ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Aplikace pro práci s daty naměřenými systémem Digiterm
Student:	Martin Votruba
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Pokyny pro vypracování:
1. Seznamte se s aplikací pro ovládání systému Digiterm (www.digiterm.cz) používaného pro monitoring teplot.
2. Na základě seznámení se systémem, pokynů od zadavatele a vyslechnutí požadavků uživatelů vypracujte seznam požadavků na změny v aplikaci.
3. Požadavky na změnu prioritizujte. Po dohodě s vedoucím práce stanovte počet požadavků na vrcholu seznamu, které budete dále řešit.
4. Pro zvolené požadavky vypracujte analýzu a návrh řešení.
5. Implementujte návrhy, opatřete je dokumentací a vytvořte pro ně testy.
6. Popište další možný rozvoj systému.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 2. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Bakalářská práce

Návrh vizualizačního prostředí systému Digiterm

Martin Votruba

Vedoucí práce: Ing. Petra Pavlíčková Ph.D.

11. května 2017

Poděkování

Chtěl bych poděkovat paní Ing. Petře Pavlíčkové Ph.D. za vstřícný přístup, odborné rady a pomoc při zpracovávání této práce. Mé díky patří také Štěpánu Vaněčkovi, který ve mně vložil svoji důvěru. Chtěl bych také poděkovat rodičům, kteří mi byli po celou dobu studia oporou. Velký dík patří také mým spolužákům, s kterými jsme se během studia vzájemně podporovali a kteří mě nikdy neváhali podpořit ve chvílích slabosti.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 11. května 2017 .....

Odkaz na tuto práci

Votruba, Martin. *Návrh vizualizačního prostředí systému Digiterm.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Předmětem práce je návrh a implementace vizualizačního prostředí pro interpretaci naměřených dat systémem Digiterm. Systém Digiterm se používá na monitoring veličin (teplot, vlhkosti ...), nejčastěji ve farmaceutickém průmyslu. Program je vyvíjen v .NET frameworku jako desktopová aplikace pro operační systém Windows. Cílem práce je zpracovat analýzu a návrh nové verze programu. Starší verze programu nevyhovovala nárokům na moderním software a neimplementovala veškeré žádané funkce. Základem práce jsou rozpracované jednotlivé případy užití a jejich specifikace. V práci je zahrnut doménový a databázový model. Úspěšně provedenou analýzou se mi podařilo formulovat podobu nového programu tak, aby lépe vyhovoval požadavkům uživatelů.

Klíčová slova Vizualizace dat, doménový model, databázový model, Digiterm, případy užití, návrh programu

Abstract

The thesis aims to design and implement visualisation environment for interpretation of values measured by the Digiterm system. Digiterm is used for measurement of physical quantities as is temperature, pressure or humidity. The system is mostly deployed to pharmaceutical industry. The visualisation application was developed as desktop application for OS Windows in .NET framework. The goal of the thesis is to elaborate an analysis and design a new version. The older version did not suit modern requirements and also does not implement all required functionality. The core of the thesis is in detailly specified use cases. The thesis contains a domain and database model. With successful analysis I was able to implement a new version of the application that better suits user's requirements.

Keywords Data visualisation, domen model, use case, database model, analysis

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V současné době je napříč odvětvími kladen čím dál tím větší důraz na sledování kvality. Jak uvádí časopis Automa *"náklady na měření a vážení dnes v Evropě představují celých 6 % celkového hrubého domácího produktu"*.[2] Jedno z odvětví kde je na sledování kvality kladen obzvláště velký důraz, zvláště z hlediska přísné legislativy, je farmaceutický průmysl spolu se zdravotnictvím. Při skladování zdravotnického materiálu a výrobách citlivých na okolní podmínky je zapotřebí mít systém na monitoring teplot, vlhkosti popřípadě tlaku. Takový systém musí být schopný zaznamenávat naměřená data a hlavně v případě vzniku poruchy či jiné mimořádné události, upozornit obsluhující personál a zabránit tak hrozící škodě.

Téma bakalářské práce jsem si zvolil, jelikož jsem dostal příležitost podílet se na obnovení systému Digiterm. Digiterm je monitorovací systém vyvinutý společností *Regucon s.r.o.*, který slouží ke sledování fyzikálních veličin. Běžně se jedná o hlídání teplot, za kterých jsou skladovány termolabilní materiály, např. pro uchování plazmy na transfúzních odděleních, skladování léčiv v lékárnách či ukládání diagnostik z laboratoří. Digiterm umí archivovat měřená data a vyvolat alarm v případě poruchy, čímž řeší většinu z výše popsaných potřeb. Softwarové vybavení softwaru je nicméně již 20 let staré a ač se prokázalo mimořádně nadčasové, tak dnes již požadavkům nevyhovuje. Pro uživatele není intuitivní a nepodporuje všechny funkce, které jsou potřeba. Vzhledem ke skutečnosti, že se tyto problémy dlouho neřešily, někteří zákazníci začali vyhledávat konkurenční řešení.

Svou prací bych chtěl tyto problémy vyřešit a systém pro zákazníky udělat opět atraktivnější. Tématem mé práce je analýza, návrh a implementace nové verze vizualizační aplikace systému - Read.

Práce je rozdělena na dvě části, teoretickou a praktickou. V teoretické části je přiblížena problematika managementu kvality. V praktické části je vytvořena analýza sytému a popsán způsob implementace.

Kapitola 1

Cíle práce a metodika

1.1 Cíle práce

Cílem práce je navrhnout nové vizualizační prostředí pro systém Digiterm. Tento program by měl vycházet z reálných požadavků zákazníků, které by měly být v práci zachyceny. Výsledný produkt by měl reflektovat požadavky uživatelů lépe, než ten stávající a jeho ovládání by pro uživatele mělo být intuitivnější.

Jelikož v oblasti může dojít k legislativním změnám, které vyústí v změny bussines procesů zákazníků, aplikace by měla být snadno rozšiřitelná. Druhým cílem práce proto je, aby sloužila i jako podklad pro zorientování se v problematice pro budoucí programátory, kteří na moji práci budou navazovat. V práci by tak měl být zahrnut jak bussines pohled na problematiku, tak stránka implementační (doménový model, databázový model).

Posledním cílem práce je program řádně otestovat a připravit pro nasazení. V závěru by měla práce revidovat, které požadavky uživatelů se zdařilo vyřešit a nastínit řešení těch nevyřešených.

1.2 Metodika

V první fázi práce vymezím na základě zpětné vazby od zákazníků a konzultace se zadavatelem problémy, které by měl program zákazníkům řešit. Tyto požadavky budou podpořeny studiem norem, v kterých jsou mnohé z těchto problémů ukotveny. Na základě bussines procesů definuji případy užití, které by měl program zahrnout. Z nich bude již vycházet implementace samotné aplikace a budou sloužit i ke kontrole její validity.

Kapitola 2

Rešeršní část

2.1 Management kvality

Systém managementu kvality zajišťuje organizaci, aby byla schopná trvale poskytovat produkt nebo službu, které splňují požadavky zákazníka a příslušné požadavky předpisů. Rovněž si klade za cíl zvyšovat spokojenost zákazníka, a to efektivní aplikací systému, včetně procesů pro jeho neustálé zlepšování. [3]

2.1.1 Normy ISO

Velkou roli v celosvětovém managementu kvality hraje Mezinárodní organizace pro normalizaci, známá také pod zkratkou ISO. Tato organizace sdružuje 161 států z celého světa [4]. Její snahou je normalizovat postupy a to především vydáváním ISO norem. Tyto normy jsou vytvářeny za spolupráce všech členských zemí. Každá země je členem společenství prostřednictvím národní normalizační instituce, která ji v debatách zastupuje. Zástupce za Českou republiku je Úřad pro technickou normalizaci, metrologii a státní zkušebnictví [5].

2.1.2 Normy ČSN

Vydávání Českých technických norem má na starosti Úřad pro technickou normalizaci, metrologii a státní zkušebnictví. Přibližně 90% tvoří přejaté normy evropské či mezinárodní. Označení těchto norem tvoří značka ČSN následována značkou příslušné přejímané normy (např. ČSN EN, ČSN ISO, ČSN EN ISO...). Zbývajících 10% tvoří původní české normy, které vznikají pouze v oblastech, kde evropské či mezinárodní normy chybí.[6]

Technické normy nejsou samy o sobě právně závazné. Jejich právní závaznost však může vyplývat z jiného právního aktu: právního předpisu, smlouvy, rozhodnutí správního orgánu nebo pokynu nadřízeného pracovníka. Každý případ odchýlení od ČSN musí být nicméně řádně odůvodněn a zvolená al-

ternativa by měla být prokazatelně vhodnější, něž jakou předepisuje norma. [7]

2.1.3 Akreditace

Akreditací se rozumí oficiální uznání (reprezentované vydáním Osvědčení o akreditaci), že subjekt akreditace je způsobilý provádět specifické činnosti.[8] Akreditaci jako oprávnění uděluje příslušná akreditační autorita, která ji odvozuje obvykle ze zákona. Při procesu přidělování akreditace provádí akreditační autorita nestranné, objektivní a nezávislé posouzení , zdali zkoumaný subjekt splňuje všechny požadavky pro její udělení. Podmínky pro udělení akreditace jsou zpravidla vymezeny technickou normou a metodickými pokyny pro akreditaci.

2.2 Systém pro sledování veličin

Z mnohých norem a nařízení vzniká provozovatelům nutnost nasadit do svého prostředí systém, který by jim usnadnil plnění legislativy. Příklady takových případů jsou:

- Společnosti deklarující kvalitu dle normy ČSN EN ISO 9001, musí mít "zavedené činnosti monitorování a měření v odpovídajících etapách k ověření, zda byla splněna kritéria pro řízení procesů nebo výstupů a také přejímací kritéria pro produkty"[9].
- Firmy nakládající s léčivy, které musí klást důraz na dodržování skladovacích podmínek. Často se jedná o choulostivý materiál. V případě jejich nedodržená může dojít ke škodě materiální a popřípadě i ke škodě na zdraví, v případě aplikace znehodnocené látky. Skladování léčiv se proto musí řídit pokynem *DIS-15 verze 3 Sledování a kontrola teploty při skladování a přepravě léčiv* [10] vydávaný Státním ústavem pro kontrolu léčiv.

Na základě těchto případů můžeme definovat obecné požadavky na takový systém.

2.2.1 Sledování a kontrola veličin

Systém poskytuje údaje o aktuálních hodnotách měřených veličin. Tyto hodnoty jsou průběžně automaticky vyhodnocovány oproti předem definovanému rozmezí, které se nastavuje na spodní i horní limit. Zařízení na měření je vybaveno signalizačním mechanismem pro případ, že stanovené teplotní rozmezí je překročeno (alarm). Signalizační mechanismus může být různý - houkačka, SMS, email - ale mělo by být takové, aby přimělo obsluhující personál nastalou situaci neprodleně řešit dle předem schváleného procesu.

Měřící čidla jsou rozmístěna tak, aby podávaly co nejlepší obraz o celém prostoru. Je vhodné před rozložením čidel měřený prostor zmapovat - v případě teploty například pomocí teplotní mapy - a čidla rozmístit až na základě takové analýzy.

2.2.2 Vedení a uchovávání záznamu

Systém periodicky zaznamenává průběh veličin v čase. Perioda je nastavena tak, aby byla dostatečně krátká, nebo aby záznam probíhal v nejméně příznivou dobu. Data ze systému jsou zálohována.

Krom historických dat samotných si systém udržuje i informace o proběhnutých alarmech. K těmto záznamům může obsluhující personál doplnit zdůvodnění a posouzení míry rizika narušení standardních procesů.

2.3 Existující řešení

V českém prostředí najdeme tři řešení, která by splňovala požadavky na systém pro sledování veličin. Kromě systému *Digiterm®*, jehož rozvojem se ve své práci zabývám jsou to *Comet* a *MS Falcon®*. V této kapitole nebudu hodnotit systém Digiterm, jelikož jeho stav hodnotím v praktické části práce.

2.3.1 Comet

Systém vyrábí firma *Comet system s.r.o.* a je distribuovaný sítí dodavatelů. Centrem systému je *Comet Database*, do které jsou ukládány data ze snímačů. Obsluha systém řídí buď skrz vizualizační program *Comet Database Viewer* nebo přímo ze snímačů.

2.3.1.1 Přednosti systému

- Signalizace alarmů může být akustická, optická nebo je obsluha informována odesláním varovného emailu/SMS. Některé řady snímačů mají možnost připojit externí signalizaci pomocí dvoustavového výstupu.
- K datům systému lze přistoupit i pomocí webového rozhraní.
- Lze nastavit automatický export dat, který se dá dobře využít pro archivaci dat.
- Snímače se vyznačují vysokou mírou autonomie, obsluha může přímo na nich zobrazit aktuální data ze systému, popřípadě změnit alarmy.
- Snímače systému jsou dodávány s kalibračním listem, který vychází z požadavků normy ČSN EN ISO/IEC 17025.

Obrázek 2.1: Vizualizace fungování systému Comet[1]

- Snímače systému jsou vybaveny náhradním napájení v podobě AAA baterií pro případ výpadku proudu.
- Obsluha si může k libovolnému naměřenému záznamu zapsat poznámku.
- Obsluha se do systému přihlašuje prostřednictvím uživatelských účtů, systém proto umožňuje vést historii změn alarmů a přihlašování uživatelů.

2.3.1.2 Slabiny systému

- Systém disponuje širokými možnostmi nastavení, což jistě ocení zkušení uživatelé, nicméně pro méně zdatné uživatele může být orientace v programu složitá.
- Při zobrazování dat vybírá uživatel sondy podle snímačů, ke kterým jsou připojeny. Uživatel tedy musí mít přehled o tom, které sondy jsou připojené ke kterým snímačům, což na něj klade zbytečné nároky.
- Jelikož systém nedodává přímo výrobce, není možné ho upravit dle přání zákazníka.

2.3.2 MS Falcon®

Systém vyráběný a dodávaný firmou *KESA, s.r.o.*. MS Falcon je koncipovaný stavebnicovým systémem, což umožňuje jeho vysokou variabilitu pro nasazení.

2.3.2.1 Přednosti systému

- Do systému může obsluha přistoupit pouze po přihlášení a je vedena důkladná historie veškerých úkonů.
- Alarmy vyhodnocuje nezávislý modul, který tak na případnou událost upozorní i v případě výpadku síťových prvků.
- Každý senzor má nastavitelné dvě meze pro spodní o horní alarm. Obsluha tak může být upozorněna předem na blížící se kritickou hodnotu.
- V případě zájmu nabízí výrobce i vzdálený dohled nad systémem.

2.4 Shrnutí kapitoly

Systém norem ISO zajišťuji mezinárodní standardizaci managementu kvality. Firmy, které jsou dle těchto norem akreditovány, pak jsou důvěryhodnější vůči svým zákazníkům a mohou mít větší důvěru k procesům nastaveným v jejich prostředí.

Firmy nakládající s materiálem vyžadujícím zvláštních skladovacích podmínek, zavádějí procesy, které se snaží zamezit porušení těchto podmínek. Tyto procesy můžou vycházet z požadavků norem, vyhlášek, pokynů dohlížejících institucí či prostě jen potřeby firmy zamezit škodám. Užitečnou roli může sehrát systém pro sledování veličin, který na podmínky dohlíží.

Kapitola 3

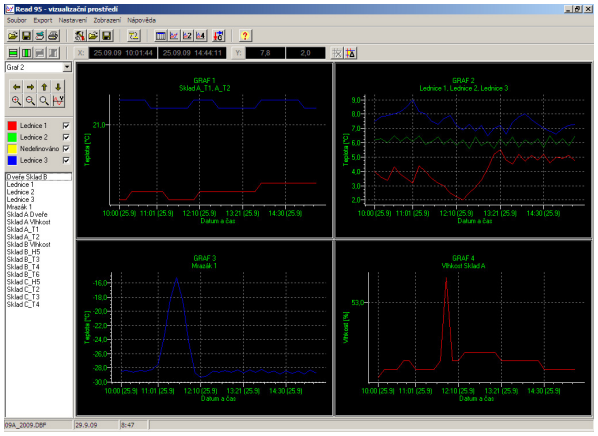
Analýza

3.1 Problémy existujícího řešení

Jak již bylo předestřeno, současná vizualizační aplikace systému trpí mnoha nedostatky, které se zde pokusím identifikovat.

3.1.1 Funkční nedostatky

- Program podporuje dva režimy práce s daty - graf a tabulku. Bohužel není možné mít zároveň zobrazenou tabulku i graf, což je pro uživatele nepohodlné.
- Read umí prohlížet v jedné chvíli data jen z jedné databáze. To způsobuje problém, pokud chtějí uživatelé pracovat s daty z delšího časového období, jelikož si typicky vytváří novou databázi pro každý měsíc.
- Aplikace v současném stavu chybí řada žádaných funkcí pro práci s grafem. Uživatelé si nemohou zobrazit hranice zobrazených dat, chybí podpora exportu do více formátů (zvláště formát PDF je žádaný) a celková orientace v grafu je možná pouze pomocí tlačítek, přičemž intuitivnější by bylo použít myš.
- Nastavování mezí alarmů je nyní řešeno skrz program Digiterm IO. Takový model je nevhodný a bylo by žádoucí, kdyby uživatelé mohli všechny běžné akce provést v programu Read. Program Digiterm IO by měl sloužit pouze pro vyčítání dat z automatů.
- Systém neumožňuje zobrazit historii změn alarmů. Pro zákazníky je důležité mít možnost zjistit, který uživatel alarm přenastavil.
- V programu není viditelný výpis nastalých alarmů, s možností ho okomentovat. Takovýto záznam je po zákaznících vyžadován sledujícími autoritami a v tuto chvíli si ho musí vést ručně mimo systém Digiterm.



Obrázek 3.1: Uživatelské prostředí programu Read95

- Při pomalejším připojení, například pokud chtějí uživatelé s programem pracovat z domova, trvá spuštění programu příliš dlouho. V případě alarmu tak nemají okamžitou kontrolu stavu měřených míst.

3.1.2 Nefunkční nedostatky

- Instalace systému byla problematická. Oba programy (Read95, NetCom) potřebovali ke svému fungování nastavit údaje v registrech, což je problém v prostředí lékáren, kde bývají přísné bezpečností podmínky a místní počítačoví správci do registrů neradi umožňují přístup cizím osobám.

- Systém byl postaven na *dBase* databázi. S ní často vystávaly problémy ve chvílích, kdy na stejném počítači běželo více aplikací využívající *dBase* databázi v jiných verzí. V současné době je navíc *dBase* již zastaralou technologií a chybí pro ni tedy nástroje a komunita uživatelů.

3.2 Požadavky na aplikaci

3.2.1 Funkční požadavky

- Pro přístup do programu bude vyžadováno přihlášení. Každý uživatel bude mít vlastní účet a bude tak možné sledovat historii změn, které pracovník provedl. Systém bude rozlišovat dvě role uživatelů - administrátoři budou moci oproti prosté obsluze spravovat uživatelské účty. Každý uživatelský účet bude mít oprávnění přistupovat jen k definované množině senzorů.

- Program bude umožňovat prohlížení naměřených dat v režimu tabulky a grafu. Data v tabulce bude možné filtrovat definováním filtru nad

sloupcem. Orientace v grafu bude možná jak tažením myši, tak pomocí šipek na klávesnici. V programu bude možné procházet historii pohybu grafem. Uživatel bude moci také přesně vymezit rozmezí zobrazených dat, upravit barevné zobrazení grafu a zapnout pravítka pro snadnější orientaci.

- Data z tabulky bude možné exportovat do formátu CSV nebo PDF, z grafu bude možný export pouze do PDF. Rozmezí exportovaných dat bude odpovídat aktuálně zobrazeným datům (nastavením filtrů u tabulky a orientací v grafu).

- Pro vizuální kontrolu naměřených dat si uživatel může pro každý senzor nastavit spodní a horní limit. Po jeho nastavení se data v tabulce a grafu barevně zvýrazní. V grafu si uživatel navíc může volitelně aktivovat zobrazení horizontálních linií představující limity senzorů.

- Data se do programu načtou po otevření souboru archivní databáze nebo po spuštění programu. Data z časových úseků zachycených ve více databázích budou moci uživatelé načíst pomocí funkce připojení dat.

- Program bude zobrazovat aktuální měřené hodnoty. Barevně odlišené budou hodnoty senzorů, které jsou alarmované. Rozložení zobrazení aktuálních hodnot bude možné upravit pomocí

  – Změny pořadí senzorů v zobrazení.
  – Seskupením více senzorů do jedné skupiny, kterou bude následně možné pojmenovat.

  Tyto změny se automaticky uloží a budou specifické pro každého uživatele.

- Uživatelé budou moci zobrazit hodnoty nastavených alarmů a historii jejich změn. Uživatel s administrátorským oprávněním bude moci navíc alarmy přenastavit. Pro každý senzor se nastavuje, zda je alarm zapnutý, horní, spodní mez, telefonní číslo a emailová adresa. V konfiguračním souboru bude možné nastavit pro celý systém, jestli se mají telefonní čísla a emailové adresy zobrazovat. Stejným způsobem bude možné skrýt celou sekci alarmů včetně historie.

3.2.2 Nefunkční požadavky

- Program bude fungovat na počítačích s OS *Windows Vista* a novějších verzí (respektive *Windows NT* 6.0 a novější).

- Data bude program čerpat z *MS Access* databáze, nebo pomocí importu dat ze starší verze systému, databáze formátu *dBase*.

- Aplikace bude napsána tak, aby byla jednoduše rozšiřitelná. Především by mělo být možné bez větších obtíží kód upravit tak, aby mohla data číst z *MS SQL* databáze.

- Po přihlášení uživatele nebude programu trvat načtení rozložení, včetně nahrání dat z databáze, déle jak 5 vteřin. Testované rozložení bude mít jednu tabulku a tři grafy, přičemž v každém okně budou data ze 4 senzorů. Jako referenční zařízení poslouží počítač s *Windows 7*, 4 GB RAM, procesorem *Intel® Core™2 Duo E7400* a pevným diskem *Western Digital WD5000AAKS*.

- Aplikace nesmí vyžadovat přístup do registrů a veškerá její konfigurace by se měla provést pomocí konfiguračních souborů. Celá instalace potom musí být nenáročná, ideálně proveditelná bez administrátorských oprávnění.

3.3 Případy užití

V následujících případech užití by měly být detailně zachyceny funkční požadavky na aplikaci. Případy užití jsou psány dle knihy Alistaira Cockubrna "*Writing effective use cases*".[11]

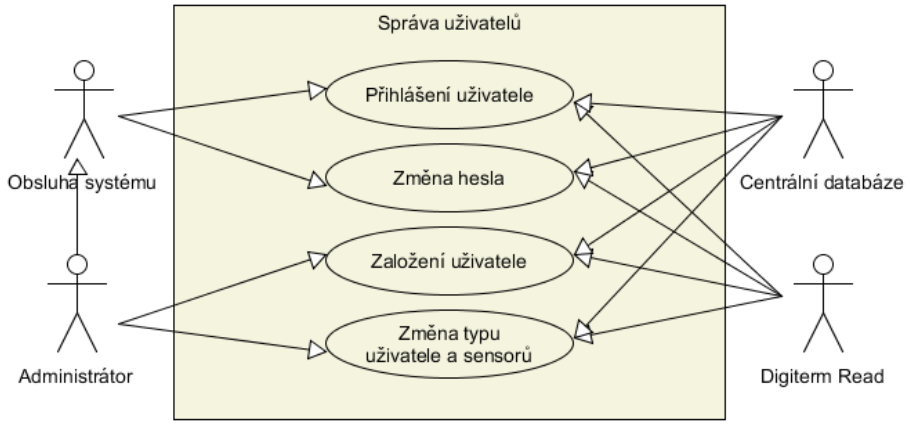
3.3.1 Aktéři

Uživatele systému můžeme rozdělit do dvou skupin.

- **Obsluha systému** - běžný uživatel

- **Administrátor systému** - pracovník ve vedoucí pozici, nebo jmenovaný spravováním systému. Krom práv běžného uživatele může zakládat nové uživatele a měnit alarmy.

3.3.2 Správa uživatelů



Obrázek 3.2: Diagram případů užití týkající se správy uživatelů

3.3.2.1 Přihlášení uživatele

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce se přihlásit do programu
- Firma - nechce dovolit neautorizovaný přístup do systému

Scénář úspěšného plnění:

1. Uživatel spustí program Digiterm Read.
2. Uživatel se přihlašuje do systému poté, co vyplní přihlašovací jméno a heslo. Pokud se uživatel rozhodne, Digiterm Read od něj při příštím přihlašování nebude přihlašovací údaje znovu vyžadovat.
3. Systém vyhodnocuje jestli zadané přihlašovací jméno v systému existuje a je svázané se zadaným heslem.
4. Zadané přihlašovací údaje jsou správně a uživatel je vpuštěn do systému.

Výjimky ze standardního průběhu:

4. Systém zjistí, že zadané přihlašovací jméno neexistuje, nebo není svázané se zadaným heslem. Tuto informaci systém prezentuje uživateli, který může přihlašovací údaje opravit.

3.3.2.2 Změna hesla uživatele

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Předpoklady:

- Uživatel má spuštěný program Digiterm Read
- Uživatel je přihlášen do systému

Zúčastněné strany:

- Administrátor má spuštěný program Digiterm Read a je přihlášen

Scénář úspěšného plnění:

1. Systém vyzve uživatele k zadání stávajícího a nového hesla. Nové heslo je třeba zadat dvakrát, aby se předešlo překlepu.
2. Uživatel vyplní stávající heslo a své nové heslo. Poté změnu hesla potvrdí.
3. Systém zkontroluje, jestli uživatel zadal správné heslo a obě nově zadaná hesla jsou shodná.
4. Systém změní uživateli heslo.

Výjimky ze standardního průběhu:

3a. Neplatné heslo

- 3a1. Systém zamítne změnu hesla.
- 3a2. Uživatel je vyrozuměn o nastalé chybě a může akci opakovat.

3b. Neodpovídající si nová hesla

- 3b1. Systém zamítne změnu hesla.
- 3b2. Uživatel je vyrozuměn o nastalé chybě a může akci opakovat.

3.3.2.3 Založení uživatele

Hlavní účastník: Administrátor systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Předpoklady:

- Administrátor má spuštěný program Digiterm Read a je přihlášen

Zúčastněné strany:

- Administrátor systému - chce založit nového uživatele
- Firma - chce mít kontrolu nad uživatelskými účty v systému

Scénář úspěšného plnění:

1. Administrátor chce založit nový uživatelský účet.
2. Systém vyzve administrátora k zadání
   - uživatelského jména
   - hesla - je třeba zadat dvakrát
   - seznamu sensorů, ke kterým má uživatel přístup
   - typu účtu - administrátor účet/běžný uživatel
3. Administrátor vyplní požadované údaje.
4. Systém zkontroluje správnost údajů a založí uživatelský účet.

Výjimky ze standardního průběhu:

4a. Zvolené uživatelské jméno je již používáno

4a1. Systém zamítne vytvoření uživatelského účtu.

4a2. Administrátor je vyrozuměn o nastalé chybě a může akci opakovat.

4b. Zadaná hesla si neodpovídají

4b1. Systém zamítne vytvoření uživatelského účtu.

4b2. Administrátor je vyrozuměn o nastalé chybě a může akci opakovat.

3.3.2.4 Změna sensorů pro uživatelský účet a jeho typu

Hlavní účastník: Administrátor systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Předpoklady:

- Administrátor má spuštěný program Digiterm Read a je přihlášen

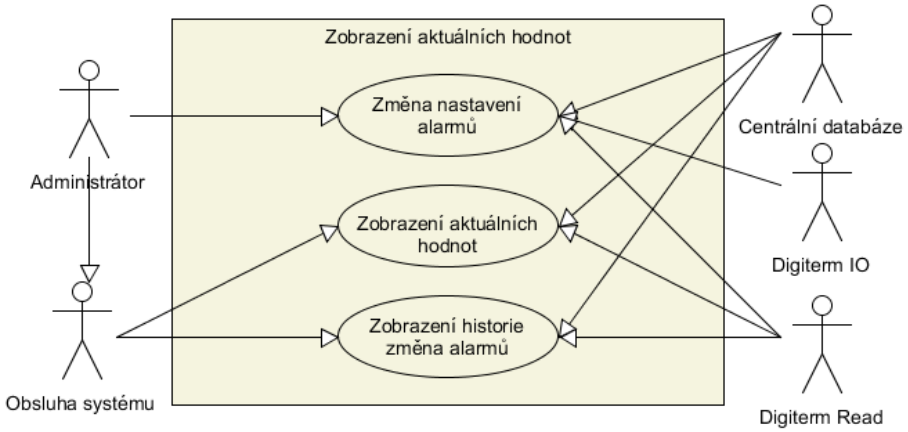
Zúčastněné strany:

- Administrátor systému - chce změnit typ uživatelského účtu nebo seznam přiřazených senzorů
- Firma - chce mít kontrolu nad uživatelskými účty v systému

Scénář úspěšného plnění:

1. Administrátor vybere uživatelský účet, který chce upravit
2. Systém zobrazí o jaký typ účtu se jedná a seznam všech senzorů, v kterém je zřejmé, ke kterým má uživatel přístup.
3. Administrátor upraví přístupová práva k senzorům popřípadě typ účtu
4. Systém zpracuje požadované změny.

3.3.3 Alarmy a aktuální hodnoty



Obrázek 3.3: Diagram případů užití týkající se alarmů a aktuálních hodnot

3.3.3.1 Změna nastavení alarmů

Hlavní účastník: Administrátor systému

Rozsah: Digiterm - celý systém

Úroveň: Záměr uživatele

Předpoklady:

- Administrátor má spuštěný program Digiterm Read a je přihlášen

Zúčastněné strany:

- Administrátor systému - chce změnit nastavení alarmů
- Firma - chce vést historii změn alarmů

Scénář úspěšného plnění:

1. Read se připojí na lokální RESTový endpoint vystavený IO. Read dotazem GET získá seznam alarmů a zobrazí ho administrátorovi.
2. Read kontroluje periodicky GET dotazy na RESTové rozhraní, zda nejsou měněny jiným uživatelem.
3. Administrátor provede změnu alarmů - alarm zapne/vypne, změní hodnotu spodní/horní hranice a poté změnu potvrdí.
4. Read odešle pomocí dotazu PUT na RESTové rozhraní seznam změn alarmů a identifikátor uživatele.
5. IO zahájí zápis změn alarmů do automatu. Po jejich dokončení přidá záznam do historie změn alarmů.
6. Read kontroluje stav zápisu alarmů periodicky GET dotazy na RESTové rozhraní. Ve chvíli, kdy je operace úspěšně dokončena aktualizuje zobrazené hodnoty tak, aby byl stav identický se stavem alarmů v automatu.

Výjimky ze standardního průběhu:

1a. Připojení na RESTový endpoint selže.

1a1. Administrátor je informován chybovou hláškou a může akci opakovat.

2/4a. Alarmy jsou ve stavu změny.

2/4a1. Administrátor je vyrozuměn chybovou hláškou.

2/4a2. Read zamezí administrátorovi v dalších změnách alarmů.

2/4a3. Když je operace změn alarmů dokončena, nastavení alarmů je aktualizováno, aby odpovídalo datům v automatu. To může vést k nepohodlí uživatelů, jelikož tím ztratí provedené změny. Uživatelé nicméně většinou mění hodnotu jednoho alarmu, situace kdy by dva administrátoři měnili najednou alarmy, je navíc nezvyklá.

3a. Neplatné nastavení mezí

3a1. Nastavení alarmů je neplatné v následujících případech:

* Spodní mez je vyšší než horní, či naopak.
* Alarmy jsou před zápisem do automatu převedeny na nezápornou celou hodnotu přičtením konstanty a vynásobením koeficientem. Velikost pole uchovávající alarmy v automatu je 2 bajty, a nastavení tedy není platné pokud neplatí $f(a) = a * k + q \in [0, 65535]$.

3ab. Uživatel je upozorněn na chybu s informací, u kterého senzoru / senzorů k chybě došlo.

5b. Komunikace s automatem selže

5b1. Digiterm IO začne po GET dotazu na stav zápisu alarmů vracet informaci o chybě.

5b2. Read vyrozumí administrátor o chybě.

5b3. Hodnoty alarmů jsou změněny tak, aby odpovídaly hodnotám v automatu.

3.3.3.2 Zobrazení aktuálních hodnot

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce vidět aktuální hodnoty senzorů

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read a je přihlášena.

Scénář úspěšného plnění:

1. Uživatel otevře okno Digiterm - View.

2. Read se připojí k centrální databázi a načte si aktuální hodnoty, které následně zobrazí uživateli. Hodnoty které překračují některou mez alarmů jsou barevně zvýrazněny.
3. Read se periodicky v intervalu 5 vteřin připojuje k databázi a načítá nová data.

3.3.3.3 Zobrazení historie změn alarmů

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce vidět historii změn nastavení alarmů

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read a je přihlášena.

Scénář úspěšného plnění:

1. Uživatel otevře okno Digiterm - View.
2. Read se připojí k centrální databázi, načte a zobrazí historii alarmů. V historii je zachyceno kdo a kdy změnil nastavení hranic a zapnutí alarmů.

3.3.4 Práce s daty

3.3.4.1 Otevření archivní databáze

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

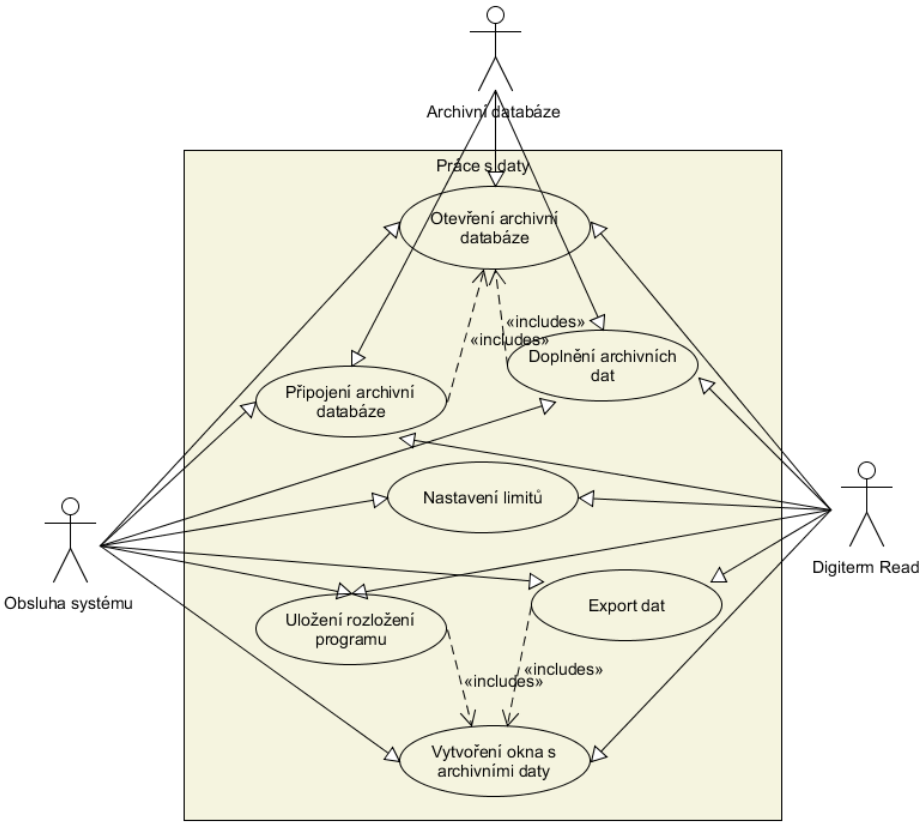
Zúčastněné strany:

- Obsluha systému - chce otevřít archivní databázi

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read a je přihlášena.

Scénář úspěšného plnění:



Obrázek 3.4: Diagram případů užití týkající se práce s daty

1. Uživatel vybere pomocí dialogového okna systému Windows soubor archivní databáze.

2. Read zkontroluje, že otevíraný databázový soubor je validní a automaticky načte data z nově otevřené databáze do všech tabulek a grafů.

Alternativy ke standardnímu průběhu:

1a. Historie databází

1a1. Uživatel může také vybrat některou ze tří naposledy otevřených databází.

1b. Import databáze předešlého formátu

1b1. Pro práci s daty pořízenými předešlou verzí systému je potřeba konvertovat data do nového formátu databáze. Uživatel vybere pomocí systémového dialogu soubor staré databáze ke konverzi.

1b2. Read data konvertuje a vytvoří databázi nového formátu se stejným jménem jako její vzor. Pokud soubor stejného jména již existuje, uživatel je dotázán, zda si ho přeje smazat.

Výjimky ze standardního průběhu:

2a. Vybraný soubor není validní - buď zadaná cesta neexistuje, nebo formát databáze neodpovídá

2a1. Uživatel je informován o chybě a je vyzván, aby zvolil jiný soubor.

3.3.4.2 Připojení archivní databáze

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce zobrazit data, která jsou uložena ve více archivních souborech, například data z databází zachycujících dva po sobě jdoucí měsíce.

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read, je přihlášena a proběhl případ *3.4.3.1 Otevření archivní databáze*

Scénář úspěšného plnění:

1. Uživatel vybere pomocí dialogového okna systému Windows soubor archivní databáze k připojení.

2. Read zkontroluje, že otevíraný databázový soubor je validní. Data z nové databáze doplní do všech otevřených tabulek a grafů. U grafů se linie dat automaticky napojí, pokud na sebe data navazují.

Výjimky ze standardního průběhu:

2a. Vybraný soubor není validní - buď zadaná cesta neexistuje, nebo formát databáze neodpovídá

2a1. Uživatel je informován o chybě a je vyzván aby zvolil jiný soubor.

3.3.4.3 Doplnění archivních dat z databáze

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - má otevřený program Digiterm Read a během jeho práce je do archivní databáze doplněný nový záznam. Uživatel chce tento záznam načíst.

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read, je přihlášena a proběhl případ *3.4.3.1 Otevření archivní databáze*.

Scénář úspěšného plnění:

1. Read se periodicky v intervalu 15 vteřin připojuje k databázi a kontroluje, jestli v ní nepřibyl nový záznam.

2. Read zjistí, že v databázi je nový záznam, periodická kontrola nových dat ustane a tuto informaci předá uživateli zeleným podbarvením ikony obnovy dat.

3. Uživatel klikne na ikonu obnovy dat.

4. Read se připojí k databázi, načte nové záznamy a doplní je do všech oken grafu/tabulek. Periodická kontrola nových dat je obnovena.

Alternativy ke standardnímu průběhu:

2a. Automatické doplnění dat

2a1. Uživatel mít nastavenou automatickou aktualizaci nových dat z databáze. V takovém případě se rovnou přejde k bodu 4.

Výjimky ze standardního průběhu:

1a. Pokud připojení k databázi selže, uživatel není upozorňován, jelikož by to bylo rušivé pro plynulou práci s programem.

3.3.4.4 Nastavení limitů

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce změnit hodnoty, které se v grafech a tabulkách zobrazují barevně zvýrazněné.

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read a je přihlášena.

Scénář úspěšného plnění:

1. Obsluha zvolí možnost *Nastavení limitů* z horního menu aplikace.
2. Program zobrazí okno s nastavením limitů pro uživatele. Pro každý senzor si uživatel může nastavit spodní a vrchní limit.
3. Uživatel změní hodnoty a nastavení uloží.
4. Read změny uloží do databáze a zároveň nové nastavení uplatní i pro všechny již otevřená okna grafů a tabulek.

Výjimky ze standardního průběhu:

3a. U všech limitů musí platit, že nižší mez je skutečně menší, než vyšší. Pokud tomu tak není, daná pole jsou červeně podbarvena a uživateli je znemožněno nastavení uložit.

3.3.4.5 Vytvoření okna s archivními daty

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce zobrazit archivní data systému buď jako tabulku či graf.

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read a je přihlášena.

Scénář úspěšného plnění:

1. Uživatel vytvoří nové okno grafu či tabulky.
2. Uživatel přidá do vytvořeného okna senzory, které má okno zobrazovat.
3. Read se připojí k otevřené archivní databázi a načte do okna data pro přidané senzory.

3.3.4.6 Export dat

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce exportovat data ve formátu tabulky či grafu.

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read, je přihlášena a proběhl případ *3.4.3.5 Vytvoření okna s archivními daty*

Scénář úspěšného plnění:

1. Dle typu okna uživatel vybere data k exportu:
   - **Graf** - exportují se data aktuálně viditelná uživateli. Rozsah exportu tedy uživatel může ovlivnit přiblížením na část zobrazených dat.
   - **Tabulka** - u každého sloupce může uživatel zadat filtr, exportují se pouze záznamy, které vyhovují všem nastaveným filtrům.
2. Pomocí dialogového okna systému uživatel vybere typ a umístění exportovaného souboru. V případě tabulky si může vybrat z exportu do souboru typu PDF nebo CSV. Výchozí umístění exportovaných souborů je složka *Digiterm* ve veřejných dokumentech.
3. Read exportuje do umístění, které si uživatel nastavil.

Výjimky ze standardního průběhu:

2a. Soubor se stejným jménem již existuje

2a1. Uživatel je dotázán, zda-li chce soubor přepsat.

3.3.4.7 Uložení rozložení programu

Hlavní účastník: Obsluha systému

Rozsah: Digiterm Read

Úroveň: Záměr uživatele

Zúčastněné strany:

- Obsluha systému - chce aby při otevírání programu nemusel pokaždé otevírat okna grafů/tabulek a přidávat do nich senzory.

Předpoklady:

- Obsluha systému má spuštěný program Digiterm Read, je přihlášena a má nastavené rozložení oken tak, jak si představuje výchozí rozložení, tedy proběhl případ *3.4.3.5 Vytvoření okna s archivními daty*.

Scénář úspěšného plnění:

1. Obsluha zvolí možnost *Uložit aktuální rozložení* z horního menu aplikace.
2. Read serializuje nastavení do souboru.
3. Při příštím spuštění Read automaticky načte rozložení z tohoto souboru.

Výjimky ze standardního průběhu:

3a. Konfigurace obsahuje špatný senzor. Buď proto, že byl ze systému vyřazen, nebo na něj uživatel ztratil práva. Tento senzor je odebrán ze všech oken konfigurace, aniž by byl uživatel upozorňován.

3.4 Shrnutí kapitoly

V úvodu této kapitoly jsou identifikovány funkční i nefunkční nedostatky současného řešení. Těchto poznatků je využito v následující kapitole, kde v sekci funkčních požadavků jsou sepsány očekávání uživatelů na funkčnost systému. Technické nároky jsou rozebrány v sekci nefunkčních požadavků. Inspirací pro formulaci požadavků byla také rešeršní část, v které byly definovány požadavky na obecný Systém pro sledování veličin vzhledem k normám a regulacím.

V systému byly identifikovány dvě role uživatelů. V základu jako *Obsluha systému* a s rozšířenými pravomocemi jako *Administrátoři systému*.

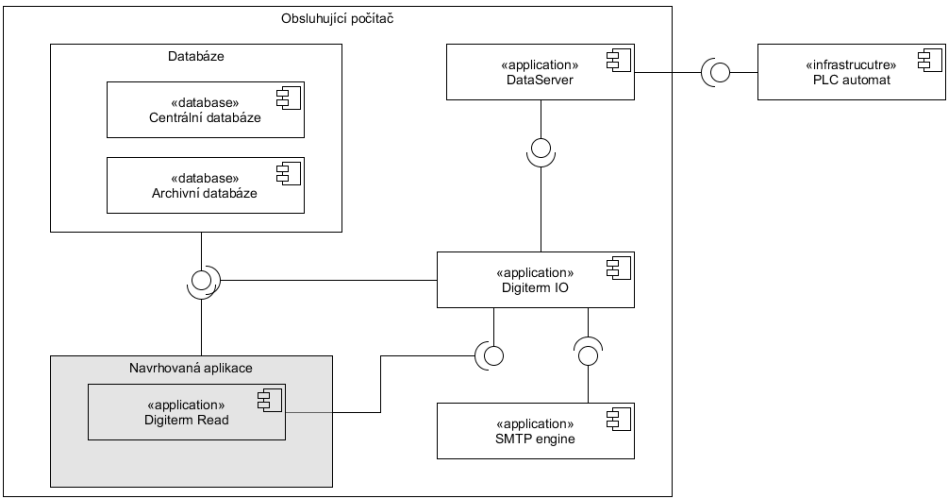
Pro podrobnější představě o funkcích aplikace a snadnější pochopení procesů, jsou v kapitole 3.3 popsány jednotlivé případy užití. Ty jsou rozděleny

do třech logických celků podle funkcionality, kterou řeší - Správa uživatelů, Alarmy a aktuální hodnoty, Práce s daty. Na jejich základě by mělo být možné vytvořit aplikaci tak, aby splnila veškeré funkční požadavky.

Kapitola 4

Implementace

4.1 Komponenty systému



Obrázek 4.1: Diagram komponent systému

4.1.1 PLC automat

Základem systému jsou PLC automaty od společnosti MICROPEL. K těmto automatům jsou připojené jednotlivé senzory, které měří teplotu, popřípadě jiné veličiny. Těchto automatů může být v systému zapojeno i více, jelikož počet připojitelných senzorů je omezený. V automatu je nahraný program zajišťující periodický zápis dat do kruhového stacku. Kapacita stacku je omezená v závislosti na počtu senzorů, připojených k automatu, nicméně i při maxi-

málním počtu připojených sond se data na stacku nezačnou přepisovat dříve jak za týden.

Aby byl systém schopný fungovat i v případě vypnutí obsluhujícího počítače, vyhodnocují se alarmy již v automatu. V případě alarmu je obsluhující personál upozorněn zvukovým signálem. Pokud je to požadavkem, automat v tu chvíli také odesílá prostřednictvím GSM brány informaci o alarmu.

4.1.2 DataServer

DataServer je komunikační aplikace vyvinutá společností Micropel, zprostředkující aplikacím systému Windows komunikovat se sítí automatů (číst a zapisovat data). Pro komunikaci mezi Windows aplikacemi a DataServerem je použité rozhraní DDE.[12]

4.1.3 Digiterm IO

Pro vyčítání dat z automatu slouží Digiterm IO. Ten prostřednictvím DataServeru vyčítá data ze stacku a zapisuje je do databáze. Jeho zodpovědností je také zápis změn alarmů do automatu. V případě alarmu odesílá prostřednictvím aplikace SMTP Engine email obsluze.

4.1.4 Databáze

Data programu můžeme rozdělit do dvou kategorií, která jsou i fyzicky oddělena do dvou různých databází.

4.1.4.1 Archivní databáze

Vytváří se periodicky (typicky po měsíci) a obsahuje naměřené hodnoty za dané časové období. Data v ní nejsou uspořádána relačně, ale jsou všechna uložena do jedné tabulky. Na každém řádku je informace o datu a času vytvoření záznamu a pro každý měřící senzor naměřenou hodnotu a její symptomy. Symptomy měřených hodnot nabývají hodnoty 0-2, viz následující tabulka.

0	první hodnota v řadě
1	normální hodnota
2	neměřená hodnota - senzor byl odpojen

Tabulka 4.1: Symptomy hodnot

Tento model usnadňuje rychlou vizuální kontrolu funkce systému bez nutnosti spouštět program Read.

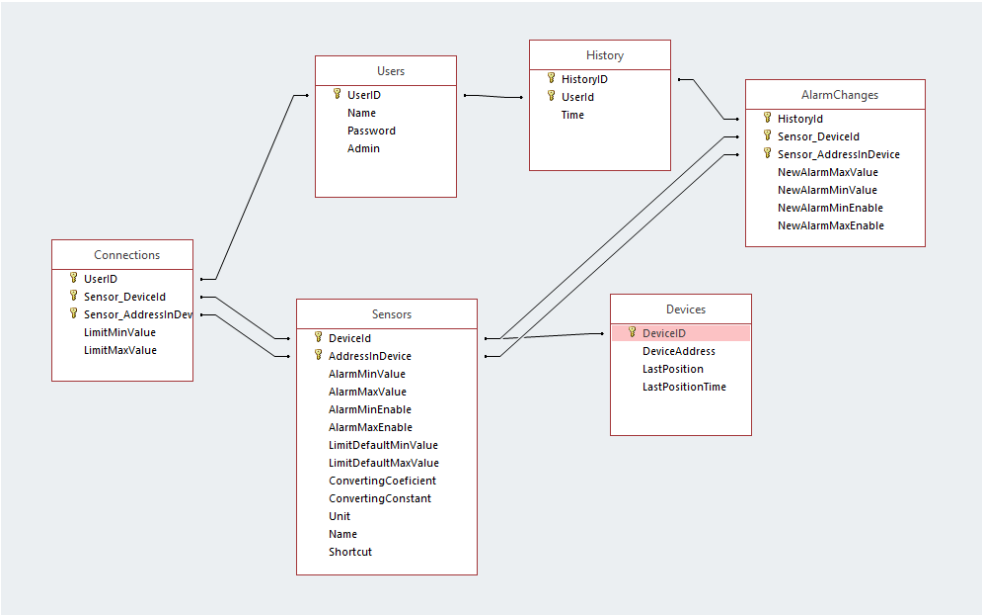
ARCH

DATE_TIME	VAR1	VV1	VAR2	VV2	VAR3	VV3	VAR4	VV4	VAR5
01.03.2017 0:09:00	19,1	1	18,6	1	18,8	1	19	1	18,5
01.03.2017 0:29:00	19,1	1	18,6	1	18,8	1	19	1	18,5
01.03.2017 0:49:00	19,4	1	18,8	1	19	1	19,2	1	18,6
01.03.2017 1:09:00	19,7	1	18,9	1	19,2	1	19,4	1	18,8
01.03.2017 1:29:00	19,9	1	19,1	1	19,4	1	19,6	1	18,9

Obrázek 4.2: Ukázka uložení dat v archivní databázi

4.1.4.2 Centrální databáze

Obsahuje informace o systému samotném - kolik automatů se senzory systém obsahuje, k jakým senzorům mají jednotliví uživatelé oprávnění, nastavení limitů a alarmů. Data jsou uspořádána v klasickém relačním modelu vycházejícím z doménového modelu.



Obrázek 4.3: Schéma centrální databáze

Centrální databáze také obsahuje tabulku s aktuálně měřenými hodnotami. Tato data nejsou uložena relačně, ale stejným modelem, jako se ukládají do archivní databáze.

4.1.5 Digiterm Read

Je vizualizační část systému. V běžném provozu je to jediný program, s kterým obsluha interaguje. Komunikuje pouze s databází, z které vyčítá data k zobrazení. Jeho funkce jsou podrobně popsány v kapitole Analýzy 3.

4.2 Možnosti nového řešení

Nová verze by mohla být vyvinuta jako webová aplikace. Toto řešení by mělo výhodu multiplatformity a rovněž by nebylo náročné na instalaci u uživatele. Pro uživatele by bylo navíc pohodlné, jelikož by k datům měli pohotovější přístup - například z tabletu či mobilního telefonu.

Druhou variantou je zachovat charakter programu jako desktopové aplikace. V této variantě je možné uvažovat navázání na původní kód aplikace psané v Delphi. Jelikož je software vyvíjen pouze na Windows, jako logická alternativa se jeví použití jazyka C# s .Net frameworkem.

4.3 Zvolená Technologie

Změna aplikace na webové řešení by znamenala rozsáhlý zásah do architektury celého systému. Data by se nemohla dále ukládat lokálně na obsluhujícím počítači. Pro mnoho zákazníků by to tak znamenalo náklady navíc a celková změna by byla pracnější. Z těchto důvodů jsem zvolil desktopové řešení. Pokud by se v budoucnu ukázal webový přístup k některým částem aplikace jako nezbytný, je možné ho dopsat jako separátní komponentu běžící na serveru.

Rozhodl jsem se nenavázat na kód napsaný v Delphi a použít C# s .NET frameworkem. K tomuto rozhodnutí mě dovedla hlavně skutečnost, že v dnešní době je vytvářet aplikace pro OS Windows pomocí .NET frameworku standard a podpora je tedy rozhodně rozsáhlejší.

Jako návrhový vzor jsem zvolil Model–view–viewmodel (MVVM), který je založen na separaci logiky programu a grafické uživatelské rozhraní. V praxi to znamená, že pro každý pohled (view) jsem v aplikaci vytvářel soubor XAML se vzhledem pojmenovaný se sufixem *View* a samotnou logiku jsem implementoval do stejně jmenující se třídy se sufixem *ViewModel*.

Kód aplikace jsem rozdělil do tří projektů:

- **Read** - kód samotné aplikace, veškeré pohledy a logika
- **Common** - doménové objekty, připojení do databáze
- **SharedResources** - znovupoužitelné komponenty uživatelského rozhraní, veškeré styly a grafické prvky

4.3.1 Použité knihovny

Všechny závislosti na knihovny 3. stran jsem definoval pomocí NuGet.

4.3.1.1 Entity Framework & JetEntityFrameworkProvider

Pro přípojení do databáze jsem použil Entity Framework. Jelikož ten se neumí v základu připojit k MS Access databázi, použil jsem ještě JetEntityFrameworkProvider.

4.3.1.2 Oxyplot

Jelikož .NET framework implicitně nemá žádnou komponentu pro vykreslování grafů, bylo potřeba vybrat nějakou knihovnu třetí strany, s preferencí open-source řešení. Takové knihovny, které by zároveň podporovaly přibližování dat v grafu, jsem našel pouze dvě - Interactive Data Display a Oxyplot, přičemž první zmíněná již má ukončenou podporu, moje volba padla tedy na druhou zmíněnou.

4.3.1.3 MahApps.Metro

Balíček MahApps.Metro je knihovna, která se snaží přenést moderní *Metro* design i do aplikací WPF.

4.3.1.4 Exceptionless

Pro logování chyb aplikace v produkci jsem zvolil Exceptionless a jejich knihovnu určenou pro WPF. Ocenil jsem zvláště velmi snadné nastavení služby a odesílání informací na jejich server.

```
ExceptionlessClient.Default.SubmitLog(typeof(Program).FullName,↩
    "This is so easy", "Info");
```

4.3.1.5 MvvmLight

Knihovna usnadňující následování návrhové vzoru MVVM.

4.3.2 Actipro Docking & MDI

Hotové řešení pro dokovací okna. Podporuje snadné uložení rozložení oken pomocí serializace a zároveň je kompatibilní s MVVM návrhovým vzorem.

4.4 Databáze

Pro ukládání dat aplikace byla již během vývoje Digiterm-IO zvolena MS Access databáze. Oproti serverovým databázím je její předností především snadná instalace. Soubor databáze můžeme jednoduše zkopírovat do počítače zákazníka, čímž se vyhneme složité inicializaci.

Neserverový charakter databáze má však i své podstatné nevýhody. MS Access databáze používá oproti server-based databázím podstatně větší šířku pásma sítě. To může negativně ovlivnit rychlost aplikace při připojení více uživatelů/načítání většího objemu data. V současnosti není systém instalován u žádného zákazníka, kde by byla tato limitace zásadní. Aplikace by nicméně měla být navržena tak, aby byla v budoucnu a případě potřeby snadno převeditelná na jiný druh databáze. Při použití Entity Frameworku je naštěstí

Obrázek 4.4: Náhled hlavního okna programu

aplikace od typu databáze odstíněna, což tento požadavek do značné míry vyřeší. Pro každou databázi bude pouze potřebovat implementovat rozhraní pro přístup k archivním datům, jelikož tato data nejsou uložena v relační databázi.

4.5 Konfigurace

Cesta k centrální databázi je uložena v souboru *init.txt*. Aby bylo možné provozovat na jednom počítači více systému, může se tento soubor odkazovat na více centrálních databází, každý záznam je na nové řádce. S kterou databází se má pracovat je určeno argumentem *installation* následovaným číslem řádku, tedy např. *DigitermRead.exe installation 2* spustí systém s centrální databází z druhého řádku. Pokud daný řádek je prázdný, uživatel je vyzván k zadání cesty pomocí dialogového okna systému Windows. Uživatelům je vytvořen zástupce s nastaveným parametrem *installation* pro každý systém. Toto nastavení by měl provádět pracovník instalující software zákazníkovi.

Krom cesty k centrální databázi si uživatelé můžou uložit rozložení aplikace (viz případ užití 3.3.4.6 *Uložení rozložení program*). Rozložení jsou ukládány zvlášť pro každý systém do složky *CONF* a jsou pojmenovány podle jména uživatele + čísla systému.

4.6 Shrnutí kapitoly

Kapitola začíná popisem komponent celého systému pro představu, v jakém kontextu aplikace vzniká. Ač by bylo výhodné implementovat novou verzi programu jako webovou aplikaci, není na to systém v současné chvíli připraven. Bylo by také možno navázat na předešlou verzi napsanou v Delphi, nicméně hlavně kvůli slabé podpoře Delphi je nejrozumnější řešení zvolit C# a .NET framework. V něm se vyvine řešení jako WPF desktopová aplikace pro OS Windows.

Implementace následuje návrhový vzor MVVM. Kód aplikace je rozdělen do více projektů podle jejich obsahu. Jelikož implementace využívá mnohé knihovny třetích stran, jsou zde vyjmenovány ty nejdůležitější s jejich popisem.

Kvůli požadavkům na bezproblémovou instalaci se veškeré nastavení aplikace ukládá do konfiguračních souborů. Ze souboru *init.txt* se potom bere informace o cestě k centrální databázi.

Kapitola 5

Ověření řešení

5.1 Metodika vývoje

Aby výsledná aplikace co nejlépe splňovala očekávání, byl během celého vývoje udržován aktivní kontakt se zadavatelem. Ač za vývojem produktu nestál tým vývojářu a nároky na organizaci práce tedy nebyly vysoké, snažil se vývoj dodržovat základy agilního vývoje. Nově vyvinuté vlastnosti aplikace byly zhruba v čtrnáctidenních cyklech prezentovány zadavateli, který tak měl možnost usměrňovat vývoj.

Pro uchování historií změn a možnosti pracovat na více verzích kódu zároveň, byl použit systém pro správu verzí Git. Při používání Gitu je běžné nastavit si pravidla větvení kódu a ta poté dodržovat, jako je například popsáno zde [13]. Jelikož aplikaci programoval pouze jeden vývojář, vývoj obvykle postupoval lineárně a nebylo třeba nastavovat pravidla větvení zbytečně sofistikovaná. Jediný zásadní požadavek byl, aby v každé chvíli vývoje byla dostupná stabilní verze programu. Větev *master* proto vždy obsahovala otestovanou aplikaci, kam byl kód mergován z větvě *develop*, v které probíhal vývoj a testování.

5.2 Testování aplikace

Program byl testován vývojářem vždy po vyvinutí nových vlastností. Před demonstrací aplikace zadavateli na konci každé iterace, proběhly krátké regresní testy, nicméně byly spíš nahodilé a nebyl vytvořený žádný systematický popis, jak je provádět. Zadavatel pak otestoval po předvedení nových schopností aplikace a chyby komunikoval zpět spolu s požadavky na změny.

Pro sledování chyb v programu bylo použito několik služeb, ale nakonec se jako vhodný nástroj ukázal *Visual Studio Team Services*, kde byl i hostován Git repositář. Chyby aplikace sem byly vypisovány buď manuálně, popřípadě reportováním přímo z aplikace.

Během vývoje byly bohužel opomíjeny automatické testy a bylo vyvinutou pouze malé množství Unit testů.

5.3 Stav aplikace

Aplikace je v tuto chvíli hotová a je nasazená u menšího počtu zákazníků, od kterých se sbírají požadavky na změny a chybová hlášení. Počítá se s jejím masovějším nasazením do produkce v horizontu několika měsíců poté, co se zmenší množství hlášených chyb. Program implementuje všechny identifikované případy užití. U některých bylo nalezeno, že tak jak byly originálně popsány nevyhovují zcela požadavkům a byly tedy zpětně opraveny.

Zpětná vazba od zákazníků, kteří již s programem přišli do kontaktů byla veskrze pozitivní a povzbuzující. Podařilo se program navrhnout tak, aby jeho ovládání bylo intuitivnější. Zároveň rozšířil schopnosti systému, takže zákazníkům odpadly některé povinnosti jako třeba vést záznam změn alarmů, což nově dělá program samotný.

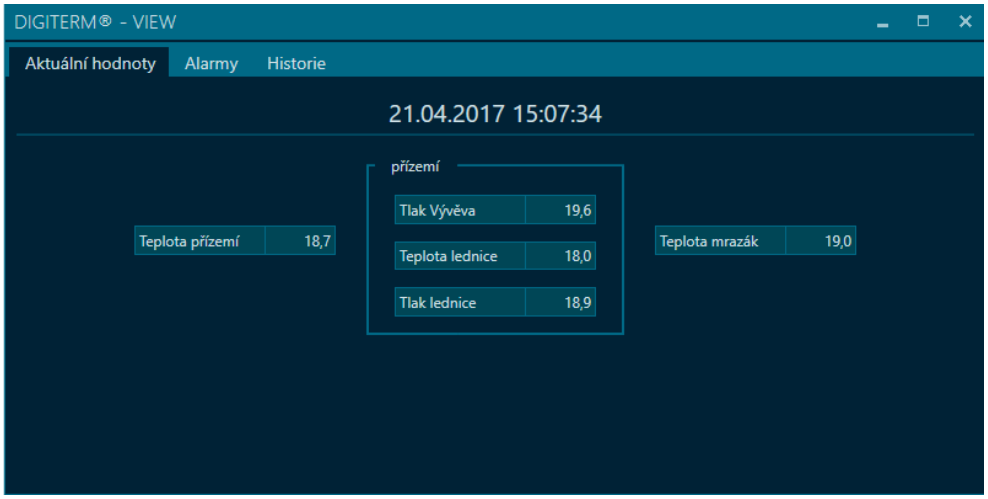
Rychlost programu je uspokojující. Bez problémů se podařilo překonat požadovaný limit 5 vteřin pro start a inicializaci programu. Veškeré nastavení je pouze formou konfiguračních souborů, takže také odpadá problém se zasahováním do registrů. Všechny konfigurace se navíc ukládají do stejné složky jako program samotný a je tedy možné ho provozovat v Portable verzi.

5.4 Možnosti rozšíření aplikace

Systému citelně chybí schopnost uchovávat historii nastalých alarmů a obsluha musí tento záznam vést tedy manuálně a mimo systém. Pro tuto funkcionalitu by bylo nicméně potřeba upravit i jiné komponenty systému - Digiterm IO a strukturu centrální databáze. V programu Read by pak byl potřeba vymyslet způsob, jak tento seznam zobrazit a umožnit uživatelům přidávat poznámky.

Pokud by se program měl dále rozvíjet, bylo by vhodné doplnit automatizované testy. Mohly by se tím navíc otevřít dveře ke *Continuous integration* (automatickému testování kódu po každé větší změně) popřípadě i k *Continuous delivery* (automatizovanému vydávání nových verzí programu).

Během vývoje se často naráželo na limity Access databáze, která se pod zátěží nechovala naprosto stabilně a občas se dokonce soubor databáze poškodil natolik, že ho bylo třeba znovu vytvořit. Do budoucna by tedy bylo rozumné přejít k jiné databázi, například MS SQL. Je možné také uvažovat nad modelem, kdy by Read k datům nepřistupoval přímo do databáze, ale získával by je přes RESTový endpoint vystaveným od Digiterm IO. Každopádně by to znamenalo změnu filozofie programu, která nyní počítá s tím, že data jsou rozdělena do souborů.



Obrázek 5.1: Náhled okna aktuálních hodnot

5.5 Shrnutí kapitoly

Implementace řešení přinesla nutnost částečně upravit navržené případy užití. Zásahy nicméně byly marginální a v celku se navrhované řešení ukázalo jako validní. To potvrzuje i kladný ohlas od zákazníků, kteří měli možnost si program již sami vyzkoušet. Podařilo se vyhovět všem nefunkčním požadavkům na program.

Závěr

Ve své práci jsem se snažil navrhnout a implementovat nové vizualizační prostředí pro systém Digiterm. Mým cílem bylo odstranit déle neřešené problémy této aplikace tak, aby se zlepšil uživatelský dojem a usnadnil uživatelům ovládání. Práce by zároveň měla sloužit jako podklad k orientaci v problematice pro někoho, kdo by na moji práci chtěl navazovat úpravou či rozšířením aplikace. Výsledkem mé práce měl být stabilní program připravený pro nasazení do ostrého provozu.

Abych naplnil vytyčené cíle práce, identifikoval jsem nejprve problémy současného řešení. Na jejich základě a díky poznatkům zjištěných v rešeršní části jsem poté definoval funkční a nefunkční požadavky na program. Funkční požadavky jsem podrobně rozepsal pomocí případů užití, které tak byly dostatečným podkladem pro implementaci řešení.

Řešení jsem se rozhodl implementovat jako WPF desktopovou aplikaci pro OS Windows napsanou v C#. Pro usnadnění mé práce jsem vyhledal volně dostupné knihovny, které vyřešily části aplikace. Implementaci jsem se snažil programovat tak, aby byl kód dobře čitelný a následoval osvědčené metody a aplikace tak byla snadno rozšiřitelná.

Výsledkem mé práce je program, který zahrnuje všechny definované případy užití. Od uživatelů, kteří jej dostali k užívání, jsem obdržel pozitivní zpětnou vazbu a pochvalují si snadnější ovládání oproti předchozí verzi. Program nevyžaduje instalaci a je možné ho provozovat jako Portable aplikaci. Poté, co bude aplikace nasazená v testovacím provozu, bude v řádu měsíců připravena i do ostrého nasazení. Úspěšně jsem tedy splnil všechny cíle své práce.

Do budoucna by systém měl být rozšířen o historii nastalých alarmů, jelikož v současné době ji musí obsluha vést mimo systém. Stejně tak by bylo vhodné přenést systém na stabilnější databázovou technologii. Pro masovější další rozvoj by také měly býti doplněny automatické testy. Tato práce snad poslouží, jako základ pro tento další rozvoj.

Literatura

[1] Comet System s.r.o.: Software pro sběr dat a analýzu naměřených hodnot [online]. [cit. 18.4.2017]. Dostupné z: http://www.cometsystem.cz/produkty/software-cz

[2] doc. Ing. Jiří Horský, C.: Revize normy ISO/IEC 17025 pro akreditované kalibrační laboratoře a zkušebny. *Automa*, ročník 2015, č. 12, 2015: s. 25–26.

[3] UNMZ: ISO 9000, ISO 9001, ISO 9004 [online]. [cit. 12.2.2017]. Dostupné z: http://www.unmz.cz/test/normy-serie-iso-9001-a-jejich-aplikace

[4] ISO: ISO members [online]. [cit. 12.2.2017]. Dostupné z: http://www.iso.org/iso/home/about/iso_members.htm

[5] ISO: Czech Republic (UNMZ) [online]. [cit. 12.2.2017]. Dostupné z: http://www.iso.org/iso/home/about/iso_members/iso_member_body.htm?member_id=2133

[6] UNMZ: Co je to technická norma? [online]. [cit. 20.2.2017]. Dostupné z: http://www.unmz.cz/urad/co-je-to-technicka-norma-

[7] Ing. Pavlát, J.: Technické normy v oblasti investiční výstavby [online]. [cit. 5.3.2017]. Dostupné z: http://www.pavlat-znalec.cz/investing/stpr/stpr/stpr09.html

[8] Český institut pro akreditaci, o.p.s.: Akreditace certifikačních orgánů certifikujících systémy managementu [online]. [cit. 10.3.2017]. Dostupné z: http://www.cia.cz/media/51666/zia-list_cosm_unor-2015.pdf

[9] ČSN EN ISO 9001: Systémy managementu kvality. Technická zpráva, Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, Praha, 2016.

[10] Státní ústav pro kontrolu léčiv: DIS-15 verze 3 Sledování a kontrola teploty při skladování a přepravě léčiv [online]. 2013, [cit. 20.2.2017]. Dostupné z: http://www.sukl.cz/file/76245_1_1

[11] Cockburn, A.: *Writing effective use cases*. Addison-Wesley, 2001, ISBN 02-017-0225-8. Dostupné z: http://alistair.cockburn.us/get/2465

[12] MICROPEL: DataSERVER [online]. 2016. Dostupné z: http://www.micropel.cz/images/stories/Dokumenty/DataSERVER.pdf

[13] Vincent Driessen: A successful Git branching model [online]. [cit. 2.5.2017]. Dostupné z: http://nvie.com/posts/a-successful-git-branching-model/

[14] Loeliger, J.; McCullough, M.: *Version control with Git, Second Edition*. Sebastopol, USA: O'Reilly, 2012, ISBN 978-1-449-31638-9.

[15] TutorialsPoint: UML - Statechart Diagrams [online]. [cit. 1.5.2017]. Dostupné z: https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm

[16] Sells, C.; Griffiths, I.: *Programming WPF, Second Edition*. Sebastopol, USA: O'Reilly, 2007, ISBN 978-0-596-51037-4.

[17] Miles, R.; Hamilton, K.: *Learning UML 2.0*. Sebastopol, USA: O'Reilly, 2006, ISBN 978-0-596-00982-3.

[18] MSDN: The MVVM Pattern [online]. [cit. 10.12.2016]. Dostupné z: https://msdn.microsoft.com/en-us/library/hh848246.aspx

[19] Ing Mlejnek, J.: Modelování požadavků. ČVUT, Praha, Přednáška BI-SI1, LS 2015/16.

[20] Oxyplot: Oxyplot [online]. [cit. 5.12.2016]. Dostupné z: http://www.oxyplot.org/

[21] Dynamic Data Display: Dynamic Data Display [online]. [cit. 5.12.2016]. Dostupné z: http://dynamicdatadisplay.codeplex.com/

[22] MSDN: Introduction to Entity Framework [online]. [cit. 10.12.2016]. Dostupné z: http://dynamicdatadisplay.codeplex.com/

Příloha A

Seznam použitých zkratek

WPF Windows Presentation Foundation

XML Extensible markup language

OS Operační systém

REST Representational state transfer

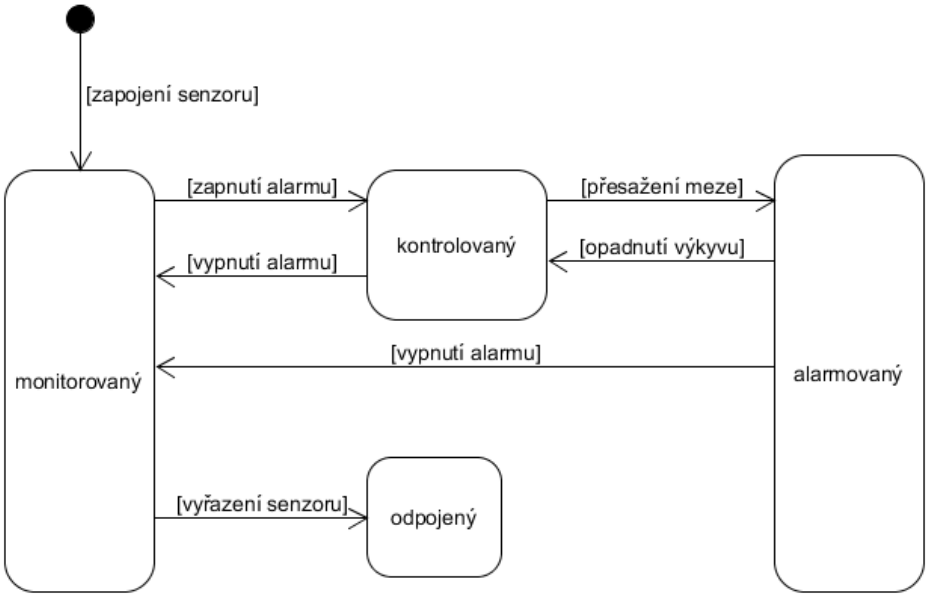
DDE Dynamic Data Exchange

GB Gigabyte

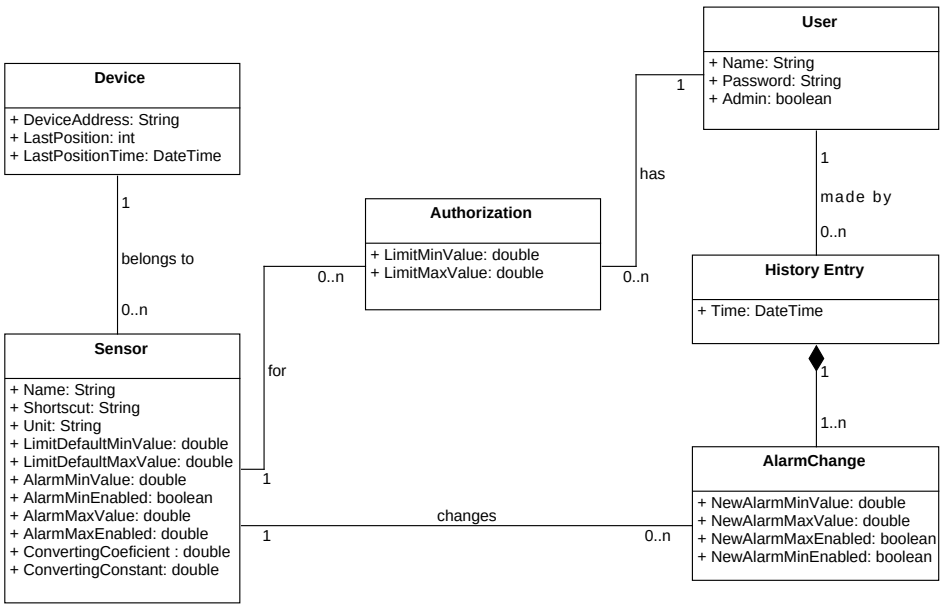
RAM Random-access memory

Příloha B

Přílohy



Obrázek B.1: Stavový diagram senzoru



Obrázek B.2: Doménový model programu

```
namespace Digiterm.Common.Infrastructure
{
  public abstract class AbstractConfiguration<T> where T : ↩
    AbstractConfiguration<T>, new()
  {
    private static readonly object syncRoot = new object();
    protected static volatile T Configuration;

    public static T Default
    {
      get
      {
        if (Configuration != null) return Configuration;
        lock (syncRoot)
        {
          if (Configuration == null)
            Configuration = (T)GetConfigurationProvider().↩
    GetConfiguration<T>();
        }

        return Configuration;
      }
    }


    private static IConfigurationsProvider ↩
    GetConfigurationProvider()
    {
      try
      {
        return ServiceLocator.Current.GetInstance<↩
    IConfigurationsProvider>();
      }
      catch (Exception)
      {
        return new XmlConfigurationProvider();
      }
    }

    public void Save()
    {
      GetConfigurationProvider().Save(Configuration);
    }
  }
}
```

Obrázek B.3: Abstraktní generická třída *AbstractConfiguration*, z které dědí třídy zajišťující konfiguraci systému. Pomocí *ServiceLocatoru* získá použitý *ConfigurationProvider*, který za běhu vrací *XmlConfigurationProvider*, pro testy ale existuje *MockupConfigurationProvider*.

```
namespace SharedResource.ViewModel
{
  public class BaseViewModel : ObservableObjectBase, IDisposable, ↩
    IBusyAware
  {
    public IWindow Window { get; set; }

    protected IExceptionHandler ExceptionHandler;
    protected INotificationComponent NotificationComponent;
    protected List<Window> Windows;

    private int _busyCounter;
    public bool IsBusy
    {
      get { return _busyCounter > 0; }
      set
      {
        if (value)
          _busyCounter++;
        else
          _busyCounter--;
        OnPropertyChanged();
        CommandManager.InvalidateRequerySuggested();
      }
    }

    private RelayCommand<Window> _closeCommand;
    public RelayCommand<Window> CloseCommand => _closeCommand ?? (↩
    _closeCommand = new RelayCommand<Window>(Close, null));
    private RelayCommand _onClosingCommand;

    public BaseViewModel()
    {
      this.Windows = new List<Window>();
      ExceptionHandler = ServiceLocator.Current.GetInstance<↩
    IExceptionHandler>();
      NotificationComponent = ServiceLocator.Current.GetInstance<↩
    INotificationComponent>();
    }

    public RelayCommand OnClosingCommand => _onClosingCommand ?? (↩
    _onClosingCommand = new RelayCommand(Dispose));

    protected virtual void Close(Window window = null)
    {
      if (window == null)
      {
        Window?.GetWindow().Close();
      }
      else
        window.Close();
    }


    public virtual void Dispose()
    {
    }

    public void StartBusy()
    {
      IsBusy = true;
    }

    public void SopBusy()
    {
      IsBusy = false;
    }
  }
}
```

Obrázek B.4: Abstraktní třída, která je předkem všem třídám *ViewModel*.

```
namespace Digiterm.Common.Infrastructure.Busy
{
  public class BusyHandler : IDisposable
  {
    public static BusyHandler Start(IBusyAware busyAware)
    {
      return new BusyHandler(busyAware);
    }

    private readonly IBusyAware _busyAware;
    private BusyHandler(IBusyAware busyAware)
    {
      _busyAware = busyAware;
      _busyAware.StartBusy();
    }

    public void Dispose()
    {
      _busyAware.SopBusy();
    }
  }
}
```

Obrázek B.5: Třída *BusyHandler* pomáhá s indikací zaneprázdněnosti. Implementuje rozhraní metodu *Dispose* rozhraní *IDisposable*, která je automaticky volána při použití v bloku u (viz *LimitsViewModel*). Není tedy třeba obalovat kód, během kterého je třída zaneprázdněna, blokem *try{...}finaly{IsBusy=false;}*, ale stačí implementovat *IBusyAware* a poté obalit kód blokem *using(BusyHandler.Start(this)){...}*.

```
namespace Digiterm.Common.Infrastructure.Busy
{
  public interface IBusyAware
  {
    void StartBusy();
    void SopBusy();
  }
}
```

Obrázek B.6: Rozhraní pro *BusyHandler*.

```
    <Grid MaxHeight="{Binding ElementName=LimitsViewContext, ↩
Path=ActualHeight, Converter={StaticResource MinusConverter↩
}, ConverterParameter=80}" >
      <ScrollViewer VerticalScrollBarVisibility="Auto" ↩
Padding="0,0,0,5">
        <ItemsControl ItemsSource="{Binding Connections}" x:↩
Name="ItemsControl" >
          <ItemsControl.ItemTemplate>
            <DataTemplate>
              <Grid Margin="0,12,0,0" >
                <Grid.ColumnDefinitions>
                  <ColumnDefinition Width="*"/>
                  <ColumnDefinition Width="92" />
                  <ColumnDefinition Width="62"/>
                  <ColumnDefinition Width="68"/>
                  <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <TextBlock Text="{Binding Sensor.Name}" Grid.↩
Column="1" Margin="12,0,0,0" Width="80" VerticalAlignment="↩
Center"></TextBlock>
                <reusable:ClickSelectTextBox Background="{↩
StaticResource UnderLimit}" IsEnabled="{Binding ElementName↩
=LimitsViewContext, Path=IsBusy, Converter={StaticResource ↩
InverseBoolConverter}}" Text="{Binding LimitMinValue, ↩
StringFormat=N1, ValidatesOnDataErrors=True}" Margin="↩
12,0,0,0" Grid.Column="2"></reusable:ClickSelectTextBox>
                <reusable:ClickSelectTextBox Background="{↩
StaticResource AboveLimit}" IsEnabled="{Binding ElementName↩
=LimitsViewContext, Path=IsBusy, Converter={StaticResource ↩
InverseBoolConverter}}" Text="{Binding LimitMaxValue, ↩
StringFormat=N1, ValidatesOnDataErrors=True}" Margin="↩
6,0,12,0" Grid.Column="3"></reusable:ClickSelectTextBox>
              </Grid>

            </DataTemplate>
          </ItemsControl.ItemTemplate>
        </ItemsControl>
      </ScrollViewer>
    </Grid>
```

Obrázek B.7: Ukázka z pohledu definujícího okno nastavení limitů.

```
public LimitsViewModel()
{
  Connections = new ObservableCollection<Connection>();
  _newToOriginal = new Dictionary<Connection, Connection>();
  Initialize();
}

private async void Initialize()
{
  using (BusyHandler.Start(this))
  {
    Connections.Clear();
    var conn = await ApplicationViewModel.Instance.GetConnectionsAsync↩
();
    foreach (var connection in conn)
    {
      var c = new Connection
      {
        Sensor = connection.Sensor,
        LimitMaxValue = connection.LimitMaxValue,
        LimitMinValue = connection.LimitMinValue
      };
      _newToOriginal.Add(c, connection);
      Connections.Add(c);
    }
  }
}

private RelayCommand _saveCommand;
public RelayCommand SaveCommand => _saveCommand ?? (_saveCommand = new ↩
RelayCommand(SaveCommandExecuted));

public async void SaveCommandExecuted()
{
  using (BusyHandler.Start(this))
  {
    var error = String.Join("\n",
      Connections.Where(i => !i.IsValid()).Select(i => $"{i.Sensor.Name↩
}: {i.Error}"));
    if (error != String.Empty)
    {
      NotificationComponent.HandleError(error, Window.GetWindow());
      return;
    }
    var provider = ServiceLocator.Current.GetInstance<ILimitsProvider↩
>();
    var result = await provider.ChangeLimitsAsync(Connections, ↩
ApplicationViewModel.Instance.User);
    if (result.IsFaulted)
    {
      NotificationComponent.HandleError(result.GetErrorMesage(), Window↩
.GetWindow());
      return;
    }

    foreach (var connection in Connections)
    {
      Connection orig;
      if (!_newToOriginal.TryGetValue(connection, out orig)) continue;
      orig.LimitMaxValue = connection.LimitMaxValue;
      orig.LimitMinValue = connection.LimitMinValue;
    }

    Close();
  }
}
```

Obrázek B.8: Ukázka z *LimitsViewModel* implementující logiku nastavování limitů.

```
namespace Digiterm.Common.Model
{
  public class User
  {
    public User()
    {
    }

    public int UserID { get; set; }
    public string Name { get; set; }
    public string Password { get; set; }
    public bool Admin { get; set; }

    public virtual ICollection<AlarmChange> AlarmChange { get; set; }
    public virtual ICollection<Connection> Connection { get; set; }

    public override bool Equals(object obj)
    {
      var user = obj as User;
      return UserID == user?.UserID;
    }
  }
}
```

Obrázek B.9: Třída doménového objektu *User*.

```
namespace Digiterm.Common.Model
{
  public class Device
  {
    [Obsolete]
    public Device()
    {
      LastPositionTime = SqlDateTime.MinValue.Value;
    }

    public Device(int id) : this()
    {
      this.Sensors = new List<Sensor>();
      DeviceID = id;
    }

    [Key]
    public int DeviceID { get; set; }
    public string DeviceAddress { get; set; }
    public int LastPosition { get; set; }
    public DateTime LastPositionTime { get; set; }

    public virtual List<Sensor> Sensors { get; set; }

    public override bool Equals(object obj)
    {
      var other = obj as Device;
      return other?.DeviceID == DeviceID;
    }
  }
}
```

Obrázek B.10: Třída doménového objektu *Device*.

```
class GraphHistory
{
  public GraphHistory()
  {
    PostitionInHistory = 0;
  }

  public int PostitionInHistory
  {
    get { return _postitionInHistory; }
    private set
    {
      _postitionInHistory = value;
      CommandManager.InvalidateRequerySuggested();
    }
  }

  private readonly Collection<HistoryGraphPosition> _graphHistory = new ↩
    Collection<HistoryGraphPosition>();
  private int _postitionInHistory;

  public void AddStateToHistory(HistoryGraphPosition position, bool ↩
    isFromButton = false)
  {
    while (PostitionInHistory < _graphHistory.Count - 1)
    {
      _graphHistory.RemoveAt(_graphHistory.Count - 1);
    }
    if (_graphHistory.Count != 0 && !isFromButton)
    {
      if ((DateTime.Now - _graphHistory[_graphHistory.Count - 1].DateTime).↩
    Milliseconds > 200)
      {
        _graphHistory.Add(position);
        PostitionInHistory++;
      }
      else
      {
        _graphHistory[_graphHistory.Count - 1] = position;
      }
    }
    else
    {
      _graphHistory.Add(position);
      PostitionInHistory++;
    }
  }

  public void ResetHistory()
  {
    _graphHistory.Clear();
    PostitionInHistory = 0;
  }

  public bool CanGoBack()
  {
    return !(_graphHistory.Count == 0 || PostitionInHistory == 0);
  }

  public HistoryGraphPosition Back()
  {
    if (_graphHistory.Count == 0 || PostitionInHistory == 0)
      return null;
    PostitionInHistory--;

    return _graphHistory[PostitionInHistory];
  }
```

Obrázek B.11: Ukázka z třídy uchovávající pohyb v grafu.

```
namespace SharedResource.Infrastructure
{

  public static class ReportHelper
  {
    static ReportHelper()
    {
      HistoryItems = new NotificationHistory(5000);
    }

    private static readonly NotificationHistory HistoryItems;

    public static void OnSubmittingEvent(object sender, ↩
    EventSubmittingEventArgs e)
    {
      if (!e.IsUnhandledError)
        return;

      Show(() =>
      {
        ExceptionReportingView erv = new ExceptionReportingView(e.Event);
        return ShowDialog(erv);
      }
      );
    }


    public static void Report(Exception exception)
    {
      if(HistoryItems.WasRecentlyNotified(exception)) return;

      Show(() =>
      {
        ExceptionReportingView erv = new ExceptionReportingView(exception);
        return ShowDialog(erv);
      }
      );
    }

    private static bool Show(Func< bool> action)
    {
      if (Application.Current == null) return true;
      if (!Application.Current.Dispatcher.CheckAccess())
        return !(bool)Application.Current.Dispatcher.Invoke(action, ↩
    DispatcherPriority.Send);
      else
        return !action();
    }

    private static bool ShowDialog(ExceptionReportingView dialog)
    {
      bool? result = dialog.ShowDialog();
      return
        result.HasValue && result.Value;
    }
  }
}
```

Obrázek B.12: Třída obalující funkci reportování chyb aplikace a odesílání zprávy na. Při opakování stejné chyby na ní není uživatel reportován dříve, jak po 5000 *ms*.

```
namespace Digiterm.Common.Utilities
{
  public static class DeepCopy
  {
    public static T DeepClone<T>(this T obj)
    {
      using (var ms = new MemoryStream())
      {
        var formatter = new DataContractSerializer(typeof(T));
        formatter.WriteObject(ms, obj);
        ms.Position = 0;

        return (T)formatter.ReadObject(ms);
      }
    }
  }
}
```

Obrázek B.13: Třída pro vytváření hlubokých kopií objektů za použití serializace.

```
namespace Digiterm.Common.Service.Data
{
  public interface IArchiveDataProvider
  {
    Task<ICommunicationResult<DataTable>> LoadAllDataForSensorsAsync(↩
    ICollection<Model.Sensor> sensors);
    Task<ICommunicationResult<DataTable>> LoadDataToDateForSensorsAsync(↩
    ICollection<Model.Sensor> sensors, DateTime date);
    Task<ICommunicationResult<DataTable>> LoadDataSinceDateForSensorsAsync(↩
    ICollection<Model.Sensor> sensors, DateTime date);
    Task<ICommunicationResult<DateTime>> GetTimeOfLastRecordAsync();
  }
}
```

Obrázek B.14: Rozhraní pro zdroj archivních dat.

```
{
  public interface IArchiveDataAware
  {
    Task OnNewArchiveData();
    Task OnDatabaseChange();
    Task OnDatabaseJoined(string path);
  }
}
```

Obrázek B.15: Toto rozhraní implementují třídy, které mají býti upozorňovány na archivní data.

```
public class DataManager : ObservableObjectBase
{
  private static DataManager _instance;

  public static DataManager Instance => _instance ?? (_instance = new ↩
    DataManager());

  private readonly ConcurrentDictionary<int,IArchiveDataAware> ↩
    _archiveDataAwares;
  private string _pathToDatabase;
  public string DatabaseName => Path.GetFileNameWithoutExtension(↩
    PathToDatabase);
  public string PathToDatabase
  {
    get { return _pathToDatabase; }
    private set
    {
      _pathToDatabase = value;
      OnPropertyChanged();
      OnPropertyChanged(nameof(DatabaseName));
    }
  }

  private DataManager()
  {
    this._archiveDataAwares = new ConcurrentDictionary<int, ↩
    IArchiveDataAware>();
  }

  public bool OnDatabaseChanged(string path)
  {
    if (!CheckDatabaseFormat(path)) return false;
    _cancellationTokenSource?.Cancel();
    PathToDatabase = path;
    SimpleIoc.Default.Unregister<IArchiveDataProvider>();
    SimpleIoc.Default.Register<IArchiveDataProvider>(() => new ↩
    AccessArchiveDatabaseProvider(path));
    ConfigurationHelper.Instance.SavePathToDatabase(PathToDatabase);
    foreach (var archiveDataAware in _archiveDataAwares.Values)
    {
      archiveDataAware.OnDatabaseChange();
    }
    Reinitialize();
    return true;
  }

  public void Attach(IArchiveDataAware archiveDataAware)
  {
    _archiveDataAwares.GetOrAdd(archiveDataAware.GetHashCode(),↩
    archiveDataAware);
  }

  public void Detach(IArchiveDataAware archiveDataAware)
  {
    _archiveDataAwares.TryRemove(archiveDataAware.GetHashCode(), out ↩
    archiveDataAware);
  }

  public async Task UpdateData()
```

Obrázek B.16: Singleton *DataManager* je třída, ke které se mohou připojit třídy implementující *IArchiveDataAware* a jsou potom upozorňovány na změny ohledně databáze - otevření nové, dočtení nových dat.

```
public abstract class DataDocumentItemViewModel : ↩
  DockingItemViewModelBase, IArchiveDataAware, IDropTarget, IBusyAware
{
  private readonly INotificationComponent _notificationComponent;
  private DateTime _lastRecordLoaded;
  public override bool IsTool => false;

  public EWindowType WindowType { get; protected set; }

  public Task OnNewArchiveData()
  {
    return Task.Run(() => LoadNewData());
  }

  public Task OnDatabaseChange()
  {
    return Task.Run(() =>
    {
      ClearAllData();
      _lastRecordLoaded = DateTime.MinValue;
      var archiveDataProvider = ServiceLocator.Current.GetInstance<↩
  IArchiveDataProvider>();
      LoadDataForAddedSensors(SelectedConnections, archiveDataProvider);
    });
  }

  public Task OnDatabaseJoined(string path)
  {
    return LoadDataForAddedSensors(SelectedConnections, new ↩
  AccessArchiveDatabaseProvider(path));
  }

  protected async Task LoadDataForAddedSensors(ICollection<↩
  ConnectionViewModel> connectionViewModels, IArchiveDataProvider ↩
  archiveDataProvider)
  {
    using (BusyHandler.Start(this))
    {
      ICommunicationResult<DataTable> result;

      if (_lastRecordLoaded == DateTime.MinValue)
      {
        result =
          await
            archiveDataProvider.LoadAllDataForSensorsAsync(
              connectionViewModels.Select(i => i.DomainObject.Sensor).↩
  ToList());
      }
      else
      {
        result =
          await
            archiveDataProvider.LoadDataToDateForSensorsAsync(
              connectionViewModels.Select(i => i.DomainObject.Sensor).↩
  ToList(), _lastRecordLoaded);
        if (!result.IsFaulted)
          _lastRecordLoaded = result.GetValueOrDefault().AsEnumerable().↩
  Max(row => (DateTime)row[0]);
      }
      if (result.IsFaulted)
```

Obrázek B.17: Abstraktní třída *DataDocumentItemViewModel* je předkem pro *ViewModel* za grafem a tabulkou. Implementuje rozhraní *IArchiveDataAware*, a je tak upozorňován na nové data v archivní databázi a další změny při práci s daty.

```
namespace Digiterm.Read.Components.History.ViewModel
{
  class HistoryViewModel : BaseViewModel
  {
    private ObservableCollection<AlarmChangeDTO> _changes;

    public ObservableCollection<AlarmChangeDTO> Changes
    {
      get { return _changes; }
      set
      {
        _changes = value;
        OnPropertyChanged();
      }
    }

    private RelayCommand _refreshCommand;
    public RelayCommand RefreshCommand => _refreshCommand ?? (↩
    _refreshCommand = new RelayCommand(InitializeAsync));

    private DateTime _lastChangeTime;

    public HistoryViewModel()
    {
      Changes = new ObservableCollection<AlarmChangeDTO>();
      _lastChangeTime = DateTimeConstatns.Min;
    }

    public async void InitializeAsync()
    {
      if(IsBusy) return;

      using (BusyHandler.Start(this))
      {
        var history = await ServiceLocator.Current.GetInstance<↩
    IAlarmsProvider>().GetAlarmChangesAsync(_lastChangeTime,
          ApplicationViewModel.Instance.User.Name,Crypto.DecryptStringAES(↩
    ApplicationViewModel.Instance.User.Password,Crypto.SECRET));
        if (history.IsFaulted)
        {
          NotificationComponent.HandleError(history.GetErrorMesage());
        }
        else
        {
          Changes.AddRange(history.GetValueOrDefault());
          if(Changes.Count > 0)
          _lastChangeTime = Changes.Max(i => i.Time);
        }
      }
    }
  }
}
```

Obrázek B.18: *ViewModel* pro pohled historie změn alarmů.

```
<Grid Margin="0,0,-272,0">
  <controls:CircularProgressBar ProgressColor="{x:Static constants:↩
   ColorsHelper.ActiveBrush}" Visibility="{Binding IsBusy, Converter={↩
   StaticResource BooleanToVisibilityConverter}}" />
  <DataGrid x:Name="HistoryGrid" Margin="5" CanUserReorderColumns="False"↩
    CanUserAddRows="False" CanUserDeleteRows="False" ItemsSource="{Binding↩
    Changes}"
                  VerticalAlignment="Stretch" HorizontalAlignment="Left" ↩
   HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto"
       AutoGenerateColumns="False"
                  Background="{StaticResource bckgrd}" BorderBrush="↩
   Transparent" Foreground="{StaticResource menu_text}" >
    <DataGrid.Columns>
      <DataGridTextColumn Header="{x:Static properties:Strings.↩
   LBL_DateTime}" Binding="{Binding Time}"></DataGridTextColumn>
      <DataGridTextColumn Header="{x:Static properties:Strings.LBL_User}"↩
    Binding="{Binding UserName}"></DataGridTextColumn>
      <DataGridTextColumn Header="{x:Static properties:Strings.LBL_Sensor↩
   }" Binding="{Binding SensorName}"></DataGridTextColumn>
      <DataGridCheckBoxColumn Visibility="{Binding Source={x:Static ↩
   infrastructure:ReadConfiguration.Default},
        Path=BothAlarmsVisible, Converter={StaticResource ↩
   InverseBooleanToVisibilityConverter}}"
        Header="{x:Static properties:Strings.LBL_NewAlarmiEnable }" ↩
   Binding="{Binding NewAlarmMaxEnable}"></DataGridCheckBoxColumn>

      <DataGridCheckBoxColumn Visibility="{Binding Source={x:Static ↩
   infrastructure:ReadConfiguration.Default},
        Path=BothAlarmsVisible, Converter={StaticResource ↩
   BooleanToVisibilityConverter}}"
        Header="{x:Static properties:Strings.LBL_NewAlarmiMinEnable }" ↩
   Binding="{Binding NewAlarmMinEnable}"></DataGridCheckBoxColumn>
      <DataGridTextColumn Header="{x:Static properties:Strings.↩
   LBL_NewAlarmiMinValue}" Binding="{Binding NewAlarmMinValue, ↩
   StringFormat=F1}"></DataGridTextColumn>
      <DataGridCheckBoxColumn Visibility="{Binding Source={x:Static ↩
   infrastructure:ReadConfiguration.Default},
        Path=BothAlarmsVisible, Converter={StaticResource ↩
   BooleanToVisibilityConverter}}"
        Header="{x:Static properties:Strings.LBL_NewAlarmiMaxEnable1 }" ↩
   Binding="{Binding NewAlarmMaxEnable}"></DataGridCheckBoxColumn>
      <DataGridTextColumn Header="{x:Static properties:Strings.↩
   LBL_NewAlarmiMaxValue}" Binding="{Binding NewAlarmMaxValue, ↩
   StringFormat=F1}"></DataGridTextColumn>

    </DataGrid.Columns>
  </DataGrid>
</Grid>
```

Obrázek B.19: Definice rozložení okna zobrazujícího historii změn alarmů.

```
public class AccessArchiveDatabaseProvider : ↩
    AccessDatabaseWithRecordsProvider, IArchiveDatabaseProvider, ↩
    IArchiveDataProvider
{

  public AccessArchiveDatabaseProvider(string pathToDatabase) : base(↩
    pathToDatabase)
  {
  }

  /// <summary>
  /// Writes record to archive database
  /// </summary>
  /// <param name="record">the record to write</param>
  public void WriteToArchive(Record record)
  {
    WriteRecord(record, "ARCH");
  }

  /// <summary>
  /// Creates new database. If the file already exists, do nothing
  /// </summary>
  /// <param name="path">Path to the database</param>
  /// <param name="deviceCount">Number of devices initialize</param>
  /// <returns>The path to new database</returns>
  public void CreateDatabase(string path, List<Sensor> deviceCount)
  {
    if (File.Exists(path)) return;
    File.Copy(System.AppDomain.CurrentDomain.BaseDirectory + "\\Resources\\↩
    empty.mdb", path, true);

    var createSql = deviceCount.Aggregate("CREATE TABLE[ARCH] ([DATE_TIME] ↩
    TEXT(20), [DTIMNUM] FLOAT NOT NULL CONSTRAINT ArchIDKey PRIMARY KEY , [↩
    PES2] FLOAT, [PC2] FLOAT,"
      , (current, sensor) => current + ($"[{sensor.ValueColumnName}] FLOAT,↩
     [{sensor.SymptomColumnName}] FLOAT,"));
    createSql = createSql.Substring(0, createSql.Length - 1) + ", [↩
    DBDATETIME] DATETIME);";
    var provider = new AccessArchiveDatabaseProvider(path);
    provider.ExecuteNonQueryCommand(createSql);
  }

  /// <summary>
  /// Reads last record written to database
  /// </summary>
  /// <returns>The last record or null, if there was no record</returns>
  public Record FillRecordWithData(Record record, List<Sensor> sensors)
  {
    string sqLselect = "SELECT * FROM ARCH WHERE DBDATETIME=#" + record.↩
    RecordTime.ToString(CultureInfo.InvariantCulture) + "#";
    var t = FillDataWhere(sqLselect,sensors);
    if(t == null) return record;
    t.PositionOnStack = record.PositionOnStack;
    return t;
  }
```

Obrázek B.20: Ukázka z třídy zprostředkující data z archivních databází.

```
using Digiterm.Common.Utilities;

namespace Digiterm.Common.Database.Provider
{
public class AccessActualDatabaseProvider : ↩
    AccessDatabaseWithRecordsProvider, IActualDatabaseProvider, ↩
    IActualDataProvider
{
  public AccessActualDatabaseProvider(string pathToDatabase) : base(↩
    pathToDatabase)
  {
  }

  public Task<ICommunicationResult<Record>> GetActualRecordAsync(↩
    ICollection<Sensor> sensors, DateTime lastRecord)
  {
    return Task.Run(() => GetActualRecord(sensors, lastRecord));
  }

  private ICommunicationResult<Record> GetActualRecord(ICollection<Sensor> ↩
    sensors, DateTime lastRecord)
  {
    try
    {
      var queryString = GetSqlString(sensors, lastRecord, "ACT", true);

      using (var connection = GetConnection())
      {
        connection.Open();
        using (var command = new OleDbCommand(queryString, connection))
        {
          using (var reader = command.ExecuteReader())
          {
            if (!reader.HasRows) return new DatabaseCommunicationResult<↩
    Record>(new Exception("No new data"));
            reader.Read();


            var dateTime = (DateTime) reader["DBDATETIME"] ;
            var ret = new Record(dateTime) {IsActualRecord = true};
            foreach (var sensor in sensors)
            {
              var smyptom = (MeasuredSymptoms) (byte) (double) reader[↩
    sensor.SymptomColumnName];
              var value = (double) reader[sensor.ValueColumnName];
              var measurement = new Measurement(value,smyptom,sensor);
              ret.Values.Add(measurement);
            }

            if (HasColumn(reader, "DOORS"))
            {
              var doors = reader["DOORS"] as byte[];
              for (var i = 0; i < ret.Values.Count; i++)
              {
                ret.Values[i].Doors = doors != null && doors.GetBit(i);
              }
            }

            return new DatabaseCommunicationResult<Record>(ret);
          }
        }
      }
    }
    catch (Exception exception)
    {
      return new DatabaseCommunicationResult<Record>(exception);
    }
  }
```

Obrázek B.21: Ukázka z třídy zprostředkující data z centrální databáze a tabulky aktuálních hodnot.

```
namespace Digiterm.Read.Converters
{
  class LessThanMultiConvertor : IMultiValueConverter
  {
    public object Convert(object[] values, Type targetType, object ↩
    parameter, CultureInfo culture)
    {
      if (values.Length < 2) return false;

      var firstValue = values[0] as double?;
      var secondValue = values[1] as double?;


      if (!firstValue.HasValue || !secondValue.HasValue)
        return false;

      if (values.Length == 3)
      {
        var enable = values[2] as bool?;
        if (enable != null && !enable.Value)
          return false;
      }

      return firstValue < secondValue;
    }

    public object[] ConvertBack(object value, Type[] targetTypes, object ↩
    parameter, CultureInfo culture)
    {
      throw new NotImplementedException();
    }
  }
}
```

Obrázek B.22: Konvertor kontrolující, jestli hodnota je menší než definovaný parametr. Používá se pro kontrolu, zda jsou naměřené hodnoty v mezích alarmů/limitů. Jako první se předává naměřená hodnota, druhá hodnota meze a třetí volitelný parametr indikuje v případě alarmu, zda je zapnutý.

```
  public enum MeasuredSymptoms : byte
  {
    FirstRecordSymptom = 0,
    NormalValueSymptom = 1,
    NotMeasuredSymptom = 2
  }
```

Obrázek B.23: Výčet hodnot, kterých nabývá symptom měření, vit tabulka 4.1.

Příloha C

Použité programy

TexStudio	psaní bakalářské práce v LaTeX a překlad do PDF
UMLet	kreslení UML diagramů
Sublime Text	úprava textových souborů
GIT	verzování kódu
Google Drive	zálohování bakalářské práce
MDB Viewer	prohlížení a editace databází
MS Visual Studio 2015	implementace programu

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX
- `images` ..... Použité obrázky z práce
- `prilohy` ..... Přilohy práce
- `diagrams` ..... Zdrohové soubory diagramů ve formátu *UXF*