VirtualBox, Autodesk aj.

Qt je mimo GNU LGPL dostupné také v komerčních licencích, za které musí koncový zákazník zaplatit. Odměnou mu je kvalitnější podpora ze strany výrobce.

V případě komerční licence stačí poskytnout exe soubor, u softwaru, co byl poskytnut zdarma, se musí přiložit i kód, pokud bylo zasaženo do implementovaných knihoven a kódů samotných Qt.

Qt 5 bylo představeno v zimě roku 2012 a přineslo spoustu nových prvků a zaměření. Jelikož Qt 4 již bylo staré sedm let a používání počítačů se během těchto let významně změnilo bylo zapotřebí zavést novou verzi, která je použitelná pro nové zařízení, jejichž množství stále narůstalo. Jednalo se o mobilní aplikace, které měly ovládání obstarány dotykovými displeji.

1.1.1 Qt Creator

Qt Creator je multiplatformní vývojové prostředí pro C++, JavaScript a QML, které je součástí SDK pro Qt GUI aplikační vývojový framework. Obsahuje vizuální ladící program a integrované grafické rozhraní a návrhář forem. Funkce editoru zahrnují zvýraznění syntaxe a automatické dokončování. Qt Creator používá kompilátor C++ z GNU Compiler Collection na Linuxu a FreeBSD. V systému Windows může používat MinGW nebo MSVC s výchozí instalací a může také použít nástroj Microsoft Console Debugger při kompilaci zdrojového kódu[6].

1.1.2 Qt Quick

Qt Quick je zastřešujícím termínem pro technologii uživatelského rozhraní používanou v Qt 5. Qt Quick je skupina několika technologií:

- QML značkovací jazyk pro uživatelská rozhraní
- JavaScript dynamický skriptovací jazyk
- Qt C++ vysoce přenosná vylepšená knihovna C++[5]

Modul Qt Quick je standardní knihovna pro psaní QML aplikací. Zatímco modul Qt QML poskytuje QML engine a jazykovou infrastrukturu, modul Qt Quick poskytuje všechny základní typy potřebné pro vytváření uživatelských rozhraní s QML. Poskytuje vizuální plátno a obsahuje typy pro vytváření a animaci vizuálních komponent, příjem uživatelského vstupu, vytváření dotykových modelů, zobrazení a zpoždění instancí objektu.

Qt Quick je snadno rozšiřitelný s vlastním nativní funkcí pomocí Qt C++. Stručně řečeno, deklarantní uživatelské rozhraní se nazývá front-end a nativní části se nazývají back-end. To vám umožní oddělit intenzivní a přirozený provoz vaší aplikace z části uživatelského rozhraní[5].

Modul Qt Quick poskytuje rozhraní QML API, které dodává QML typy pro vytváření uživatelských rozhraní s jazykem QML a C++ API pro rozšíření QML aplikací s kódem C++. Qt Quick poskytuje vše potřebné k vytvoření bohaté aplikace s plynulým a dynamickým uživatelským rozhraním. Umožňuje uživatelským rozhraním být postaveným kolem chování součástí rozhraní a vzájemném propojení a poskytuje vizuální plátno se svým vlastním souřadnicovým systémem a vykreslovacím modulem. Animační a přechodové efekty jsou koncepty první třídy v Qt Quick a vizuální efekty mohou být doplněny speciálními komponentami pro efekty částic a shaderů[8].

1.1.3 QML

Podobně jako HTML, QML je značkovací jazyk. Skládá se ze značek nazývaných prvky v Qt Quick uzavřených složených závorkách. Je to deklarativní jazyk podobný CSS a JSON pro tvorbu aplikací s uživatelským rozhraním, který umožňuje jejich popis z hlediska vizuálních komponent a vzájemných vztahů a interakcí. Jedná se o vysoce čitelný jazyk, který byl navržen tak, aby umožnil vzájemné propojení komponent dynamickým způsobem a umožnil snadné opětovné použití a přizpůsobení komponent v rámci uživatelského rozhraní. Pomocí modulu Qt Quick mohou návrháři a vývojáři v QML snadno vytvářet animovaná uživatelská rozhraní a mají možnost připojit tato uživatelská rozhraní k libovolným knihovnám C++[2].

Qt Quick se často používá pro mobilní aplikace, kde jsou rozhodující dotykové vstupy, plynulé animace (60 FPS) a uživatelský zážitek. QML se také používá s Qt3D k popisu 3D scény a metodologie rámcového vykreslování. Dokument QML popisuje hierarchický strom objektů. Mezi QML moduly dodávané s Qt patří primitivní grafické stavební bloky (např. Rectangle, Image), komponenty modelování

(např. FolderListMode, XmlListModel), komponenty chování (např. TapHandler, DragHandler, State, Transition, Animation) a ovládání (např. Button, Slider, Drawer, Menu). Tyto prvky je možné kombinovat tak, aby vytvářely komponenty od jednoduchých tlačítek a posuvníků až po dokončené internetové programy.

Prvky QML mohou být rozšířeny o standardní JavaScript, a to jak v řádku, tak prostřednictvím zahrnutých souborů .js. Elementy mohou být také bezproblémově integrovány a rozšiřovány komponentami C++ pomocí Qt frameworku.

```
1 Button {
    id: quitButton; label: qsTr("Quit");
    onClicked: Qt.quit()
    anchors.horizontalCenter: parent.horizontalCenter
  }
```

Zdrojový kód 1.1: Blok QML kódu

1.1.4 Signály a sloty

Důležitou vlastností Qt toolkitu je přítomnost signálů a slotů[7] pro komunikaci mezi objekty. Mechanismus signálů a slotů je hlavní prvek Qt a část, která se nejvíce liší od funkcí jiných frameworků. Tyto rozšiřující prvky fungují díky meta-objektovému systému Qt.

Při programování grafického rozhraní, když změníme jednu grafickou komponentu, tak často požadujeme, aby byla jiná komponenta upozorněna. Obecněji chceme aby objekty jakéhokoli typu byly schopné komunikovat mezi sebou. Například pokud uživatel stiskne tlačítko "Zavřít", tak pravděpodobně chceme, aby se zavolala funkce pro zavření okna.

Jiné sady nástrojů dosáhly tohoto druhu komunikace využitím tzv. callbacků. Callback je ukazatel na metodu objektu, kterou chceme vyvolat po nějaké události jiného objektu. Zatímco úspěšné frameworky používající tuto metodu existují, callbacky mohou být neintuitivní a mohou trpět problémy při zajištění správnosti typů argumentů callbacku.

Sloty a signály mohou být využity ve všech objektech, které jsou přímo nebo nepřímo zděděny ze třídy QObject. Při propojování signálů a slotů může být s jedním slotem spojeno několik různých signálů a stejně tak na jeden signál napojeno několik slotů. Sloty mohou být použity pro přijímání signálů a zároveň mohou být použity jako standardní metoda objektu.

V Qt jsou alternativou ke callbackům popisované signály a sloty. Signál je vyslán když nastane konkrétní událost. Qt widgety mají mnoho předdefinovaných signálů, ale můžeme vždy vytvořit podtřídu widgetu abychom do ní vložily své vlastní signály. Slot je funkce, která je volána v reakci na určitý signál. Qt widgety mají také předdefinované sloty, ale je běžné vytvořit jejich podtřídu a přidat své vlastní sloty, tak že můžeme reagovat na signály které nás zajímají.

Mechanismus signálů a slotů je typově bezpečný, tzn. že podpis signálu musí odpovídat podpisu přijímacího slotu. (Ve skutečnosti může mít slot kratší podpis

než signál, který obdrží, protože může ignorovat další argumenty.) Vzhledem k tomu, že podpisy jsou kompatibilní, kompilátor nám může pomoci při zjišťování nesouladů typů při použití syntaxe ukazatelů na funkce. Syntaxe SIGNAL a SLOT rozpozná nesprávné typy za běhu. Signály a sloty jsou volně spojeny: Třída, která vydává signál, neví ani se nezajímá, které sloty přijímají signál. Signály Qt a sloty zajišťují, že pokud připojíte signál do slotu, bude slot ve správném čase vyvolán parametry signálu. Signály a sloty mohou mít libovolný počet argumentů jakéhokoliv typu. Jsou zcela bezpečné.

Všechny třídy které dědí z QObjectu nebo jedné z jeho podtříd (např. QWidget) mohou obsahovat signály a sloty. Signály jsou vyslány objektem když změní svůj stav na takový, který zajímá jiné objekty. Toto co je objekt nutný vykonat pro komunikaci. Objekt neví a ani se nestará jestli něco přijímá signál, který vyslal. Jedná se o zapouzdření informací, který zajišťuje, že objekt může být použit jako softwarová součást.

Sloty mohou být užité pro příjem signálů, ale jedná se také o normální členské funkce. Tak jako objekt neví jestli jiná komponenta obdržela jeho signál, tak i slot neví jestli jsou k němu připojeny signály. Tento způsob zajišťuje že mohou být v Qt vytvořeny zcela nezávislé součásti.

Do jednoho slotu může být připojeno více signálů a signál může být připojen do kolika slotů je zapotřebí. Dokonce je možné připojit signál přímo do jiného signálu. Tento způsob vyšle druhý signál okamžitě jakmile byl vyslán první.

Společně tvoří signály a sloty silný programovací mechanismus komponent.

Signály

Signály jsou vyslány objektem, když se jeho vnitřní stav nějakým způsobem změnil, který může být zajímavý pro klienta nebo vlastníka objektu. Signály jsou veřejně přístupné funkce a mohou být vysílány odkudkoliv, ale je vhodné vydávat signál pouze ze třídy, která definuje signál a její podtříd.

Když je signál vyslán, sloty připojené k němu jsou obvykle okamžitě spuštěny, stejně jako normální funkční volání. Když nastane tato situace, mechanismus signálů a slotů je zcela nezávislý na jakékoliv smyčce událostí GUI. Provedení kódu, který byl přerušen vyslaným příkazem, bude spuštěn ve chvíli, kdy se všechny sloty ukončily. Situace je poněkud odlišná při použití spojení ve frontě. V takovém případě bude kód, nacházející se za vyslaným signálem, pokračovat okamžitě a sloty budou provedeny později.

Pokud je k jednomu signálu připojeno několik slotů, sloty budou prováděny jeden po druhém v pořadí, v jakém byly připojeny, když je signál vysílán.

Signály jsou automaticky generovány modulem moc a nesmí být implementovány v souboru .cpp. Nikdy nemohou mít návratové typy (tzn. použití void).

Ze zkušenosti jiných se poukazuje na to, že signály a sloty jsou více použitelné, pokud nepoužívají speciální typy. Pokud by *QScrollBar::valueChanged()* používal speciální typ, například hypotetický QScrollBar::Range, mohl by být připojen pouze k slotům určeným speciálně pro QScrollBar. Spojení různých vstupních widgetů by bylo nemožné.

Sloty

Slot je vyvolán ve chvíli kdy signál, který je k němu připojen, byl vyslán. Sloty jsou normální C++ funkce a mohou být standardně volány. Jejich jedninou speciální vlastností je, že k nim mohou být připojeny signály.

Vzhledem k tomu, že sloty jsou normální členské funkce, tak se řídí klasickými C++ pravidly, když jsou volány na přímo. Jako sloty však mohou být vyvolány libovolnou komponentou, bez ohledu na její úroveň přístupu, přes spojení signál-slot. To znamená, že signál vysílaný z instance libovolné třídy může způsobit vyvolání privátního slotu v instanci nesouvisející třídy.

Lze také definovat sloty, které mají být virtuální, což je v praxi velmi užitečné.

Porovnání

Ve srovnání s callbacky jsou signály a sloty o něco pomalejší z důvodu zvýšené flexibility, kterou poskytují, ačkoliv rozdíl v reálných aplikacích je zanedbatelný. Obecně platí, vyslání signálu, který je připojen k některým slotům, je přibližně desetkrát pomalejší než přímé volání přijímačů s nevirtuálními funkcemi. Toto jsou režijní požadavky k lokaci objektového spojení a bezpečné iteraci přes všechna spojení (např. kontrola následujících přijímačů, zda nebyly odstraněny během vyslání signálu). Zatímco deset nevirtuálních funkčních volání se může zdát jako velké množství, je to mnohem méně režie než volání nové nebo mazání operace. Jakmile je provedena operace s řetězcem, vektorem nebo listem, která za scénou potřebuje funkci new nebo delete, režie signálů a slotů je zodpovědná pouze za malou část těchto nákladů pro volání. To samé platí vždy, když se provádí systémové volání ve slotu nebo když se nepřímo volá více jak deset metod. Jednoduchost a flexibilita mechanismu signálů a slotů stojí za dodatečnou režii, které si uživatelé ani nevšimnou.

```
connect(button, SIGNAL(clicked()),this,SLOT(OnClicked()))
```

Zdrojový kód 1.2: Ukázka propojení signálu a slotu

1.2 GTK+

GTK+ původně GIMP Toolkit je v informačních technologiích sada knihoven určených pro běh programů v grafickém uživatelském rozhraní. Knihovna původně vznikla pro potřeby grafického rastrového editoru GIMP a byla poté použita pro prostředí GNOME. Velmi rychle se tak stala jednou ze dvou nejpopulárnějších knihoven a spolu s knihovnami Qt nahradila dříve používané knihovny Motif. GTK+ je šířeno jako open source software s licencí LGPL jako součást projektu GNU.

GTK+ vytvořili v roce 1997 členové skupiny eXperimental Computing Facility.

Knihovna GTK+ obsahuje sadu grafických ovládacích prvků (widgety). GTK+ je objektově orientovaný widget toolkit napsaný v programovacím jazyce C; používá objekt GObject, tj. objektový systém GLib, pro objektovou orientaci. Zatímco

GTK+ je primárně zaměřen na systémy pro vytváření oken na bázi X11 a Wayland, pracuje i na jiných platformách, včetně Microsoft Windows (propojených s Windows API) a macOS (propojených s Quartz). Má to také back-end HTML5 nazvaný Broadway.

GTK+ může být nakonfigurován tak, aby změnil vzhled kreslených widgetů; to se provádí pomocí různých zobrazovacích enginů. Existuje několik zobrazovacích nástrojů, které se pokoušejí napodobit vzhled nativních widgetů na použité platformě.

Počínaje verzí 2.8, která byla vydána v roce 2005, zahájil GTK+ přechod k použití Cairo, aby vystihl většinu svých grafických prvků. Od GTK+ verze 3.0, všechno vykreslování se provádí pomocí Cairo.

GTK+ je převážně vyvíjen GNOME Project, který taktéž vyvíjí GNOME Development Platform a GNOME Desktop Environment.

Vývoj GTK+ je volně řízen. Vývojáři a uživatelé GNOME se scházejí na každoročním setkání GUADEC, kde se diskutuje o současném stavu a budoucím směřování GNOME. GNOME obsahuje standardy a programy od freedesktop.org, aby lépe spolupracoval s ostatními desktopy.

GTK+ je napsáno převážně v C. K dispozici je několik jazykových vazeb.

Nejběžnější kritika GTK+ je chybějící zpětná kompatibilita ve významných aktualizacích, zejména v API.

Chyby kompatibility mezi menšími aktualizacemi během vývojového cyklu GTK+ 3.x byly zaviněny silným tlakem inovací, jako poskytování prvků vyžadovanými moderními uživateli a podporou stále vlivnějšího Wayland display server protokolu. S vydáním GTK+ 4, tlak z potřeby inovací povolil a rovnováha mezi stabilitou a inovacemi se bude blížit vršku stability[9].

1.3 Electron

Electron je otevřená zdrojová knihovna vyvinutá společností GitHub pro budování multiplatformních desktopových aplikací s využitím HTML, CSS a JavaScript. Jeho architektura je poměrně jednoduchá, Electron využívá kombinace Chromium pro frontend a Node.js pro backend. Tento koncept přijalo mnoho společností, aby poskytlo širší podporu operačním systémům spolu se snížením nákladů na vývoj. Některé aplikace, které jsou implementovány pomocí tohoto frameworku, jsou Slack, Skype, WhatsApp, Signal, Discord, Twitch, VS Code, Atom, Bitwarden a GitHub Desktop. Aplikace mohou být baleny pro Mac, Windows a Linux.

Electron začal v roce 2013 jako rámec, na kterém bude postaven Atom, GitHubův hackovatelný textový editor. Tyto dvě byly otevřeny na jaře 2014.

Electron umožňuje vytvářet desktopové aplikace s čistým JavaScriptem tím, že poskytuje runtime s bohatými nativními API. Můžete ji vidět jako variantu runtime Node.js, která je zaměřena na desktopové aplikace místo webových serverů.

2 Současná řešení

Měřicích přístrojů je v dnešní době velké množství. Můžeme je řadit dle jejich velikosti a použití. K dispozici jsou přenosné měřicí přístroje pro měření elektrických i neelektrických veličin, stolní přistroje pro použití ve vývojových laboratořích, zkušebnách pro výuku a speciální aplikace. Přístroje pro elektroakustiku, testování AD/DA převodníků, měření kvality TV signálů apod. Vícekanálové měřicí systémy pro sběr a záznam technologických veličin v laboratořích i ve výrobě, zapisovače rychlých dějů pro aplikace v energetice. Jak je vidět měřicí přístroje se používají v různorodých odvětvích a neexistuje jedna aplikace, která by uměla pracovat se všemi druhy přístrojů. Software, který již existuje, tak je spíše zaměřen na dané typy přístrojů nebo pouze obsluhují přístroje své vlastní značky. Níže jsou popsané některé z těchto aplikací.

2.1 HMGWin 3000

Software byl vyvinut pro zpracování a vyhodnocení naměřených dat, které byly zaznamenány pomocí zařízení HYDAC, jež také vyvinula tento software.

2.1.1 Požadavky

Program HMGWin 3000 je velice nenáročný a v dnešních podmínkách bude funkční na jakémkoli osobním počítači.

- Pentium 400 MHz, 256 MB operační paměti
- Operační systém Windows XP / 2000
- 3 MB volného prostoru na HDD
- RS232 rozhraní, RS232 konektor, USB 1.1
- Grafické rozlišení 640 x 480, 256 barev nebo vyšší

Nutné nainstalovat USB řadič. Program je taktéž uzpůsoben pro novější operační systémy a je možné jej spustit i na Windows Vista a 7.

2.1.2 Rozhraní RS-232

Jedná se o sériový port, který se používá jako komunikační rozhraní osobních počítačů a další elektroniky. RS-232 umožňuje propojení a vzájemnou sériovou komunikaci dvou zařízení, tzn., že jednotlivé bity přenášených dat jsou vysílány postupně za sebou (v sérii) po jednom páru vodičů v každém směru. Na rozdíl od síťové technologie Ethernet nebo rozhraní USB se tedy jedná o zcela bezkolizní fyzickou vrstvu.

Jedná se o technologii od které se téměř úplně odstoupilo mimo průmyslu, protože ji nahradilo výkonnější univerzální sériové rozhraní (USB).

Na počítači bývá linka RS-232 vyvedena pomocí konektoru D-Sub typu DE-9 M (samec), zařízení se tedy připojuje šňůrou s konektorem DE-9 F (samice). U starších počítačů byla druhá linka vyvedena na konektor DB-25 M (ten doporučuje původní norma), používal se například pro připojení modemu. Elektricky jsou oba konektory shodné (u velkého je jen mnoho pinů nevyužitých), takže se mohla případně použít jednoduchá pasivní redukce na DE-9 M a teoreticky i naopak. Pro připojení zařízení používajících RS-232 k současným počítačům se používají buď rozšiřující desky, nebo převodníky USB/RS-232. Převodníky USB/RS-232 mají proti originální „skutečné“ lince RS232 výrazně delší dobu odezvy, což může v některých aplikacích způsobovat značné problémy až nefunkčnost. Ačkoliv moderní základní desky většinou nemají sériový port na zadním panelu, mohou ho některé mít vyveden na 10-pinový konektor na jiném místě na desce (podobně jako „interní“ USB).

2.1.3 Vybavení programu

Program má možnost změny jazyku mezi němčinou, angličtinou a francouzštinou. Automatické hledání HMG zařízení připojených k počítači. Při vyhledání a nalezení dále vypíše na jakém portu se dané zařízení nachází. Možnosti použití: V softwaru je možné zobrazovat naměřené hodnoty a ukázání naměřeného minima a maxima. Ze získaných hodnot je dále možné sestavit křivku. K dispozici máme několik pohledů.

- Graf: zobrazuje křivku naměřených hodnot. Čas je zobrazen na ose X, naměřené hodnoty na ose Y. Křivky jsou barevně rozlišeny.
- Tabulka: ukazuje naměřené hodnoty jednotlivých senzorů v určitém časovém rozmezí.
- Nahraná data: typ měřicí křivky, počet snímačů, počet záznamů dat, počáteční a konečný čas měření atd. Kanály s jejich rozsahy měření.
- Popis: označení a poznámky zadané pro křivku. Jak označení tak poznámky lze zde změnit nebo přidat.

Graf

V grafu je možné zobrazit hodnoty z křivky. Po označení dvou bodů můžeme změřit jejich rozdíl. V grafu je možné nastavit časový bod, svislou přímku, kterou pohybuje

podle vlastního uvážení a procházíme jednotlivé hodnoty grafu. Také se dají nastavit tyto přímky dvě a sledovat rozdíly mezi nimi.

Mezi vybavení programu patří i možnost přiblížení (zoom). Vyříznutí chtěné části (panning), zobrazení na celou obrazovku, automatické škálování. Tvoření poznámek a jejich přichycení k jednotlivým hodnotám. Procházení listu poznámek s možností úprav. Vytvoření snímků obrazovky. Dialogy pro otevírání a ukládání souborů. Tisk.

Editace

Extrahování hodnot z grafu a jejich editace bez zasahování do původních dat. Překrývání křivek. Překrytí dvou měřicích křivek je užitečné pro porovnání měření, např. měření cyklu stroje pořízeného před třemi měsíci ve srovnání s jeho současným stavem. Posunutí kanálu v čase. Přidání dopočítaného kanálu.

2.2 SensoWin

Počítačový software SensoWin je jednoduše ovladatelný softwarový balíček, který umožňuje čtení a zpracování měřených hodnot grafů. Hodnoty musí být měřené přístroji Parker Serviceman Plus, Parker Service Master Easy nebo Parker Service Master Plus. Aplikace je spustitelná pouze pod operačním systémem Windows.

2.2.1 Funkce

Křivky mohou být zobrazeny v diagramu. Graf umožňuje přesnou analýzu hydrauliky. K vyhodnocení výkonu čerpadla lze vytvořit křivku výkonu. Úniky a tlakové ztráty mohou být detekovány generováním funkce rozdílové hodnoty.

Pomocí kurzoru lze zjistit časově závislý hydraulický postup. Pro každou křivku jsou uvedeny podrobné informace. To znamená, že měření provedená s Parker Serviceman Plus, Parker Service Master Easy nebo Parker Service Master Plus mohou být kdykoliv reprodukována. Změna váhy a jednotek umožňuje pozdější úpravu pro zobrazení v diagramu. Tabulkové znázornění hodnot ACT, MIN a MAX, vyhlazování křivky měření a matematických vazeb jsou důležitými funkcemi při analýze hydraulického systému.

Datum a čas jsou dokumentovány při každém měření. To značně usnadňuje pozdější přidělování hodnot. V aplikace je umožněn přímý přenos naměřených hodnot z Parker Serviceman Plus, Parker Service Master Easy nebo Parker Service Master Plus do PC.

Aktuální události (tlakové špičky atd.) Jsou viditelné, když je proces spuštěn (online funkce).

Podle typu zakoupeného softwarového balíku jsou k dispozici následující prvky:

- číselný, sloupcový, křivkový graf,
- současné zobrazení 16 kanálů,
- osciloskop,

- funkce přiblížení,
- funkce výpočtu,
- funkce analýzy,
- připojení pomocí USB nebo Ethernetu,
- online zobrazení měřených hodnot,
- ukládání projektů
- a jejich export do Excelu.[10]

3 Analýza

Prvotním krokem při produkci nového softwaru je analýza. Analýza dává návrhářům určitý obraz o tom, jak bude software vypadat a co budou jeho hlavní funkce. Během analýzy se určují dané požadavky a specifikace na software, co je zapotřebí aby program uměl, co je jeho cílem a jakou skupinu uživatelů má zasáhnout. Výsledkem analýzy je zpráva zachycující veškeré požadavky a také umožňuje náhled do rizikových míst a jejich případnému předejití.

V kapitole jsou nejdříve popsány požadavky na aplikaci a jejich analýza a pro jakou vrstvu uživatelů bude program navržen.

3.1 Požadavky

Jak se uvádí v Základech systémového inženýrství[13], požadavky na software se dají rozdělit do několika kategorií. Pro mé účely postačí požadavky klasifikovat do tří skupin, požadavky funkční, nefunkční a chování. Analýza požadavků je dosažena konzultací na schůzkách s vedoucím práce ve spojení se zadáním práce. Jedná se o náhled jakým směrem by se aplikace měla vyvíjet, aby se zachovaly veškeré důležité prvky.

Nefunkční požadavky jsou požadavky, které specifikují kritéria, která mohou být použita k posouzení provozu systému, nikoli specifického chování. Vychází z implementace prostředí a použitých měřicích přístrojů.

Funkční požadavky definují nezbytné úkoly, aktivity či činnosti, které musí být splněny. Funkční požadavky mohou zahrnovat výpočty, technické detaily, manipulaci a zpracování dat a další specifické funkce, které definují, čeho má systém dosáhnout.

Požadavky chování popisují všechny případy, kdy systém používá funkční požadavky, které jsou zachyceny v use case.

3.1.1 Nefunkční požadavky

Multiplatformní architektura definuje operační systém nebo skupinu operačních systémů, na kterých bude aplikace spustitelná. Aplikace musí být spustitelná alespoň na jednom z operačních systémů Microsoft Windows nebo Linux. Vzhledem k nutnosti využití určitého balíčku pro komunikaci, který byl

na počátku vývoje dostupný pouze pro operační systém Linux, byla pro prototyp aplikace využita tato platforma.

C++ programovací jazyk je zvolen z důvodu již napsané C++ knihovny, která zprostředkovává spojení s měřicími přístroji některých výrobců. Z hlediska pracování s větším množství dat, je využití tohoto jazyka i výhodnou volbou.

Měřicí přístroj je jednou z nutných komponent pro testování aplikace v provozu. Jelikož je aplikace zamýšlena jako grafické rozhraní vícekanálových měřicích systémů, tak je zapotřebí míti přístroj dostupný na školní síti.

3.1.2 Funkční požadavky

Navázání spojení s přístrojem je základním kamenem pro realizaci zadaného softwaru. Bez tohoto připojení, by nebyly žádná data a nebylo by co zobrazit v grafech. Po navázání spojení nám použité knihovny umožní přístup do přístroje, nahlédnutí do konfigurace, sběr dat a změny v konfiguraci.

Vykreslení grafů a jejich ovládání slouží k reprezentaci měřených hodnot. Měřicí přístroje neměří pouze jednu veličinu, takže je pravděpodobné, že bude zapotřebí více než jeden graf s několika křivkami. Pro zpracování grafů mi posloužila knihovna z vývojového prostředí Qt Charts. V předešlém rozhodnutí ohledně použití frameworku Qt pro vývoj aplikace se tímto způsobem velice zjednodušila implementace grafů do GUI aplikace, jelikož je většina důležitých prvků pro tvorbu potřebných grafů již v knihovně.

3.1.3 Požadavky chování

Model pro požadavky chování softwaru se podobá tzv. use case modelu, neboli modelu pro případy použití. Dle příspěvku[11] se jedná o techniku modelování softwaru, která definuje funkce, které mají být implementovány, a řešení všech chyb, se kterými se lze setkat. Use case definují interakce mezi externími aktéry a systémem, popřípadě mezi dvěma systémy, k dosažení konkrétních cílů. Existují tři základní prvky, které tvoří use case:

- **Aktéři** Aktéři jsou typem uživatelů, kteří pracují se systémem.
- **Systém** Use case zachycuje funkční požadavky, které určují zamýšlené chování systému.
- **Cíle** Use case jsou obvykle iniciovány uživatelem aby splnili cíle popisující aktivity a varianty, které jsou nutné pro dosažení cíle.

Use case jsou modelovány pomocí UML (Unified Modeling Language) a jsou reprezentovány ovály obsahujícími názvy use case. Aktéři jsou reprezentováni pomocí řádků s názvem aktéra napsaným pod řádkem. Pro reprezentaci účasti aktéra v systému je mezi aktérem a use case nakreslena čára.

Charakteristiky use case modelu jsou:

- organizování funkčních požadavků,
- modelování cílů interakce systému s uživatelem,
- záznam scénářů id spouštěcích událostí až po konečné cíle,
- popis základního průběhu akcí a mimořádného toku událostí,
- povolení přístupu uživatele k funkcím jiné události.

Kroky při návrhu use case modelu jsou:

- Určit uživatele systému.
- Pro každou kategorii uživatelů vytvořit uživatelský profil. To zahrnuje všechny role dané uživatelům, které jsou relevantní pro systém.
- Identifikovat významné cíle spojené s každou rolí na ztužení systému.
- Vytvořit use case pro každý cíl spojený s šablonou use case a udržovat stejnou úroveň abstrakce v celém use case.
- Strukturovat use case.

Vytvořením modelu se dosáhne konkrétních cílů kladených na software.

Po pouhém nahlédnutí do use case diagramu je ihned zřetelné, že již při návrhu modelu softwaru bude využita MVC (model-view-controller) struktura. MVC je návrhový vzor, který se často využívá při vytváření uživatelského rozhraní.

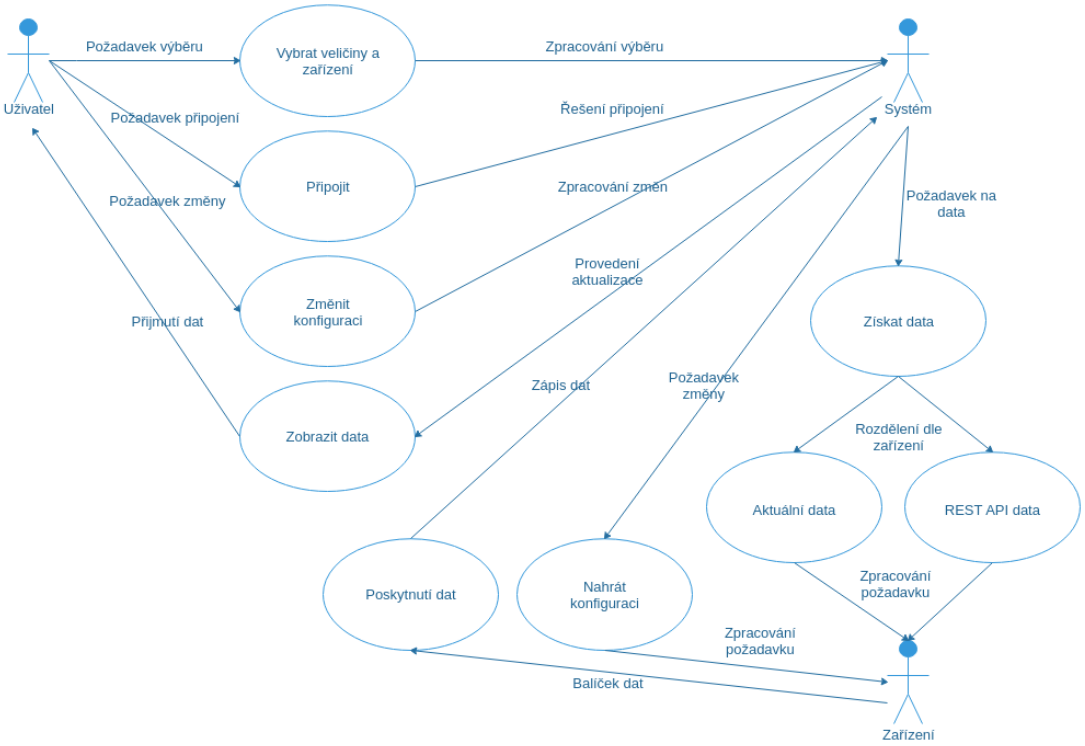
MVC se skládá ze tří druhů objektů. Model je objekt aplikace, View je jeho prezentací obrazovky a Controller definuje způsob, jakým uživatelské rozhraní reaguje na vstup uživatele. Před MVC návrhy uživatelského rozhraní byly návrhy spíše spojená skupina těchto objektů bez oddělení. MVC je odděluje pro zvýšení flexibility a opětovného použití.

Když jsou spojeny View a Controller, výsledným produktem je model-view architektura. Tento design stále odděluje způsob, jakým jsou data uložena, od způsobu, jakým jsou prezentována uživateli, ale poskytuje jednodušší framework založený na stejných principech. Toto oddělení umožňuje zobrazit stejná data v několika různých pohledech a implementovat nový typy pohledů bez změny podkladových datových struktur. Pro umožnění flexibilní manipulace s uživatelským vstupem je zde nasazen koncept delegáta. Výhodou toho, že v tomto frameworku je delegát, je to, že umožňuje, aby byly položky dat vykreslovány a upravovány tak, aby byly nastavitelné.

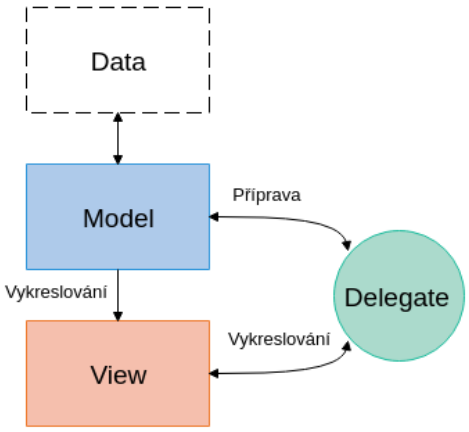
V obrázku 3.2 je zobrazen základní princip model-view architektury. Model komunikuje se zdrojem dat a poskytuje rozhraní pro ostatní komponenty v architektuře. Povaha komunikace závisí na typu zdroje dat a způsobu implementace modelu.

View získá modelové indexy, jedná se o odkazy na údaje. Dodáním modelových indexů view může načíst položky ze zdrojových dat.

Ve standardních zobrazeních delegát vykreslí datové položky. Když je položka upravována, delegát komunikuje s modelem přímo pomocí modelových indexů [12].



Obrázek 3.1: Vytvořený use case diagram.



Obrázek 3.2: Struktura model-view.

3.2 Případy užití

Aplikace je určena pro techniky, kterým je pomocí vizualizace dat do různých typů grafů (koláčový, spojnicový apod.) umožněno nahlédnout do aktuálně měřených (resp. historických) dat. Vizualizaci je možné nastavit pomocí výběru pouze zajímavých hodnot a jejich následném ořezu.

4 Návrh

Po analýze (viz 3) je dalším hlavním prvkem při vývoji softwaru návrh aplikace. Při návrhu aplikace se již zabýváme předvídáním a definováním softwarových řešení pro jednu nebo více sad problémů.

Hlavní rozdíl mezi analýzou softwaru a návrhem je, že závěrem softwarové analýzy je soubor problémů, které je třeba řešit. Analýza by se neměla lišit v závislosti na pracovní skupině nebo členech týmu. Naproti tomu se návrh zaměřuje na možnosti, jak dané sady problémů řešit. Pro jeden daný problém může a bude existovat více návrhů. V závislosti na prostředí se návrh často liší, ať už je vytvořen ze spolehlivých frameworků nebo je implementován vhodnými konstrukčními vzory. Příklady návrhů zahrnují operační systémy, webové stránky nebo mobilní zařízení.

4.1 Jazyk programu

Programovací jazyk dle nefunkčních požadavků 3.1.1 je definován jako C++. Použitím frameworku Qt však je tento jazyk rozšířen o další funkční prvky jako je komunikace mezi objekty při použití makra. Komunikaci mezi aplikací a přístrojem bude uskutečňovat knihovna libKMBPublic. C++ je také nejvhodnější řešení vzhledem k povaze aplikace, která musí zpracovávat data, která se mění každou sekundu, jelikož bývá i rychlý na méně výkonných počítačových sestavách. Jazyk lze použít k objektově orientovanému programování, které se stalo normou vývoje aplikací.

Při vývoji grafického rozhraní aplikace se jednotlivé prvky tvoří pomocí QML značkovacího jazyku. V kódu pro grafickou část bude použit i programovací jazyk JavaScript, který bude mít na starosti iteraci přes objekty, tvorbu dynamického počtu prvků, zpracování požadavků nebo jejich odeslání do C++ části.

4.2 Rozdělení aplikace

Pro snazší postup při tvorbě návrhu a pozdější implementaci je vhodné aplikaci rozdělit na jednotlivé části. Každá z těchto částí se bude již zaměřovat na jeden konkrétní problém nebo skupinu problémů, které jsou následně řešeny implementační třídou nebo skupinou tříd, metodami apod. Tyto třídy a metody jsou řešeny v kapitole 5. Ve společnostech pro vývoj softwaru je běžné software rozdělit na co nejjednodušší prvky, aby byl program jednoduše rozdělitelný do pracovních skupin. S ohledem k vývoji softwaru jednou osob bylo rozdělení aplikace zjednodušeno

a spousta prvků na sebe navazuje. Software byl rozdělen na tyto části:

- Datová struktura, která bude uskutečňovat komunikaci a uchování dat.
- Třídy pro možnost stažení JSONu z webové stránky.
- Grafické rozhraní, které obsahuje veškeré prvky, které bude uživatel moci použít.

Některé ze skupin problémů budou v dalších sekcích rozděleny na menší části.

4.2.1 Datová struktura

Datovou strukturou rozumíme třídu, která bude uchovávat veškeré měřené hodnoty, které jsou aktuálně měřeny nebo jsou nahrány pomocí souborů s daty. Třída bude přijímat a zprostředkovávat data jiným třídám, které o ně požádají. Jelikož přijímaná data nejsou pouze jedné struktury, ale získávají se z více vstupních typů, je zapotřebí tento problém řešit. Data budou přicházet z aktuálních měřených veličin z měřicího přístroje. Přístroj však data sbírá v nastaveném intervalu, takže když žádáme o data, tak nebudou přímo aktuální, ale budou to hodnoty z posledního měření. Tato data jsou zprostředkovány pomocí knihovny libKMBpublic, která umožňuje komunikaci s přístrojem. Data je však možné získat jako archivní JSON z webového rozhraní REST API. Datovou strukturu by bylo vhodné navrhnout tak, aby do ní bylo možné vložit data různých vstupů. Třída by měla být obecná natolik, že bude snadné rozšíření o další vstupní parametry.

Nejdříve bylo zapotřebí navrhnout jak data vůbec uchovat. Proto byla navržena struktura pro jednotlivé záznamy, tím se rozumí objekt, který v sobě uchovává časovou značku a veškeré veličiny, co byly v tento okamžik naměřeny. Každá veličina bude mít uchována své jméno, měřenou hodnotu a název fyzikální jednotky, ve které byla měřena. Jednotlivé veličiny budou uloženy v mapě, tak aby se nemohly opakovat stejné názvy, tzn. pro každou veličinu bude právě jedna položka v mapě. Použitím mapy se docílilo jednoduchosti v přístupu k jednotlivým prvků, přehlednosti a snadné rozšiřitelnosti o nové veličiny, o které může být přístroj rozšířen, nebo které naopak umí jiný typ přístroje.

Pro datový typ času jsem vybral tři vhodné kandidáty. Používáním knihovny libKMBpublic vzešla možnost použití třídy KMBTime, kterou knihovna poskytuje. Tato časová třída umožňovala převody do velkého množství časových tříd či struktur, které byly již navržené a byly často používané. Jednou z nich je například knihovna chrono. Ta byla taky druhou možností, jak přistoupit k danému problému. Využitím časové značky získáme hodnotu, která reprezentuje počet uběhnutých sekund od 1. ledna 1970 (UTC). Jedná se o nejjednodušší reprezentaci času. Poslední, třetí možností, je využít struktury vytvořené od používaného frameworku Qt. Třída QDateTime se zdála být nejlepší volbou z hlediska používání Qt frameworku, jelikož vytvořený objekt času bylo možné využít v dalších částech tvořené aplikace bez nutnosti použití převodníku na jinou třídu.

Tyto záznamy, ve kterých jsou veškeré naměřené hodnoty z dané časové značky, budou následně vloženy do vektoru, který uchovává všechny záznamy.

```
[
{
        "device_id":1,
        "tags":[
                "Status_Flagging_Freq",
                "Freq",
                "Status",
                "Flagging",
                "SMC_235_19"
        ],
        "timestamp":1519948800,
        "values": [
                221.97206115722656,
                224.1116180419922,
                224.18093872070312,
                224.4557342529297,
                 ...
        ],
        {
                "datatype":"Value",
                "propname":"U_avg_U1",
                "subvargroup":"avg",
                "type":"Float",
                "unit":"V",
                "username":"U1",
                "vargroup":"U"
        },
        "variable_id":1
}
]
```

Zdrojový kód 4.1: Ukázka přichozího JSONu

Získáváním dat z více zdrojů bylo zapotřebí datovou strukturu rozdělit. Rozdělením vznikla abstraktní třída (datastore), která nese proměnné, struktury a metody společné pro všechny třídy, které implementují abstraktní třídu. Mimo jiné vytvořením abstraktní třídy se také docílí jednoduché rozšířitelností, pokud bude zapotřebí přidat novou vstupní strukturu.

Abstraktní třída s sebou ponese metody pro komunikaci s jinými objekty, přístup do jednotlivých záznamů aj. Jednotlivé třídy, které implementují abstraktní, budou mít své vlastní metody a proměnné nutné pro chod celé aplikace.

4.2.2 Stahování

Stahování je myšleno přístup k REST API. Získávání aktuálních dat z měřicích přístrojů je sice implementováno přímo do datové struktury kvůli aktualizacím dat,

avšak toto řešení není možné použít pro JSON data, jelikož jsou neměnná. Dle zadané adresy se jednou stáhne balíček archivních dat a ty budou následně zobrazeny do grafů. Pro tyto účely jsou navrženy další dvě třídy, které se tímto zabývají a to třídu pro stažení JSONu a třídu pro samotné nahrání dat.

Třída pro stahování by měla obsahovat nějaký prostředek pro navázání spojení. Framework Qt již obsahuje knihovnu network, kterou pro tyto účely využiji. Třída pro nahrání staženého JSONu, čeká do chvíle, než třída pro stažení potvrdí příjem souboru a následně tento soubor zpřístupní GUI, ze kterého si uživatel vybere data jeho zájmu.

4.2.3 Grafické rozhraní

Samotné grafické rozhraní by mělo pouze zprostředkovávat komunikaci se zbytkem komponent a vykreslení dat v požadovaném formátu. Hlavní vrstva aplikace by měla obsahovat pole, do kterého se zadá připojovací adresa k zařízení. Adresa pro přístup přímo k zařízení je rozdílná od adresy pro stažení JSONu. Tato problematika bude vyřešená přepínačem na zvolenou adresu. Dále by bylo vhodné umožnit uživatelům ještě před připojením, jaké veličiny chtějí sbírat z měřicího přístroje a které chtějí mít zobrazené v grafu. Pokud by na tomto místě nebyla implementovaná žádná funkce pro redukci příchozích dat, tak by se veškerá data z aktuálních měřených hodnot zobrazovala do grafu. To by mělo za následek rychlé zahlcení systému a nečitelnost grafů, jelikož měřicí přístroje mohou zaznamenávat více jak 200 hodnot. Mít takové množství křivek v grafu je velmi nepřehledné a z uživatelského hlediska nepřijatelné, proto si uživatel musí zvolit takové křivky, které jsou pro jeho potřeby zajímavé. Po vyplnění příslušných údajů bude již možné připojit se k danému zařízení a data budou po intervalech nahrávána do datové struktury.

Po připojení se uživateli zpřístupní veškeré prvky aplikace založené na typu dat, které se rozhodl sledovat. Rozhraní pro aktuální data zpřístupní veškeré prvky aplikace. Tyto prvky jsou:

- nahlédnutí do konfigurace a možnost stažení archivních dat. Konfigurace představuje základní informace o zařízení jako je IP adresa, typ přístroje, sériové číslo, topologie zařízení apod.

- Zobrazení vybraných křivek do grafu. Samotný graf bude umožňovat funkci přiblížení, návrat do původního zobrazení a možnost skrytí jednotlivých křivek. Většinu z těchto akcí zařídí samotné objekty QML.

- Fázorový diagram měl být původním nápadem zpracován pomocí QML prvku ChartView, avšak zde nastal jeden zásadní problém. Tento prvek umožňoval použití koláčového, spojnicového, křivkového, plošného, bodového, sloupcového a polárního grafu. V žádné z těchto možností nebylo možné vytvořit spojnici pomocí úhlu a délky pro požadovaný výsledek. Jednou z možností bylo využit klasického spojnicového grafu a daný úhel přepočítat na požadovaný bod. Mezi tímto bodem a nulovým bodem následně protnout spojnici. Avšak tato spojnice by neměla žádnou možnost výběru, jaký koncový bod bude mít tvar.

Řešením bylo vytvořit vlastnoručně tento graf pomocí JavaScriptu a plátna a vytvořit tímto samostatný nový prvek do QML.

- Pro koláčový graf spotřeby se využije ChartView s tím, že bude nutné dopočítat chtěné hodnoty v grafu z měřených hodnot.

Ne všechny prvky aplikace jsou dostupné oběma typům vstupních struktur. Pro data přijatá z JSONu je umožněno pouze sledovat data v hlavním grafu měřených veličin.

5 Implementace

Samotná implementace se snaží vycházet z předchozích kapitol analýzy 3 a návrhu 4. Zde se budou řešit jednotlivé problémy, které vznikly při analýze a návrhu z hlediska implementování do aplikace, tak aby fungovala podle daných požadavků.

5.1 Datová struktura

Při implementaci datové struktury bylo objeveno hodně zásadních problémů a nebylo možné uskutečnit takový vzor tříd, jako bylo popsáno v návrhu 4.2.1. Zásadním problémem zde nastalo využití rozšířeného jazyka C++ od Qt. Přesněji se jedná o problematiku signálů a slotů (viz 1.1.4).

Použití abstraktní třídy nebyla špatná volba. Všechny metody, sloty a signály pracovaly tak, jak bylo zamýšleno. Problém nastal až u tříd, které implementovaly abstraktní třídu. U těchto tříd nebylo možné vytvoření vlastních signálů a slotů, které by komunikovaly s QML vrstvou grafických prvků. Všechny signály bylo nutné vytvořit v abstraktní třídě. U signálů to znamenalo pouze přesun z jedné třídy do druhé bez nutnosti dalších úprav kódu. Se sloty to již bylo obtížnější, jelikož slot který byl využit v třídě pro aktuální data neměl žádné zastoupení v třídě pro REST API. To znamenalo, že v abstraktní třídě byl vytvořený abstraktní slot, který byl definován pouze v jedné třídě (aktuální data nebo REST API) a v druhé měl vždy prázdné tělo. Do určitého množství metod je to udržitelné, ale u většího množství je to již problém, protože vzniká kód, který nemá žádnou funkcionalitu.

Závěrem tohoto problému bylo zrušení abstraktní třídy, která byla popsána v návrhu, a vytvoření dvou samostatných tříd, kde si každá nesla svou vlastní datovou strukturu, proměnné a funkce. Bohužel se tímto docílilo jisté duplikace kódu, ale v současném stavu nebylo nalezeno lepší a efektivnější řešení.

Mdatastore.h třída umožňuje čtení a zprostředkování reálných dat.

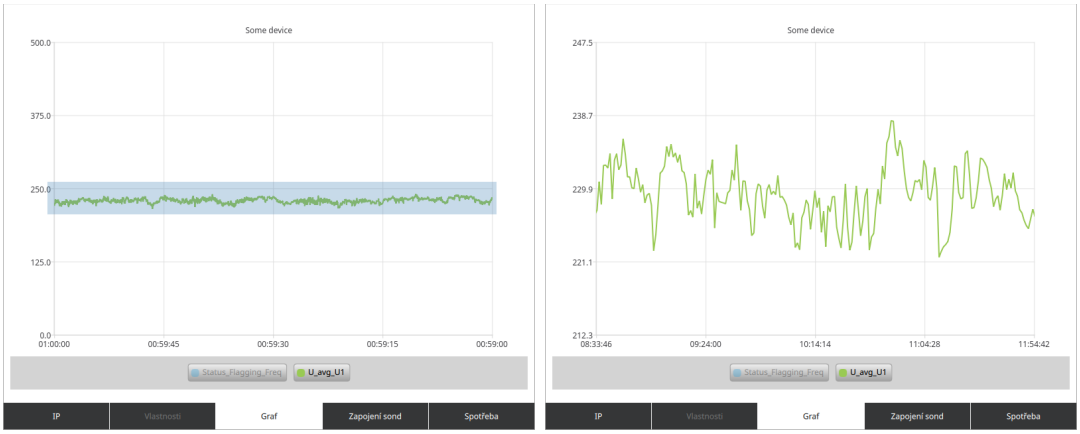
- *init()* je metoda, která převezme cestu k zařízení, kterou uživatel vyplnil v GUI. Funkce načte veškeré hodnoty, které uživatel zvolil, že chce snímat, společně s předvyplněnými veličinami pro fázorový diagram a koláčkový graf spotřeby. Zároveň metoda uloží do privátních proměnných údaje o konfiguraci zařízení. Dané nastavené snímané veličiny vytvoří prázdné křivky do grafu pro následné snímání hodnot.

- *getPieValues()* získá z privátních proměnných struktury pouze data potřebná pro funkci koláčového grafu spotřeby a zprostředkuje je GUI.

- *getPhasor()* dává přístup k jednotlivým fázorům, které jsou vykresleny ve fázorovém diagramu.
- *getConfiguration()* je podobného rázu s rozdílem vyplnění konfiguračních hodnot.
- *getSelected()* zaznamenává jednotlivé veličiny, které uživatel vybral.
- *update()* vyčte nová data z měřicího přístroje a rozšíří tak databázi hodnot.
- *createEntry()* vytvoří nový záznam hodnot, které byly získány volám funkce *update()*.
- *updateSeries()* doplní novou hodnotu na konec křivky po zavolané funkci *update()*.
- *setZoom()* umožňuje v grafu použití přiblížení zajímavých dat. Aplikace funkce je znázorněna v obrázku 5.1.
- *getArchive()* je vyvolán stisknutím tlačítka pro stažení CEA archívů uložených v přístroji a uloží je do složky ve které se nachází software.

Mjsondatastore.h drží data přijatá z JSONu.

- *init()* přijme ve vstupních proměnných JSON dokument a interval, ve kterém jsou data zaznamenány, a uloží je do privátních proměnných.
- *load()* funkce může být zavolána až poté, co byla zavolána funkce *init()*. Tato metoda postupně rozdělí JSON z privátní proměnné na menší celky, které nahraje do datové struktury.
- *getEntries()* zpřístupní kopii datové struktury. Funkce je důležitá pro třídy uvedené v sekci stahování 5.2
- *getTypes()* vytvoří kopii listu typů veličin, které byly v JSONu.
- *getDeviceId()* získá název zařízení.
- *setLineSeries()* nastaví veškerá data do křivky spojnicového grafu.
- *size()* vrátí velikost datové struktury, tzn. počet veličin v jednom záznamu.
- *name()* vrátí jméno veličiny.



Obrázek 5.1: Vlevo znázorněn výběr plochy k přiblížení. Po několika opakovaném přiblížení bylo dosaženo stavu grafu umístěného vpravo.

5.2 Stahování

V době psaní kódu pro třídy **load_handler.h** a **json_downloader.h** bylo zapotřebí použít Qt knihovny network. Tato část byla psaná na verzi Qt 5.11.2 a zastihl mě bug frameworku. V této používané verzi byl problém právě s knihovnou network a to ten, že využívala knihovny openssl pro své účely. Avšak zde nastal výše zmíněný problém v tom, že knihovnu bylo nutné mít v určité verzi, což vývojáři Qt neměli v úmyslu.

Pro řešení problému se vyskytly tři možnosti. Vyčkat na nový update, kde plánovali danou chybu opravit. Upravit framework tak, aby umožňoval práci s mojí verzí openssl knihovny, nebo stáhnout verzi knihovny, kterou v tuto chvíli Qt podporovalo. Řešení nakonec bylo velice jednoduché, jelikož jsem neměl nejnovější verzi frameworku, takže stačilo pouze nainstalovat novější verzi a knihovna network již fungovala s mojí verzí knihovny openssl.

Návrhem dvou tříd se ulehčila problematika stahování. **json_downloader.h** je třída, která pouze čeká na zavolání funkce *doDownload()* pro stažení souboru. Přístup k přijatým datům je zařízen pomocí signálů a slotů. Třída je spojená se signálem dokončení stahování, který je následovně propojen se slotem *downloadFinished()*, který naplní JSON přijatými binárními daty a daný dokument vyšle v signálu dál.

Třída **load_handler.h** úzce spolupracuje s třídou pro stahování. Obě třídy čekají na signál z druhé třídy. Počátečním bodem je vyplnění pole pro adresu, ze které se mají data přijmout. Jakmile je stisknuté tlačítko, je zavolána metoda *download()*, která zavolá funkci třídy **json_downloader.h** *doDownload()*. Následně slot *handleInput()* čeká na signál s JSON dokumentem. Po přijetí dokumentu se celý nahraje do datové struktury, která se zpřístupní GUI.

5.3 Grafické rozhraní

Grafické rozhraní je tvořené skupinou QML modulů společně s JavaScriptovými kódy, které zařizují volání signálů a slotů jak v QML tak mezi QML a C++.

Po spuštění aplikace se zobrazí úvodní obrazovka. V dolní části aplikace se nachází menu.

IP, kde si uživatel vybere jestli chce sbírat aktuální data nebo jestli chce stáhnout JSON archivní data. U aktuálních hodnot si uživatel vybere jaké hodnoty chce sledovat, vyplní adresu k zařízení a tlačítkem potvrdí připojení k měřicímu přístroji. Druhou možností je stažení JSON dat.

Dle zvolené možnosti QML signál *onClicked* vyhodnotí v jakém stavu jsou vyplněné prvky a na jejich základě se rozhodne, zda se připojí k měřicímu přístroji, nebo stáhne JSON dokument. Po provedení veškerých nutných akcí k získání veličin, tak se uživateli zpřístupní ostatní položky v menu aplikace.

Vlastnosti jsou vytvořeny pouze z textových bloků, které jsou po připojení k přístroji vyplněny konfigurací zařízení. Jedná se pouze o informativní položku.

Graf je rozdělen na několik částí. V grafu se řeší vykreslování jednotlivých křivek, které lze skrýt z dynamicky vytvořené legendy znázorněné v obrázku 5.2. Legenda se tvoří již při výběru veličin, které chce uživatel sledovat. Legenda grafu je samostatným modelem, který bylo nutné vytvořit kvůli funkci skrytí křivek. V tomto grafu je implementovaná funkce zoomu, která umožňuje v grafu výběr čtverce, na který se následně stane zobrazením grafu. V grafu bez přiblížení se každou sekundu obnovují data, které jsou měřené z měřicího přístroje. Když jsou data přiblížena, tak se křivka nerozšiřuje o další data, ty jsou pouze zapsaná do datové struktury. Jakmile uživatel dvojklikne na graf, tak se navrátí do původního zobrazení a do grafu se vyplní nová data, která byla pouze v datové struktuře.

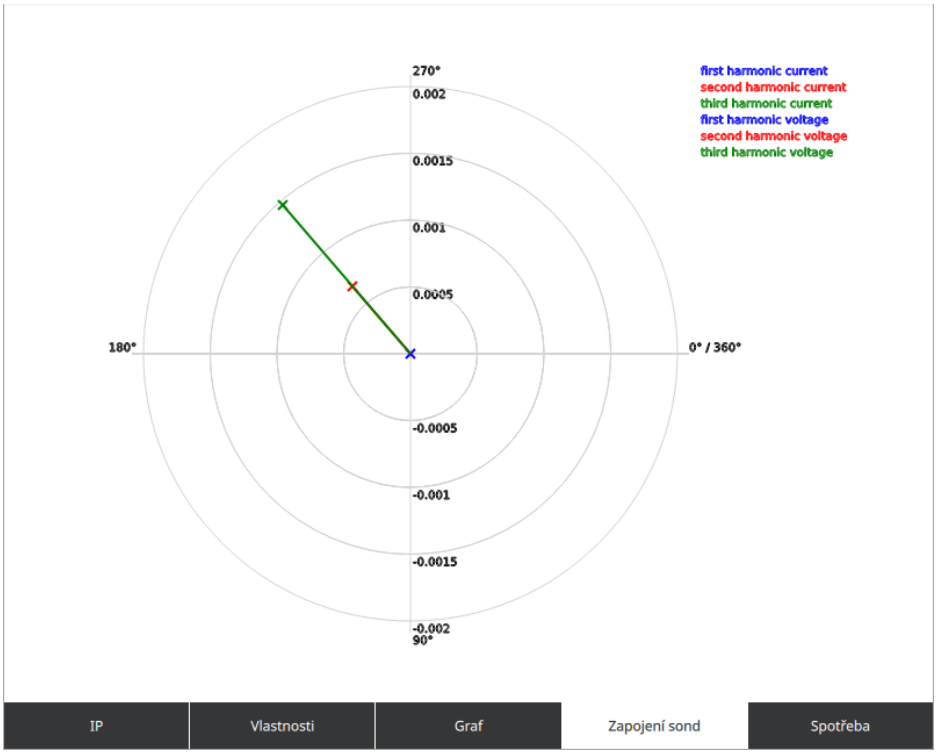
Zapojení sond se stalo nejobtížnější částí na programování jelikož se dalo využít modulu ChartView, který je tvořený zbytek grafů v aplikaci. Fázorový diagram se obnovuje na základě nově přijatých dat. K správné funkci stačí pouze poslední hodnota ze všech měření. Graf se přizpůsobuje velikosti okna. Osy grafu jsou auto-

Obrázek 5.2: Možnost skrytí křivky kliknutí na položku v legendě.

maticky dopočítávány na základě nejvyšší hodnoty fázoru s mírným zaokrouhlením. Celý diagram viz obrázek 5.3 má na starosti následující skupina JavaSriptových funkcí pro správné vykreslení plátna. Nejdříve vysvětlím pomocné prvky.

- *drawLegend()* vykreslí do pravé horní části legendu jmen jednotlivých fázorů. Fázory jsou barevně rozlišené.
- *drawLine()* vykreslí do plátna základní kříž, který se stane osami diagramu.
- *drawCircles()* vytvoří čtyři kružnice, které jsou od sebe stejně vzdáleny. Kružnice reprezentují v místě protnutí s osou vzdálenost, od středového bodu.
- *drawDegrees()* vypíše do grafu označení os, jaký úhel reprezentují.
- *drawPhasor()* vykresluje spojnici mezi nulovým bodem a bodem, který je reprezentovaný úhlem a vzdáleností od středu. Tyto hodnoty se získávají z aktualizace aktuálně měřených veličin. Pomocí úhlu a vzdálenosti však nelze ihned zjistit, kde se bude daný bod nacházet. Přepočty úhlu a vzdálenosti od středu do souřadnicového bodu [x,y] má na starosti funkce *preprocess()*.
- *preprocess()* řeší přepočet úhlu do bodu. Podle toho v jakém kvadrantu by se bod měl z úhlu vykreslit se zavolá s danými parametry funkce *calculate()* pro výpočet.
- *calculate()* pomocí pravidel pro výpočet stran pravoúhlého trojúhelníka dopočítá z hodnot vzdálenosti a úhlu umístění, kde se bod bude nacházet.
- *resize()* přepočítá, dle měřítka osy, bod [x,y] na jeho správné místo.
- *getSize()* řeší veškeré problémy ohledně zvětšování okna aplikace, tak aby byla vždy správně zobrazená. Přepočítává to velikosti kružnic v *drawCircles()*, velikosti os z *drawLine()*, umístění legendy, okraje diagramu, umístění popisků os apod.
- *drawAxis()* vykreslí podle typu fázoru na konec jeho přímky rozpoznávací značky, křížek a kolečko.

Graf spotřeby je vytvořen koláčovým grafem. Jednotlivé části jsou dopočítány z fází. Jedná se o poměr mezi jednotlivými hodnotami.



Obrázek 5.3: Fázorový diagram vytvořený pomocí JavaScriptu.

6 Závěr

Cílem bakalářské práce bylo vytvoření vlastního návrhu aplikace pro sběr aktuálních dat z měřicích přístrojů. S ohledem na stáří zadání práce byl software rozšířen o nové chtěné požadavky jako možnost získání dat uložených v JSON formátu, které byly staženy z webového serveru pomocí rozhraní REST API. Obě možnosti umožňují reprezentaci dat do grafu. Při práci s JSON daty je aplikace omezená funkčností pouze na některé prvky. Zpracováním požadavku o přístup k REST API jsem umožnil získání zpětné reakce na zpracování archivu, jak k němu přistoupit a jak se dostat k vybraným datovým polím. Pro návrh celé aplikace bylo využito frameworku Qt s jeho bohatou dokumentací s různými příklady, jak docílit chtěných výsledků. Framework Qt mi zprostředkoval množství grafických prvků, bez kterých by samotná tvorba byla velice náročná.

Aplikace je zamýšlena spíše pro techniky měřicích systémů než-li jejich majitele. Z tohoto důvodu nebylo nutné mít graficky pestrou aplikaci, avšak šlo o jednoduchost použití a splnění veškerých požadavků.

Při práci s grafy byly splněné veškeré požadavky, takže je možné jednotlivé křivky v grafech skrývat a oříznout pomocí funkce přiblížení. Kontrolu zapojení sond umožňuje vytvořený fázorový diagram.

Software je možné rozšířit o nové funkce. Aplikaci by bylo vhodné lépe zpracovat po grafické stránce mimo jiné by mohla být rozšířena o nové vstupní struktury, například načtení CEA archívu, který aplikace umožňuje stáhnout z měřicího přístroje. Zajímavým rozšířením by byla také možnost připojení a správa několika přístrojů zároveň s možností zobrazení hodnot do jednoho grafu, aby bylo možné hodnoty mezi sebou porovnávat mezi jednotlivými přístroji a sledovat jejich odchylky, na základě kterých by se daly zjistit chyby v instalacích.

Vzhledem k stále přibývajícím požadavkům na možnost práce s novými měřicími přístroji a měřenými veličinami, je zřejmé, že software je možné dále vyvíjet o nové prvky.

Literatura

[1] LAZAR, Guillaume a Robin PENEA, 2017. Mastering Qt 5. Birmingham Mumbai: Packt Publishing - ebooks Account. ISBN 978-1-78646-712-6.

[2] QML Applications. Qt Documentation [online]. Finland: The Qt Company, 2017 [cit. 2018-08-03]. Dostupné z: http://doc.qt.io/qt-5/qmlapplications.html

[3] JINHUI, Q., L. D. HUI a Y. JUNCHAO, 2012. The Application of Qt/Embedded on Embedded Linux. In: 2012 International Conference on Industrial Control and Electronics Engeneering [online]. s. 1304-1307.

[4] Learning Linux for embedded systems. Embedded [online]. 2015 [cit. 2017-10-10]. Dostupné z: https://www.embedded.com/electronics-blogs/open-mike/4420567/Learning-Linux-for-embedded-systems

[5] RYANNEL, J. a J. THELIN. Qt5 Cadaques [online]. 2018 [cit. 2018-08-03]. Dostupné z: https://qmlbook.github.io/index.html

[6] Qt Creator. Qt [online]. Finland: The Qt Company, c2016 [cit. 2019-04-28]. Dostupné z: https://wiki.qt.io/Qt_Creator

[7] Signals and Slots. Qt Documentation [online]. Finland: The Qt Company, c2016 [cit. 2019-04-28]. Dostupné z: https://doc.qt.io/archives/qt-4.8/signalsandslots.html

[8] Qt Quick 5.11. Qt Documentation [online]. Finland: The Qt Company, c2018 [cit. 2018-08-04]. Dostupné z: https://doc.qt.io/qt-5.11/qtquick-index.html

[9] GTK+. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-08-04]. Dostupné z: https://en.wikipedia.org/wiki/GTK%2B

[10] Parker Hannifin Corporation. SensoControl®: Diagnostic Test Equipment for Hydraulics [online]. 2010, s. 32-33 [cit. 2019-04-02]. Dostupné z: http://www.parker.cz/wp-content/uploads/2013/12/CAT-4054-2-UK.pdf

[11] Use case. Techopedia [online]. Techopedia, c2019 [cit. 2019-04-19]. Dostupné z:
https://www.techopedia.com/definition/25813/use-case

[12] Model/View Programming. Qt Documentation [online]. Finland: The Qt Company, c2019 [cit. 2019-04-20]. Dostupné z:
https://doc.qt.io/archives/qt-4.8/model-view-programming.html

[13] Systems engineering fundamentals [online]. Fort Belvoir, Va.: Defense Acquisition Universi-
ty Press, [2001], [1999] [cit. 2019-04-20]. ISBN 978-1484120835. Dostupné z:
https://web.archive.org/web/20170131231503/http://www.dau.mil/publications/publicationsdocs/sefguide%2001-01.pdf

[14] Using the Meta-Object Compiler. Qt Documentation [online]. Finland: The Qt Company, c2016 [cit. 2019-04-21]. Dostupné z:
https://doc.qt.io/archives/qt-4.8/moc.html

[15] CHRISTENSSON, Per. Framework. In: TechTerms [online]. c2019 [cit. 2019-04-22]. Dostupné z:
https://techterms.com/definition/framework