TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro konfiguraci a správu dat měřicích přístrojů v prostředí Android

Bakalářská práce

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

Autor práce: **David Eisler**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2016

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

Configuration and Data Management Application for Measuring Instruments in Android

Bachelor thesis

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

Author: **David Eisler**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**David Eisler**
Osobní číslo:	**M12000121**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro konfiguraci a správu dat měřicích přístrojů v prostředí Android**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s nástroji aplikace ENVIS, zejména s jejími funkcemi pro konfiguraci a archivaci dat podporovaných měřicích přístrojů (ENVIS.Daq).
2. Navrhněte vlastní knihovny funkcí pro načítání a archivaci dat analyzátorů kvality elektrické energie pro zvolenou mobilní platformu.
3. Navrhněte vhodně zjednodušené grafické rozhraní, optimalizované pro běžná Android zařízení a vytvořte v něm ukázkovou aplikaci.
4. Demonstrujte správnou funkci vytvořené aplikace a diskutujte další možnosti jejího rozvoje.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **KRAUS, Jan a Martin BLÍŽKOVSKÝ. KMB SYSTEMS, S.R.O. Uživatelská příručka aplikace ENVIS v. 1.2 [online]. 2015. [cit. 2015-10-08]. 1.2. Dostupné z: http://www.kmb.cz/**

[2] **GOOGLE INC. Online dokumentace Android [online]. 2015 [cit. 2015-10-08]. Dostupné z: http://developer.android.com/**

[3] **XAMARIN INC. Xamarin documentation and guides [online]. [cit. 2015-10-08]. Dostupné z: http://developer.xamarin.com/guides**

Vedoucí bakalářské práce: **Ing. Jan Kraus, Ph.D.**
Ústav mechatroniky a technické informatiky

Konzultant bakalářské práce: **Ing. Pavel Štěpán**
KMB systems, s.r.o.

Datum zadání bakalářské práce: **10. října 2015**

Termín odevzdání bakalářské práce: **16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2015

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.

Datum: 16. 5. 2016

Podpis:

Poděkování

Rád bych poděkoval zástupcům společnosti Skeleton Software s.r.o. za poskytnutí Xamarin licence pro tuto práci. Také bych chtěl poděkovat vedoucímu práce Ing. Janu Krausovi za rady a ochotu při zpracování tohoto zadání. V neposlední řadě také konzultantovi Ing. Pavlu Štěpánovi a jeho kolegům za osobní rady ohledně protokolu KMB Long a zdrojovým kódům aplikace ENVIS.

Abstrakt

Tato bakalářská práce popisuje postup vývoje knihoven pro komunikaci s měřiči elektrické energie firmy KMB systems s. r. o. přes Ethernet a zpracování takto získaných konfiguračních dat. Práce zahrnuje také navržení uživatelského grafického rozhraní a tvorbu aplikace pro Android, která demonstruje funkci těchto knihoven. Vývoj probíhal v jazyce C# za pomoci Xamarinu. Díky inspiraci v dependency injection je možné knihovny použít i na jiných mobilních platformách.

Klíčová slova

vývoj aplikací, návrh uživatelského grafického rozhraní, C#, Xamarin, Android

Abstract

This bachelor thesis describes the development of libraries for Ethernet communication with measuring instruments of company KMB systems s. r. o. and processing acquired configuration data. The thesis also includes design of graphic user interface and development of Android application that demonstrates the functionality of these libraries. Development was conducted in C# with Xamarin. These libraries can be used on other mobile platforms thanks to the inspiration in dependency injection.

Keywords

application development, graphic user interface design, C#, Xamarin, Android

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

XML	Extensible Markup Language
CSV	Comma-separated values
TCP	Transmission Control Protocol
GUI	Graphic User Interface
API	Application Programming Interface
AXML	Android XML
LINQ	Language Integrated Query
IDE	Integrated Development Environment
IL	Intermediate Language – vrstva mezi C# a nativní assembly [17]
JNI	Java Native Interface
SDK	Software development kit
UWP	Universal Windows Platform
MVC	Model-view-controller
IoC	Inversion of Control
DI	Dependency injection
DLL	Dynamic-link library
TCP	Transmission Control Protocol
IP	Internet Protocol
CRC	Cyclic redundancy check
OS	Operační systém
HW	Hardware
SW	Software
UI	User Interface
AP	Access point

1 Úvod

Důvodem vzniku této aplikace je skutečnost, že v dnešní době u sebe téměř každý uživatel nosí nějaký druh mobilního zařízení (chytrý telefon, tablet apod.). I když aplikace ENVIS pro Windows a Linux již existuje, ne každý má u sebe svůj počítač pokaždé, když potřebuje na měřiči něco zkontrolovat nebo upravit. V rámci mobilních zařízení existuje mnoho různých platforem. Tato práce řeší i přenositelnost již napsaného kódu (business logiky) na tři platformy, které jsou nejsilnější na trhu (Android, Windows Phone a iOS).

Pro funkci i grafické rozhraní této práce byla předlohou aplikace ENVIS.Daq pro konfiguraci a vizualizaci dat měřicích přístrojů v OS Windows. Aplikace ENVIS.Daq slouží k nastavování přístrojů, stahování dat a zobrazení aktuálního stavu. Pomocí ENVIS.Daq mohou být data stahována do binárních souborů, exportována do xml/csv a nebo ukládána přímo do databáze [5].

Hlavním cílem této práce bylo vytvoření knihoven, nad kterými bude napsána aplikace pro Android, jež bude sloužit k nastavování měřicích přístrojů kvality elektrické energie ARTIQ nebo SMC 144. Aplikace se tedy musí umět připojovat k zařízením přes TCP protokol, má být schopna stahovat jeho konfigurační data, umožnit uživateli jejich editaci a odesílat je zpět k měřicímu přístroji. Komunikace by měla probíhat pomocí protokolu KMBLong. Aplikace by měla být schopna ukládat a načítat konfiguraci z binárních konfiguračních souborů.

Dalším úkolem této práce je navržení grafického uživatelského rozhraní. Měl by se klást důraz na intuitivnost ovládání aplikace. Tudíž by se měla svým uspořádáním a rozdělením jednotlivých obrazovek podobat aplikaci ENVIS.Daq. Aplikace také musí zapadat do prostředí Androidu. GUI by mělo být definováno relativně, aby bylo možné aplikaci pohodlně používat na zařízeních s různými typy displejů.

Toto téma jsem si vybral z toho důvodu, protože bych se i v budoucnu chtěl profesně zabývat vývojem mobilních aplikací. Chtěl bych porozumět problematice multi-platformního vývoje pomocí Xamarinu a Dependency injection a zdokonalovat se v něm. V neposlední řadě bych byl rád, kdyby se výsledná aplikace opravdu používala v praxi a usnadňovala tím práci jejím uživatelům.

2 Podrobnější představení problematiky

Snažil jsem se nalézt nějaké články či publikace, které by se zabývaly podobným tématem. Existují aplikace, ve kterých se využívá komunikace s měřiči pomocí protokolu Modbus nebo M-bus. Nejpodobnější řešení jsem našel u svého kolegy, který řešil podobné zadání v rámci svého bakalářského projektu. On komunikaci řešil pomocí protokolu M-bus. Zařízení firmy KMB systems s.r.o. umí komunikovat pomocí protokolu Modbus, konkrétně jeho verze Modbus RTU. Tento protokol je ovšem pro svá omezení nevhodný pro tuto aplikaci, protože je délka zprávy omezená na 260 bytů a slouží tak tedy spíše pro vyčítání jednoduchých dat z měřicího přístroje [3]. Pro robustní a komplikovanou komunikaci je nepoužitelný. Proto se v aplikaci pro vyčítání konfigurace používá protokol KMB Long.

Původní aplikace ENVIS.Daq pro Windows je napsaná v jazyce C#. Abych mohl využít již napsaný kód, rozhodl jsem se napsat knihovní funkce pro načítání a archivaci dat také v jazyce C#. K tomu jsem se rozhodl využít Xamarin Framework.

2.1 Android

Jako minimální podporovanou verzi jsem zvolil API 16, které je implementované v Android 4.1 Jelly Bean. Maximální podporovaná verze je API 23, což odpovídá Androidu 6.0 Marshmallow. Toto rozmezí nyní zahrnuje 95 % všech zařízení registrovaných v Google Play Store [4], viz Obrázek 1.

Obrázek 1: Na prvním grafu je vyobrazeno zastoupení verzí Androidu v Google Play Store. Na druhém grafu je znázorněno zastoupení velikostí displejů v Google Play Store [4].

Grafické uživatelské rozhraní by mělo být optimalizované pro běžná Android zařízení. Z Obrázek 1 je patrné, že valná většina displejů je velikosti „Normal“, což odpovídá reálné velikosti displeje přibližně od 3 do 4,5 palce. Je tedy vhodné optilmalizovat grafické uživatelské rozhraní právě pro tento typ displeje. Díky relativní definici GUI

pomocí AXML souborů by se měly komponenty u větších displejů roztáhnout a u menších smrštit, případně zalomit.

Aplikace pro OS Android se většinou skládají z tříd typu Activity [1], případně dalších rozšiřujících tříd. Tyto třídy většinou reprezentují jen jednu stránku obsahu aplikace a zprostředkovávají interakci mezi view (uživatelem), telefonem (přístup k systémovým zdrojům) a business logikou. Z pohledu návrhového vzoru MVC je to tedy controller. Funkci některých Activity částečně přebírají Fragmenty [6].

2.2 Xamarin

Klíčovým nástrojem pro mne byl Xamarin Framework [8], který umožňuje psát aplikace pro mobilní zařízení s operačním systémem iOS nebo Android v jazyce C# a přitom zachovává veškerou funkcionalitu nativních jazyků (Objective-C a Java).

Platforma Xamarin se skládá z několika prvků, které umožňují vyvíjet aplikace pro operační systémy Android a iOS [17].

- C# – Umožňuje použití jeho syntaxe a sofistikovaných funkcí, jako je např. genericity, syntaxe LINQ, asynchronní obsluha pomocí Task apod.
- Mono .NET Framework – Poskytuje multiplatformní implementaci funkcí v rámci .NET od společnosti Microsoft.
- Compiler – U Androidu integruje do standardního aplikačního balíčku napsanou .NET aplikaci a běhový modul. Překladač také provádí mnoho optimalizací pro mobilní nasazení.
- IDE tools – Instalace zahrnuje Xamadin Studio IDE a plug-in pro Visual Studio.

Při kompilaci je kód napsaný v C# zkompilován do IL a následně je vytvořen balíček pomocí MonoVM + JIT'ing. Nevyužité třídy ve frameworku jsou odstraněny během linkování. Do nativního assembly je balíček zkompilován až při spuštění. Aplikace běží bok po boku s Java/Dalvik a spolupracuje s nativními typy přes JNI.

Xamarin.Android implementuje Google Android SDK, takže se můžu odkazovat na jakoukoli část podporovaných SDK, jako je například použití Android.Views (ovládací prvky uživatelského rozhraní) apod.

Protože je využíván jazyk C# s implementací .NET frameworku, projekty mohou být strukturovány tak, aby bylo možné sdílet kód skrze všechny tři mobilní platformy a případně i Windows a UWP. Ovšem konkrétní implementace různých specifických funkcí, jako jsou např. práce se soubory a komunikace přes síť, je na každém systému velmi odlišná.

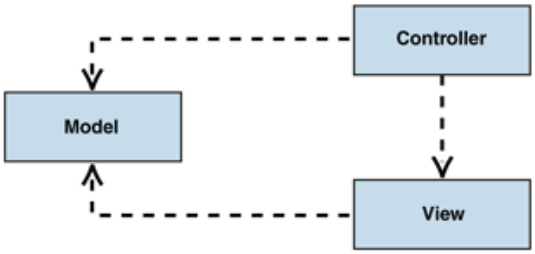
Xamarin mi tedy zajišťuje stoprocentní přístup k nativnímu API a zároveň sdílení aplikační logiky skrze platformy. To při vývoji mobilních aplikací šetří mnoho času a peněz, ale také velmi zjednodušuje jejich údržbu a případné rozšiřování. Umožňuje mi psát knihovny, které lze sdílet napříč aplikacemi i platformami.

2.3 Použité návrhové vzory a principy návrhu

Při vývoji knihoven a aplikace jsem pro její dobrou využitelnost a rozšiřitelnost použil řadu návrhových vzorů a principů, které jsem plně implementoval, nebo jsem z nich vycházel.

2.3.1 Model-View-Controller

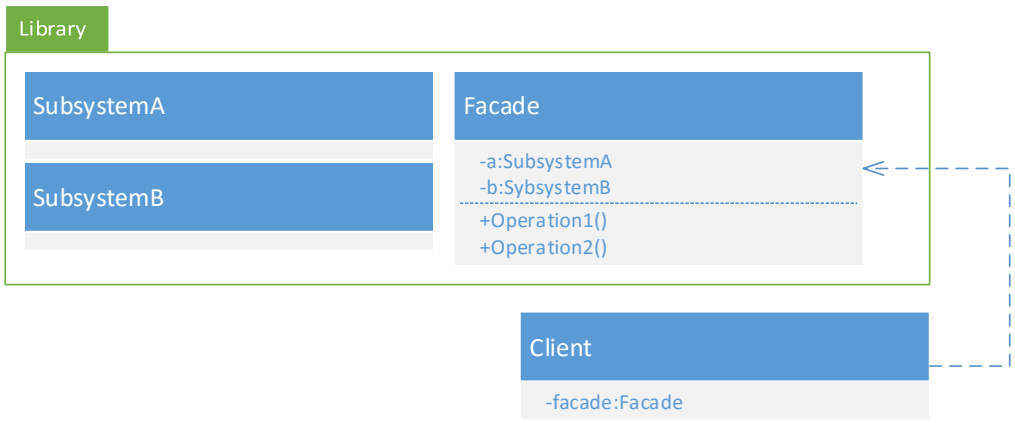
Model-view-controller (MVC) je softwarová architektura (návrhový vzor), která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má jen minimální vliv na ostatní. Model jsou tedy data a business logika. Může zprostředkovávat připojení k databázi, komunikaci s webovými službami, ale také zpracování těchto dat (výpočty apod.). View je surová definice zobrazení dat. Určuje, jak budou data uživateli prezentována. Tato komponenta obsahuje layouty, obrázky a podobně. Součástí view může být i kód, který ošetřuje uživatelské vstupy (kliknutí, psaní na klávesnici, aj.) Controller je ústřední jednotkou, která se stará o celkové provázání aplikace. Tedy spojuje view a model. Controller zpracovává akce uživatele a o těchto akcí informuje model a/nebo view.



Obrázek 2: Koncepční schéma model-view-controller [13]

2.3.2 Facade pattern (Business facade)

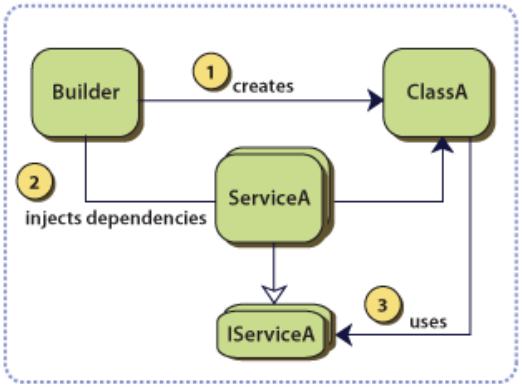
Jedná se o návrhový vzor, který poskytuje konzistentní rozhraní pro business logiku. Role vzoru Facade je poskytovat různé pohledy do subsystému skrytého před uživatelem. Obecně platí, že Facade obsahuje různé výběry operací, které by moly být žádoucí z pohledu uživatele. Komponenta Facade je většinou implementována formou managerů (občas je tento vzor nazýván jako Manager pattern). [14]



Obrázek 3: Koncepční schéma Facade pattern

2.3.3 Dependency injection

Vkládání závislostí (dependency injection) je implementační technika principu IoC (Inversion of control), kde framework volá kód závislý na úloze [12]. Technika DI slouží pro vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době sestavování programu referenci [11].



Obrázek 4: Koncepční pohled návrhový na vzor Dependency Injection [11]

2.4 Komunikační protokol KMB Long

Analyzátory elektrické energie umí komunikovat pomocí USB přes emulovanou sériovou linku pomocí protokolu KMB Long. Případně může být každé zařízení vybaveno RS232, nebo RS485. Zde může komunikace probíhat jako pomocí protokolu KMB Long, tak i Modbus RTU. Nejpokročilejší možností komunikace je s využitím Ethernetu. Tato implementace umožňuje se zařízením komunikaci pomocí Modbus RTU a KMB Long protokolu, nebo přes vestavěný web server s aktuálními daty a konfigurací [16].

Komunikační kanál používá osmibitovou zprávu pro nastavení. Zpráva protokolu neobsahuje žádnou paritu a končí jedním stop bitem. Nastavit lze adresu a rychlost. Protokol využívá filozofii master-slave. V reakci na příjem zprávy nebo příkazu od master zařízení pošle zpět příslušnou odpověď. Všechny podporované zprávy mají jednotný formát (frame) [16]:

1. Adresa přístroje (1 byte), hodnoty 0 a 255 jsou rezervované
2. Délka zprávy (2 byty)
3. Typ zprávy (1 byte)
4. Tělo zprávy – data v závislosti na typu zprávy
5. 16bitový CRC

2.4.1 Kódování hodnot

1. Datum a čas jsou kódované jako 64bitové hodnoty reprezentující počet milisekund od 00:00:00 1.1.2000.
2. MTN, MTNN, MTP a MTPN jsou multiplikátory definující aktuální poměr transformátoru. Poměry xxxN se používají pro čtvrté fáze/kanály.
    a. Pro MTN a MTNN (napětí) hodnota 0xFFFF znamená přímé měření, jinak to je hodnota pro poměr xxx/100.
    b. Pro MTP a MTPN (proud) – poměr xxx/1A nebo xxx/5A je definovaný na nejvyšším bitu (MSB). V případě, že je nastaven, tak poměr je (hodnota&0x7FFF)/5(A) jinak je poměr hodnota/1(A).
3. Specifická interpretace hodnot nebo bitového mapování (bit mapping) jsou popsány ve struktuře zprávy.
4. Naměřená data z přístroje jsou optimálně kódována podle následující tabulky (Tabulka 1).

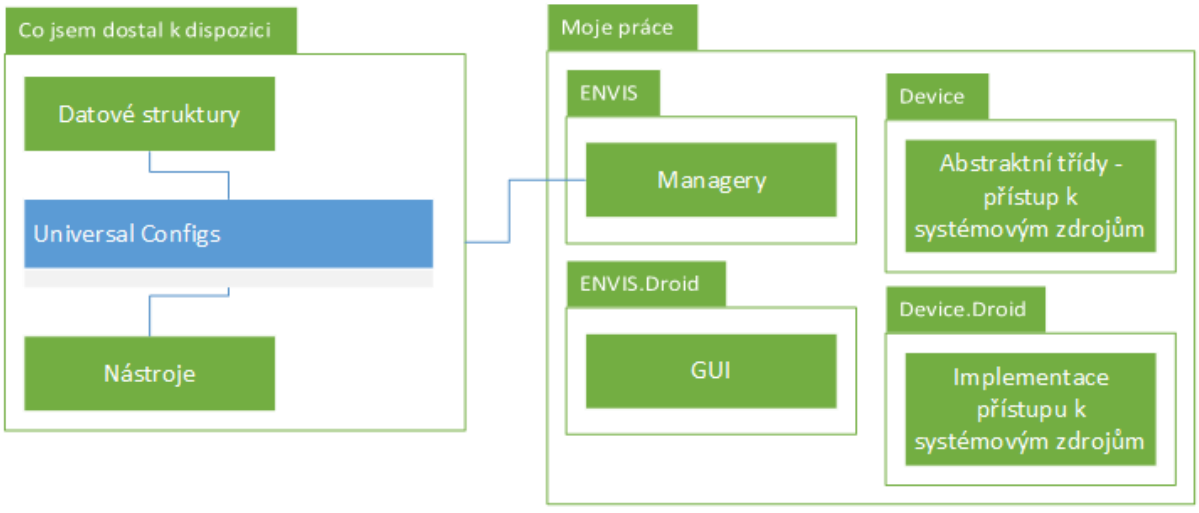
Tabulka 1: Kódování naměřených dat v protokolu KMB Long [16]

Quantity	Coding			Invalid Value
U_LN	u16	$*\{MTN\|MTNN\}/40$	V	
U_LL	u16	$*\{MTN\}/40$	V	
I	u16	$*\{MTP\|MTPN\}/5000$	A	
I_{calc}	u16	$*MTP/5000$	A	
Powers	float	$*\{MTP\|TPN\} * \{MTN\|MTNN\}$	W, Var, VA	
Voltage Unbalance	u16	$/100$		$0xFFFF$
Voltage THD	u16	$/100$	%	$0xFFFF$
Current THD	u16	$/100$	%	$0xFFFF$
Voltage Harmonics	u16	$*MTN/40$	V	$0xFFFF$
Current Harmonics	u16	$*MTP/5000$	A	$0xFFFF$
Harmonic Angle	s16	$/4096$	rad	$0x7FFF$
Frequency	u16	$/100$	Hz	$0xFFFF$
Flicker	u16	$/100$		$0xFFFF$
Energies	u32	$*MTN * MTP$	Wh, Varh	

3 Návrh řešení

K dispozici jsem dostal zkompilované knihovny (DLL), které byly napsané pro aplikaci pro Windows. Tyto knihovny jsem chtěl přímo využít také v aplikaci pro Android. Abych zjistil, jestli jsou knihovny kompatibilní s platformou Android, tak jsem použil webovou aplikaci od Xamarinu [7], která zkontroluje, jaké prostředky knihovny používají. Tato webová aplikace bohužel ukázala, že v knihovnách jsou použité reference na System.Windows.Forms, Systém.Drawing a podobně, tudíž použití již zkompilovaných knihoven pro mne v tomto případě nebylo možné.

Abych v budoucnu předešel takovýmto situacím, tak jsem se snažil oddělovat aplikační logiku od GUI hned od začátku návrhu. Snažil jsem se tedy využívat návrhový vzor MVC. Android navíc přímo vede k tomu, aby bylo view striktně odděleno od zbytku aplikace. Celé view je definované v Resources pomocí surových dat, jako jsou obrázky, xml soubory a podobně. Veškerá funkcionalita by tedy měla být implementována právě v knihovnách, aby zůstala oddělená. View a model jsou propojeny controllerem, který je v případě Androidu reprezentován třídami typu Activity nebo Fragment. K tomu, abych dokázal správně navrhnout strukturu aplikace a docílil přitom co nejčistšího kódu, mi hodně napomohla publikace [10] McConnell Steve. *Dokonalý kód: umění programování a techniky tvorby software. Brno: Computer Press, a.s., 2006. ISBN 80-251-0849-X.*.



Obrázek 5: Zjednodušený diagram aplikace. Znázorňuje, jaké třídy jsem dostal k dispozici a jaké knihovny jsou součástí mojí práce.

Kvůli tomu, že jsem nemohl použít zkompilované knihovny, bylo potřeba z kódu původní aplikace vybrat třídy, které by se daly v aplikaci pro Android použít (Obrázek 5), a některé tyto třídy upravit nebo vytřídit to, co je pro aplikaci užitečné a to co se

naopak může smazat. Řada funkcí se musela napsat znova s ohledem na konkrétní platformu – tedy Android. Bylo vytříděno 12 souborů s jednotlivými třídami (u některých byla zachována jen hlavička pro budoucí použití), jež bylo třeba projít a upravit tak, aby plnily svou funkci. K tomu bylo potřeba osobně se sejít s vývojářem, který je psal.

Výchozím bodem byla třída UniversalConfigs, která reprezentuje univerzální strukturu pro uchovávání konfiguračních dat měřicího zařízení a nástroje pro práci s nimi. Z ní se využily funkce pro serializaci a deserializaci konfiguračních dat a popisů struktury přijatých ze zařízení. Pro tyto funkce bylo potřeba použít další třídy z původní aplikace pro Windows. Jako například třídy reprezentující zprávu v protokolu KMBLong, třídu pro práci s bytovými poli, struktury pro uchování dat apod. Já jsem musel implementovat třídu pro komunikaci se zařízením přes TCP/IP a třídy pro dekódování hodnot z univerzální struktury dat. Aplikace je rozdělena na tři knihovny (Knihovna ENVIS, Knihovna Device a Knihovna Device.Droid) a samotnou aplikaci (Aplikace ENVIS a realizace GUI).

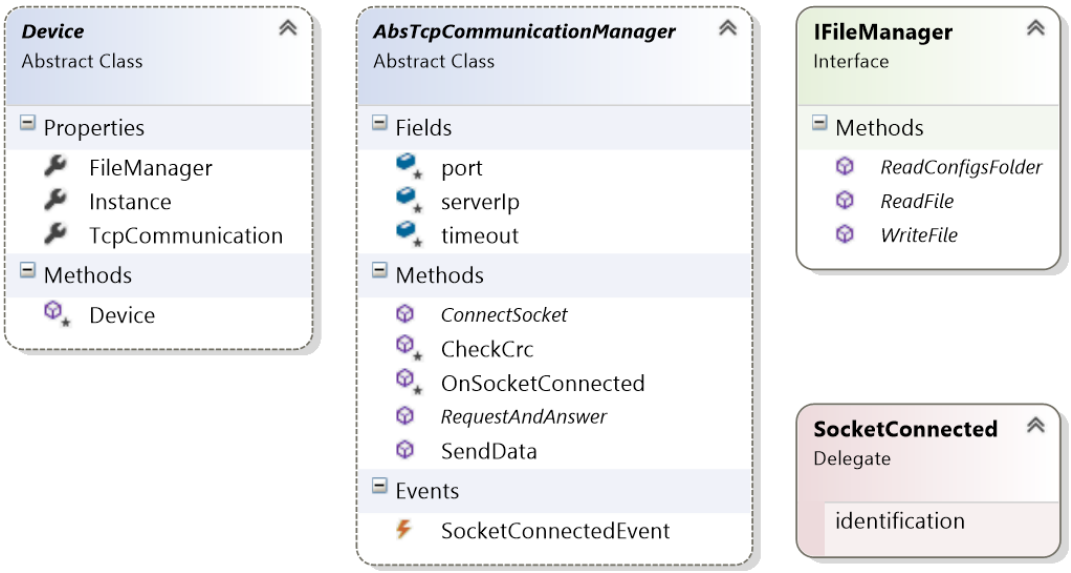
3.1 Knihovna Device

Portable knihovna Device slouží k definici přístupu k systémovým zdrojům. Jejím hlavním prvkem je abstraktní třída Device. Tato třída drží informaci o tom, jaké platformě závislé funkce definuje a drží instanci konkrétní implementace Device (např. Device.Droid) pro použití v portable knihovně ENVIS. Vycházím zde tedy z techniky zvané vkládání závislostí (dependency injection – viz 2.3.3). Knihovna ENVIS s funkcemi normálně pracuje, aniž by bylo jasné, na jaké platformě poběží. Konkrétní implementace je do této instance vložena až při inicializaci za běhu programu.

Dále obsahuje abstraktní třídu pro komunikaci pomocí socketu přes TCP/IP protokol. Tato třída je abstraktní a definuje metody pro připojení socketu a pro komunikaci (RequestAndAnswer – odeslání příkazu a přijetí odpovědi). Dále má implementované metody pro vyvolání události SocketConnectedEvent, pro kontrolu CRC a pro odeslání dat (viz Obrázek 6).

Definice metody ConnectSocket má tři vstupní parametry. Pole bytů reprezentující zprávu s příkazem o zaslání identifikace měřicího přístroje, adresu (IP, nebo doména) jako řetězec a číslo portu. Hlavička metody RequestAndAnswer má jako vstupní parametr pole bytů reprezentující odchozí příkaz a její návratová hodnota je asynchronní úloha, jejímž výsledkem je taktéž pole bytů, které reprezentuje příchozí

zprávu (odpověď slave zařízení). Metoda pro odeslání dat SendData pouze využije metodu RequestAndAnswer, kde jako vstupní data nastaví příkaz s konfigurací a zkontroluje, zda přišla validní odpověď.



Obrázek 6: Struktura knihovny Device – definice platformě závislých funkcí.

V knihovně Device je také implementovaný interface IFileManager, který definuje metody pro práci s binárními soubory, A sice hlavičku metody pro načtení seznamu souborů ve složce, definici metody pro zápis dat do binárního souboru a metodu pro čtení dat z binárního souboru. Metoda pro načtení seznamu souborů (ReadConfigsFolder) má jediný parametr s názvem adresáře, stejně jako metoda pro načtení souboru (ReadFile), jež má také jediný parametr s názvem souboru. Metoda pro uložení dat (WriteFile) má jeden parametr s názvem souboru a druhý parametr pro bytové pole, který má proměnný počet argumentů (klíčové slovo params). V případě, že je předáno do metody více bytových polí, tak jsou do souboru uložena bezprostředně za sebou.

3.2 Knihovna ENVIS

Portable knihovna ENVIS implementuje veškerou business logiku aplikace. Ke své funkci využívá vybrané třídy a fragmenty kódu z původní aplikace ENVIS.Daq pro OS Windows (viz Obrázek 5). Všechny její třídy (DeviceManager, DataManager a ConfigurationManagery) vždy obsluhují určitou část funkčnosti, u které je žádoucí, aby byla vystavena ven z knihovny.

Třída DataManager slouží k práci s daty. Obsahuje tedy metody LoadData obstarávající stažení konfiguračních dat do univerzální struktury (refresh dat),

metodu LoadDataInfo, která zařídí stažení dat potřebných k vytvoření univerzální struktury a následně ji pomocí metody LoadData naplní a metodu SendData pro odeslání konfiguračních dat uložených v konfigurační struktuře. Třída také obsahuje metody pro načítání a ukládání konfiguračních dat z univerzální striktury do binárních souborů.

Stažení parametrů k vytvoření univerzální konfigurační struktury probíhá ve dvou částech. Každá z částí má na svém začátku na dvou bytech uloženou svoji celkovou délku. Nejprve je potřeba získat data o kategoriích konfigurace (viz Tabulka 2). Každá kategorie má své ID, sekundární ID a název v podobě řetězce.

Tabulka 2: Popis struktury konfigurace – jednotlivé položky reprezentují jednu kategorii.

Popis struktury konfigurace														
délka bloku	ID	IDs	n	a	m	e	\0	…	ID	IDs	n	a	m	e	\0

Ve druhé zprávě se stahuje struktura jednotlivých hodnot v konfiguračních kategoriích (Tabulka 3). Každá položka má své ID, ID kategorie, informaci o délce hodnoty v bytech a mód. Po vytvoření konfigurační struktury se mohou stáhnout data. Ta přijdou už jen jako stream bytů, který je zpracován podle informací uložených ve struktuře.

Tabulka 3: Popis struktury dat – jednotlivé položky nesou parametry konkrétní hodnoty.

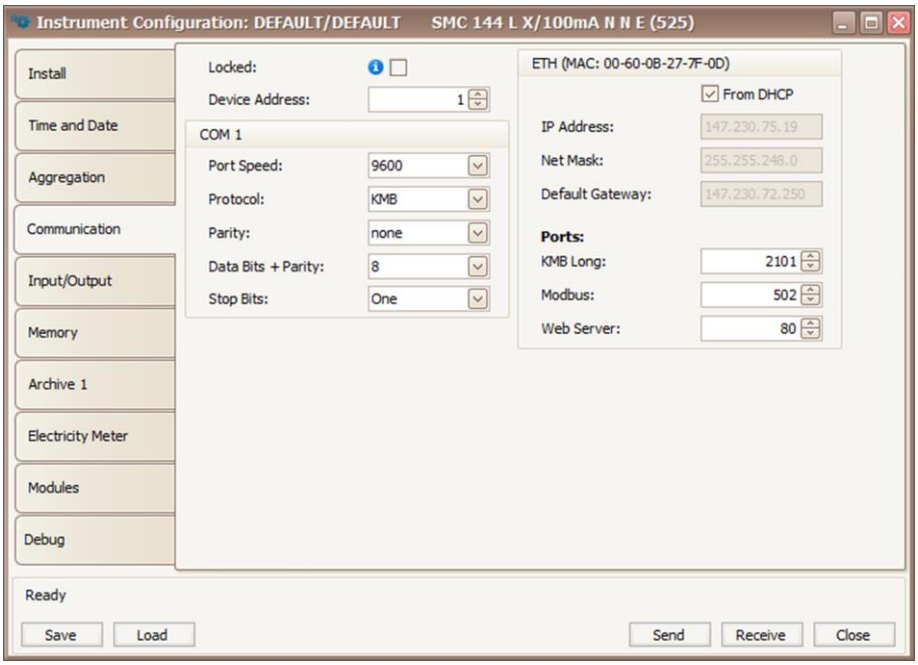
Popis struktury dat								
délka bloku	ID	DT	délka hodnoty	mode	…	ID	DT	délka hodnoty	mode

Třída DeviceManager pouze pro jednodušší práci zprostředkovává instanci třídy TcpCommunication z knihovny Device a drží identifikační informace o měřiči, ke kterému je aplikace aktuálně připojena. Tyto informace přesně definují typ měřicího zařízení.

Tato knihovna dále obsahuje managery pro zpracování dat z univerzální struktury. Všechny tyto třídy dědí od abstraktní třídy ConfigurationManager, jež pouze definuje metodu LoadData, která je bez vstupních parametrů a vrací pouze bool informaci o tom, zda se data podařilo načíst. O konkrétní implementaci se píše v kapitole níže – 4.1 ConfigurationManagery.

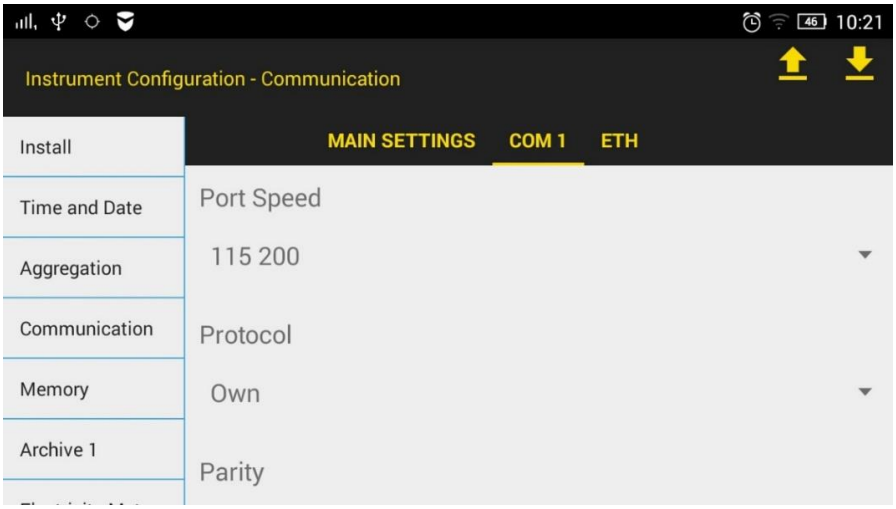
3.3 Návrh GUI

Grafické rozhraní aplikace je orientováno na šířku a je členěno do pěti částí. Podobně jako vzorová aplikace pro OS Windows (viz Obrázek 7 a Obrázek 8). Snahou bylo sloučit layout navržený pro Windows se zvyklostmi užívanými v prostředí Androidu.



Obrázek 7: ENVIS.Daq pro Windows – konfigurace komunikace.

Menu aplikace jsem nechal na levé straně obrazovky. Menu slouží pro navigaci mezi jednotlivými agendami nastavení měřicího přístroje. V horní liště (tzv. app bar nebo ActionBar) se zleva zobrazuje název stránky a zprava akční tlačítka pro načtení a odeslání konfiguračních dat. V aplikaci pro Windows jsou tato tlačítka ve spodní liště.



Obrázek 8: Aplikace pro Android – konfigurace komunikace. Ostatní screenshody v Příloha 3 – Screenshoty

Zbytek obrazovky je určen obsahu právě zvolené agendy. Některé stránky s nastavením mají mnoho konfiguračních hodnot, které spolu různě souvisí. V aplikaci pro Windows je toto rozdělení realizováno pomocí různých boxů (rámečků), které související hodnoty drží pohromadě. V aplikaci pro Android jsem toto rozdělení realizoval pomocí záložek, jejichž seznam je horizontálně zobrazen v liště ihned pod aplikační lištou. Mezi záložkami lze přepínat kliknutím na její název v liště nebo přetažením (swipe) obsahu záložky do strany. K této implementaci jsem využil volně dostupného balíčku s komponentou Refactored.PagerSlidingTabStrip [9].

Největší plochu na obrazovce zabírá obsahová část. Na této ploše se zobrazují samotná nastavení. Ve stránce jsou pak za sebou naskládané jednotlivé položky. Pokud je počet položek takový, že obsah přesáhne dolní okraj obrazovky, tak s nimi lze posouvat ve vertikálním směru (scroll). Jedna konfigurační položka obsahuje vždy svůj název a hodnotu/nastavení. Hodnota je v aplikaci zpravidla zobrazována třemi způsoby:

- jako editovatelný text (komponenta EditText) – adresy, názvy a podobně
- nabídka několika hodnot, kde lze zvolit jednu (komponenty Spinner, nebo RadioButton) – rychlosti, frekvence, podporované konfigurace
- nabídka hodnot, z nich lze zaškrtnout více voleb (CheckBox komponenty) – výběr prvků, které se mají archivovat

4 Realizace řešení

Implementace business logiky (načítání a zpracování konfiguračních dat) je realizována v knihovně ENVIS v třídách nazvaných Manager. Tyto třídy reprezentují komponentu Facade popsanou ve Facade patternu (viz 2.3.2) a zapouzdřují tak celou knihovnu a tím pádem i business logiku. V aplikaci (ENVIS.Droid) je pak už jen realizován návrh GUI a využití knihovních funkcí pro konfiguraci měřicích zařízení. Ve třídách typu Activity jsou pak vytvářeny instance managerů a skrze ně jsou data načítána do view.

4.1 ConfigurationManagery

Klíčovou roli v aplikaci ENVIS hrají managery, které rozšiřují abstraktní třídu ConfigurationManager (Obrázek 9). Každý z těchto managerů obsluhuje určitou kategorii konfigurace a implementuje metodu LoadData. V této metodě dojde k načtení a zpracování hodnot kategorie z univerzální konfigurační struktury. Hodnoty jsou poté uloženy do veřejných vlastností třídy. Všechny vlastnosti jsou udělané jako nullable hodnoty (int?, bool? apod.) a to z toho důvodu, že některá nastavení nejsou pro určité typy měřičů dostupná (softwarově nebo hardwarově ji nepodporují). V případě, že se tak stane, manager nastaví příslušnou hodnotu na null a view na tuto skutečnost může zareagovat například tím, že příslušné položky skryje nebo aspoň zneaktivní.

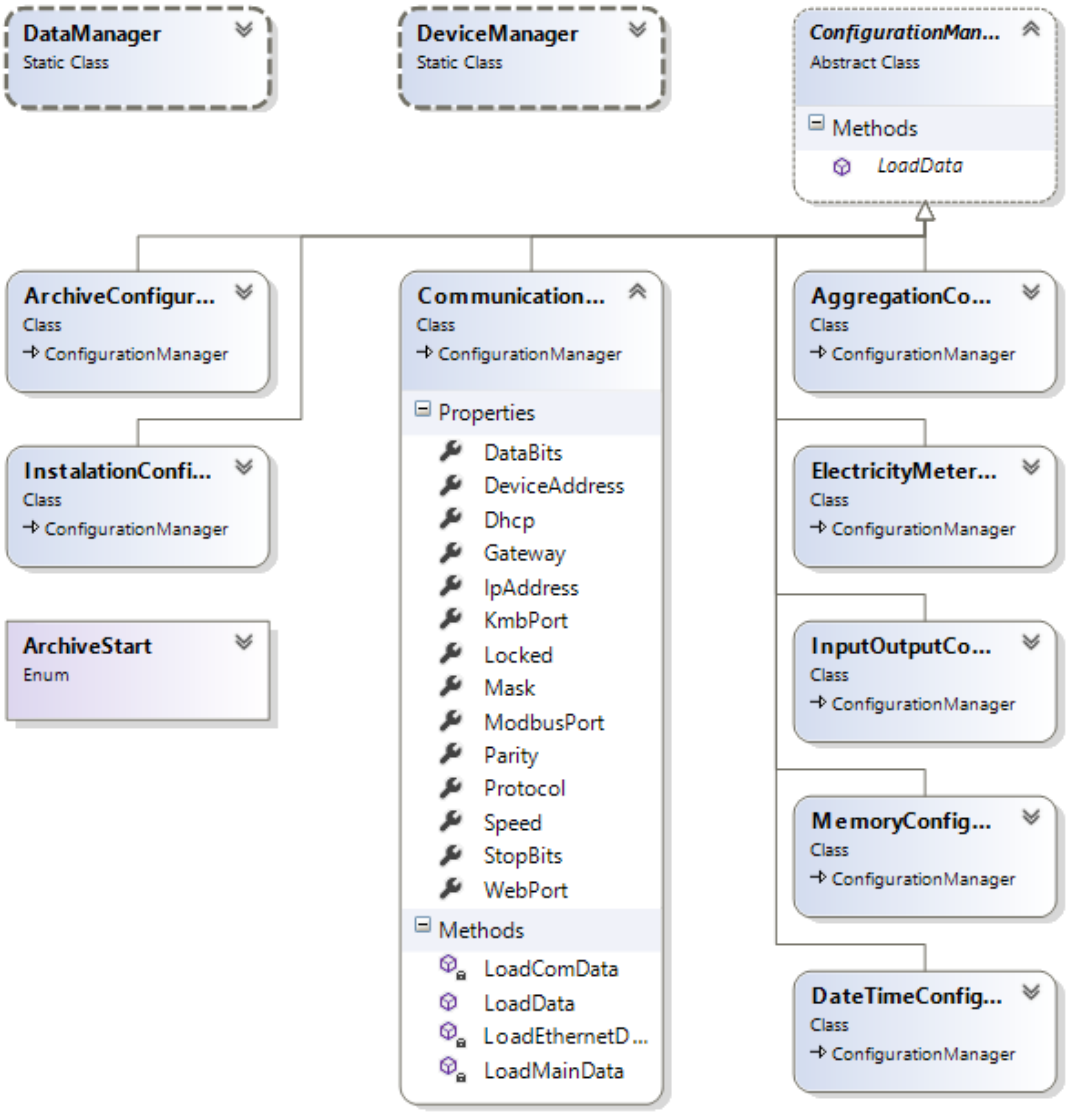
```
var ic = uniC.GetConfIC("Archive");
var rm = uniC.GetValue<byte>(ic, "Mode" + ids, 0);
DateTime startT = KMBTime.milisecondsToTime(
       uniC.GetValue<UInt64>(ic, "StartTime" + ids, 0));

bool immediatelly = (rm & 0x02) == 0x02;
bool startDi = (rm & 0x04) == 0x04;
bool startTc = (rm & 0x10) == 0x10;

if (immediatelly) this.ArchiveStart = Managers.ArchiveStart.Imediatelly;
else if (startDi) this.ArchiveStart = Managers.ArchiveStart.StartDi;
else if (startTc) this.ArchiveStart = Managers.ArchiveStart.StartTc;
else this.ArchiveStart = Managers.ArchiveStart.Never;
this.ArchiveStartTime = startT;
```

Zdrojový kód 1: Zpracování dvojice hodnot Mode a StartTime a jejich poskytnutí ven z knihovny pomocí veřejných vlastností ArchiveStart a ArchiveStartTime

Ve Zdrojový kód 1 jsem vybral přiklad zpracování dvou konfiguračních hodnot v ArchiveConfigurationManageru. Proměnná `uniC` je lokální referencí na univerzální konfigurační strukturu. Zde konkrétně jde o hodnotu ArchiveStart, která říká, kdy má započít archivování měřených dat (nikdy, ihned po odeslání změny, řízené HW nebo SW) a ArchiveStartTime, která určuje čas začátku archivace v případě, že je zdroj začátku nastaven na StartTc.

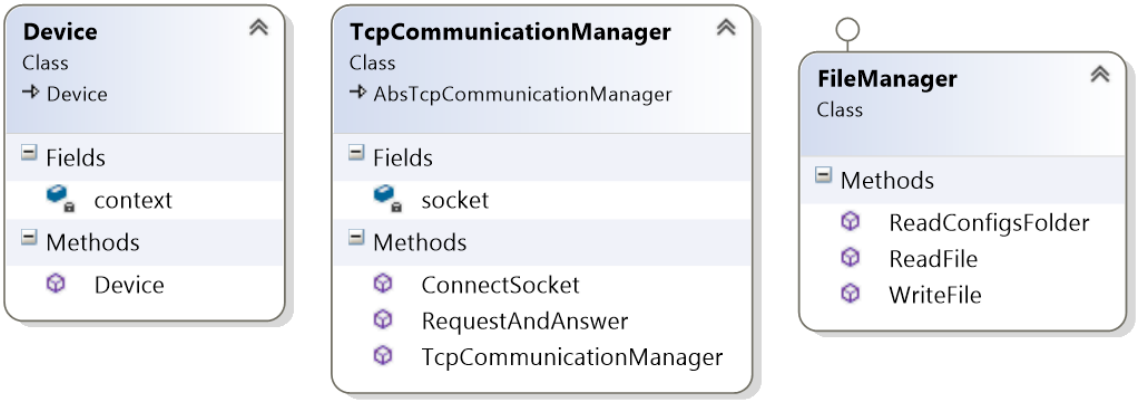


Obrázek 9: Struktura knihovny ENVIS

4.2 Knihovna Device.Droid

Knihovna Device.Droid je určená pro implementaci platformě závislých funkcí definovaných abstraktními třídami a rozhraními v knihovně Device. V případě této aplikace bylo potřeba implementovat třídu FileManager, TcpCommunicationManager a také třídu Device (Obrázek 10).

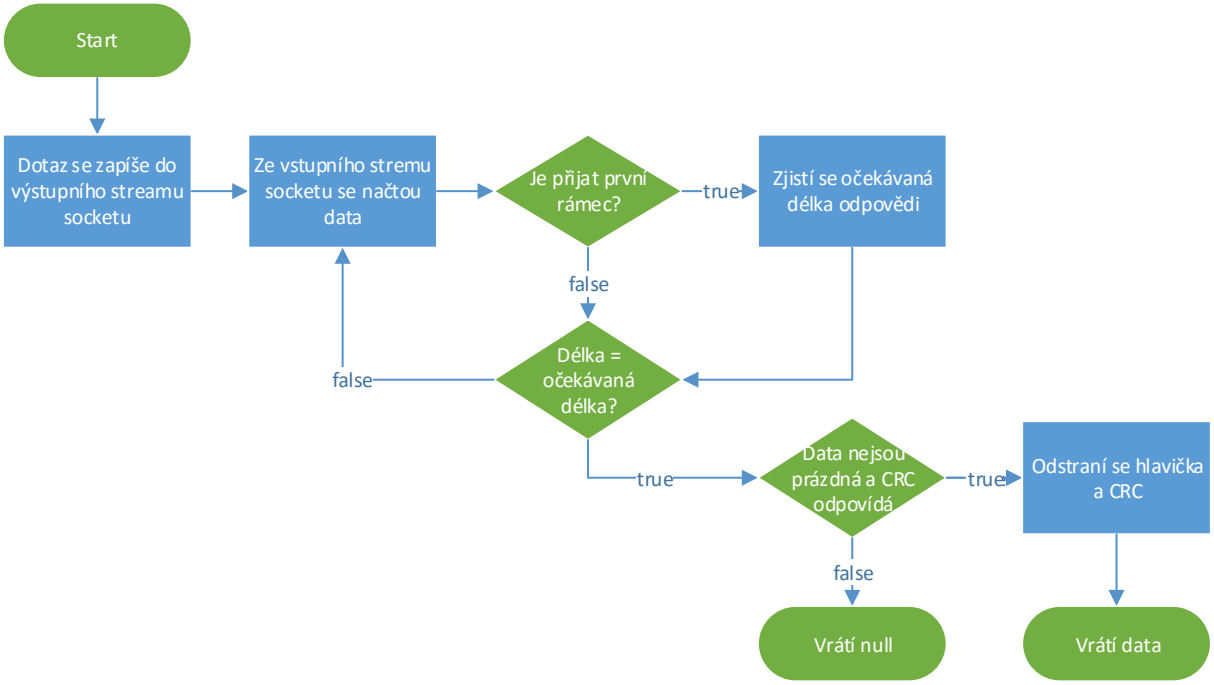
Třída FileManager implementuje interface IFileManager z knihovny Device. Všechny její metody probíhají v nových úlohách (Task), tudíž nedochází k blokaci hlavního vlákna. Metoda ReadConfigsFolder načte seznam všech souborů uložených v adresáři, který je určen string parametrem. Samozřejmě zde dochází také ke kontrole, jestli daný adresář vůbec existuje, a v případě, že tomu tak není, tak jej vytvoří. Metoda ReadFile zkontroluje existenci soboru, jehož název je předán parametrem a poté ho načte jako bytové pole a vrátí jako návratovou hodnotu. Metoda WriteFile zapíše data do souboru. V případě, že soubor se stejným názvem již existuje, je přepsán. Poté se do prázdného souboru zapíšou všechny bytové pole předané v násobném byte[] parametru (params byte[][]). Data jsou do souboru zapsána bezprostředně za sebou v pořadí od indexu 0 do konce.



Obrázek 10: Struktura knihovny Device.Droid – Implementované abstraktní třídy a interface z knihovny Device

Třída TcpCommunicationManager rozšiřuje a implementuje abstraktní metody abstraktní třídy Abs TcpCommunicationManager z knihovny Device. V konstruktoru se nastavuje timeout pro komunikaci.

Metoda ConnectSocket spustí v novém Tasku připojení socketu a odešle zařízení žádost o identifikaci, která je po přijetí předána skrze event SocketConnected k Activity. Ta ji uloží do statické vlastnosti CIdent třídy DeviceManager. Zde se mi bohužel zatím nepodařilo stoprocentně vyřešit problém, jenž nastane při pokusu o připojení socketu k zařízení, které není v síti. Aplikace se v tomto bodě zasekne (jen příslušné vlákno, UI vlákno běží dál) a nepomohlo ani nastavení timeoutu pro tento task.



Obrázek 11: Vývojový diagram metody RequestAndAnswer

Metoda RequestAndAnswer spouští svůj obsah taktéž v novém Tasku, aby nedocházelo při komunikaci k blokování UI vlákna. V metodě se nejdříve odešle žádost o data, která je předaná jako parametr v podobě pole bytů. Poté jsou v cyklu přijímány všechny rámce odpovědi, dokud nesplňují délku, která je uložena na druhém a třetím bytu prvního rámce (Obrázek 11). Poté se zkontroluje CRC, který je uložen na posledních dvou bytech posledního rámce, a z výsledné zprávy se oříznou první čtyři a poslední dva byty. Výsledek je v podobě pole bytů vrácen jako návratová hodnota.

Třída Device z této knihovny rozšiřuje abstraktní třídu Device z knihovny Device. Jedinou implementací v této třídě je konstruktor, který vytváří instance platformě závislých tříd. Instance této třídy je při spuštění aplikace inicializována a nahrána do statické vlastnosti Instance třídy Device v knihovně Device.

```
Device.Device.Instance = new Device.Droid.Device(this);
```

Zdrojový kód 2: Inicializace knihovny Device při spuštění aplikace.

Tato implementace vychází z programovací techniky zvané dependency injection (viz 2.3.3) upravené pro potřeby této aplikace, čímž je dosaženo určité multiplatformnosti aplikace. Vložení závislosti, což je v tomto případě platforma, probíhá při startu aplikace. Naskytuje mi to také možnost využívat platformě závislé funkce

v implementaci knihovny ENVIS. V případě tvorby aplikace na další platformu by již bylo potřeba implementovat knihovnu Device.

4.3 Aplikace ENVIS a realizace GUI

Diagram tříd pro tuto část aplikace jsem pro jeho rozměry musel přesunout do příloh. Konkrétně Příloha 2 – Class diagramy, Obrázek 13: Struktura aplikace ENVIS pro Android. V diagramu je vidět hierarchie Activit a Fragmentů. Dále sada struktur, statických definicí a nástrojů.

V aplikaci je vytvořena EnvisActivity, která dědí od AppCompatActivity (Activity ze support balíčku pro MaterialDesign [2]), a má implementované funkce pro zobrazení progress indikátoru, nastavení app baru a pro zamykání rozsvíceného displeje. Od této třídy poté dědí každá Activity v aplikaci. Dále je v projektu implementovaná MenuActivity. Tato třída obsluhuje veškerou práci s postranním menu – Vykreslení položek, skrývání/zobrazování menu a přepínání mezi položkami. Dále jsem si připravil ViewPagerActivity, která rozšiřuje MenuActivity a implementuje funkce pro práci se záložkami (viz Obrázek 13: Struktura aplikace ENVIS pro Androi).

DeviceConnectActivity slouží pro připojení k měřicímu zařízení. Její layout se skládá pouze z toolbaru, políčka na vyplnění adresy a tlačítka pro potvrzení. Po připojení socketu ke slave zařízení a úspěšném přijetí validní identifikace zařízení naviguje na ConfInstalationActivity. Když se spojení nepodaří, zobrazí příslušnou hlášku. Pro každou kategorii konfigurace je implementovaná Activity. Tam, kde bylo potřeba rozdělit obsah do záložek, tak Activity slouží pouze pro vygenerování těchto záložek a komunikaci s příslušným managerem.

Tím, jak se Activity v hierarchii dědí od EnvisActivity po ViewPagerActivity, tak jsou do sebe view postupně zabalována. EnvisActivity vloží layout potomků do view s toolbarem, ve kterém jsou připraveny komponenty s nadpisem, tlačítky i progress indikátorem. ManuActivity má k dispozici připravený layout s menu a prostorem pro obsah, do kterého vloží layout potomka. Na stejném principu funguje i ViewPagerActivity.

V případě, že je obsah stránky rozdělený na záložky, tak nad jednotlivými záložkami přebírají kontrolu Fragmenty, které mají jednu parent Activity typu ViewPagerActivity. I mezi fragmenty jsem vytvořil podobnou hierarchii jako u tříd typu Activity. Všechny fragmenty mají za předka třídu EnvisFragment. Tato třída má několikanásobně přetíženou metodu SetValue, která má jako vstupní parametry vždy hodnotu určitého datového typu a nějakou grafickou komponentu, do které má být

hodnota nastavena. Obsluhu manageru i v tomto případě řídí Activity (načtení dat apod.), Fragment z něj potom už jen zobrazuje data ve view.

V aplikaci jsou vytvořené také třídy pro statickou definici prvků výběrů ve Spinnerech (ComboBox) a položek v menu. Tyto prvky jsem jednoduše zapsal jako veřejné atributy tříd, jejichž název končí Spinners rozdělených podle kategorií (CommunicationSpinners, AggregationSpinners apod.). Každá třída pak má veřejné atributy pojmenované podle konkrétního názvu hodnoty. Některé položky Spinnerů jsou generované dynamicky na základě typu zařízení v knihovně ENVIS. Například v InstalationConfigurationManageru je metoda SetConnectionTypesByDevice, která podle typu měřicího zařízení (typ uložený v identifikaci zařízení ve vlastnosti DeviceManageru) určí, jaké kombinace připojení jsou hardwarově možná. Tím se specifikuje výběr položek ve Spinneru pro výběr způsobu připojení.

5 Vyhodnocení řešení

Podle zadání jsem měl za úkol navrhnout knihovny pro konfiguraci a správu dat analyzátorů kvality elektrické energie. Velké množství času jsem strávil samotným návrhem struktury knihoven. Vzhledem ke stavu kódu původní aplikace mi při implementaci zabralo mnoho času také oddělení aplikační logiky od GUI. Z těchto důvodů jsem nestihl implementovat funkce pro práci s archivy a jejich daty. Implementované jsou tedy jen klíčové funkce, jako jsou: práce s binárními konfiguračními soubory, komunikace pomocí protokolu KMB Long přes TCP/IP se zařízením a dekódování vybraných hodnot z univerzální konfigurační struktury.

Pro ověření funkčnosti aplikace jsem se rozhodl změřit některé parametry a mezi ně jsem zařadil měření rychlosti komunikace se zařízením za určitých podmínek a měření výkonu pomocí profileru.

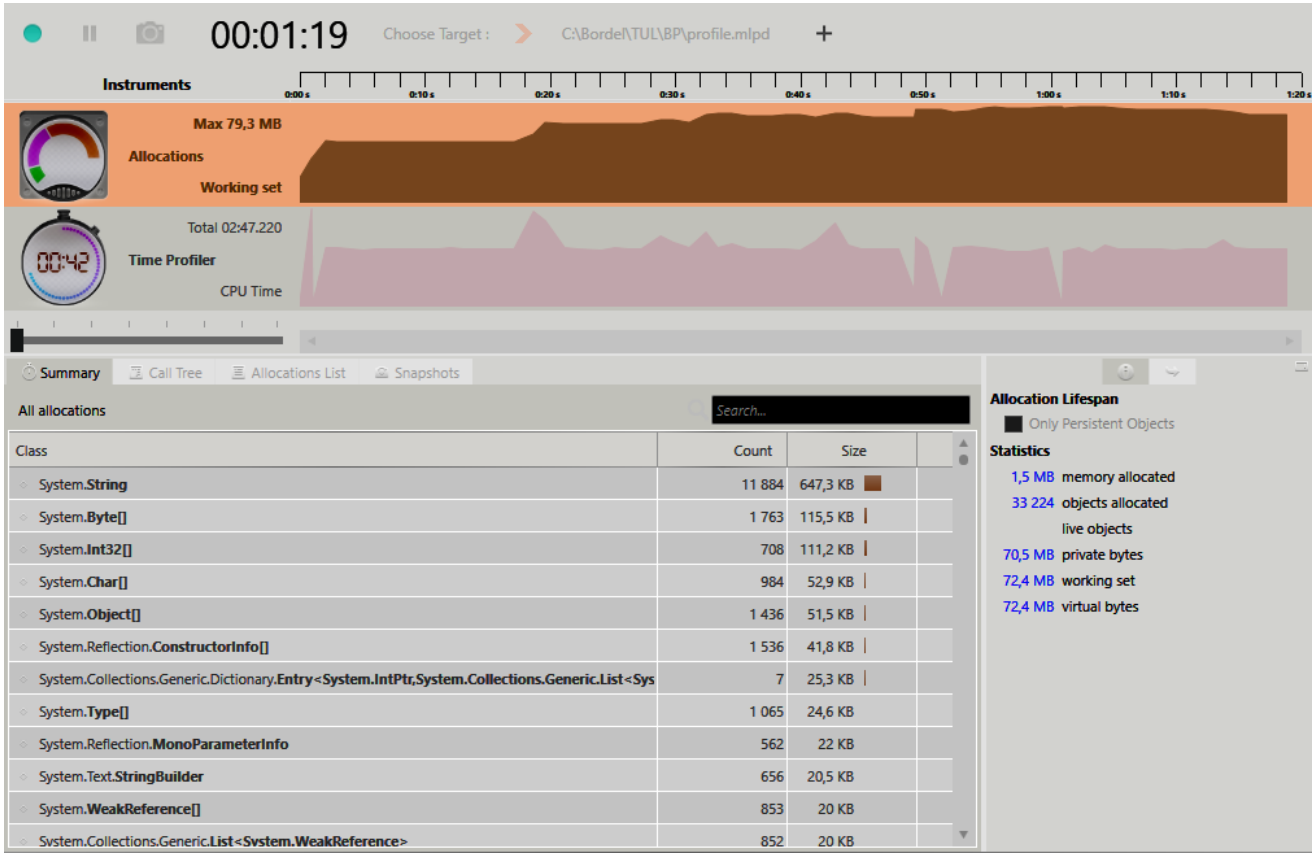
5.1 Profiler

K profilování aplikace jsem využil Xamarin Profiler [18], který je sice ve verzi early preview, ale už je poměrně dobře odladěný. Na screenshotu z profileru (Obrázek 12) je v horní části vidět graf, který ukazuje množství alokované poměti v závislosti na čase. Jelikož probíhá spouštění aplikace a vykreslení grafických prvků na displej, tak graf první dvě sekundy stoupá. Poté je náročnost konstantní (cca 50 MB), dokud se nenapíše adresa a nespustí se připojování ke vzdálenému zařízení. Zde se zvednou paměťové požadavky kvůli staženým konfiguračním datům přibližně o 15 MB. Poté jsou nároky na paměť v podstatě opět konstantní. Při rychlém přepínání stránek se alokovaná paměť dostala na 79,3 MB (nevyšší místo v grafu), což by se při normálním používání nemělo stát, protože by paměť stíhal čistit Garbage Collector.

Pod oběma grafy jsou v tabulce seřazeny datové typy podle toho, kolik v paměti zabírají místa. Jednoznačně nejvíce místa zabírají data typu string a hned po něm byte[] a Int32[]. To je způsobeno univerzální strukturou, která je tvořena z řetězců. Tyto řetězce slouží jako ID jednotlivých kategorií a hodnot, které jsou převážně číselné.

Druhý graf na screenshotu z profileru (Obrázek 12) znázorňuje vytížení procesoru. Na něm je vidět největší vrchol při spouštění. Nicméně si myslím, že zde není mnoho prostoru pro optimalizaci. Druhý největší vrchol nastane při komunikaci se vzdáleným měřičem. V tuto chvíli probíhá připojování socketu, stažení identifikace měřiče a stažení všech konfiguračních dat. Zde by se při bližším prozkoumání určitě

nějaká optimalizace našla. Ostatní menší vrcholy nastanou při přepínání mezi jednotlivými stránkami. Dochází zde k vykreslování GUI a načítání hodnot z univerzální struktury.



Obrázek 12: Výstup z profileru. Nahoře graf paměťové náročnosti v čase, pod ním graf využití procesoru a dole tabulka zastoupení jednotlivých datových typů.

5.2 Orientační měření rychlosti komunikace

Rychlost komunikace jsem měřil jen na jednom zařízení s Androidem a pouze na jednom měřiči kvality elektrické energie. Výsledky bych tudíž považoval jen za orientační. I tak jsem se ale snažil dodržet určité podmínky popsané v dalším odstavci, při kterých bylo měření prováděno.

Měření probíhalo na mém mobilním zařízení Lenovo P70-A s verzí OS Android 4.4.4. Při měření nebylo zařízení připojené k internetu a neběžela na pozadí žádná aplikace. Komunikace probíhala na uzavřené lokální síti. Měřič SMY 133 U 230 X/5A RI E byl propojen s notebookem přes Ethernet. Počítač by použit jako AP, přes které byl do sítě pomocí WiFi připojen telefon. Měřený časový úsek vždy zahrnoval zprávu směrem k zařízení, zprávu zpět od zařízení k aplikaci a zpracování přijatých dat. Tyto intervaly jsem měřil nejjednodušším možným způsobem. Přímo v kódu aplikace jsem si ukládal do statických proměnných časová razítka začátku a konce měřené události.

Měřil jsem tři případy komunikace – připojení k zařízení, stažení konfiguračních dat i s popisem struktury a odeslání dat. Připojení k zařízení, kde dochází k vytvoření spojení pomocí socketu a k identifikaci zařízení, trvalo 240 – 250 milisekund. Stažení dat, jak už jsem zmiňoval výše (viz 3.2), je rozdělené do tří částí. Stažení kategorií konfigurační struktury, včetně zpracování dat ve struktuře trvalo 115 – 125 milisekund, stažení popisu hodnot konfigurace trvalo včetně zpracování 30 – 35 milisekund a stažení a uložení samotných dat potom už jen 20 – 25 milisekund. Celkem tedy stažení dat trvalo 165 – 185 milisekund. Poslední měřenou hodnotou bylo odeslání dat, kde jsem naměřil 70 – 80 milisekund.

6 Závěr

Podle funkcí aplikace ENVIS.Daq jsem navrhl knihovny pro konfiguraci podporovaných měřicích přístrojů elektrické energie firmy KMB systems s. r. o. Třídy pro vytváření univerzální konfigurační struktury a zpracování přijatých dat jsem použil ze vzorové aplikace pro OS Windows. Jedná se o různé fragmenty tříd, které bylo potřeba vytřídit za asistence autora zdrojových kódů. Dále jsem podle vzorové aplikace navrhnul grafické uživatelské rozhraní napojené na tyto knihovny. Aplikace pro Android podporuje verze OS s API 16 a vyšší (až API 23), což znamená verze Android 4.1 Jelly Bean až Android 6.0 Marshmallow (95 % všech zařízení registrovaných v Google Play [4]).

Pro naplnění univerzální konfigurační struktury daty bylo potřeba, abych napsal třídu, která zprostředkuje komunikaci přes síť pomocí protokolu KMBLong v TCP. Metody této třídy mají na starost připojení a komunikaci se zařízením. Zde jsem vycházel z techniky dependency injection (upravené pro potřeby aplikace) a vytvořil jsem platformě nezávislou (portable) knihovnu Device, která implementuje abstraktní třídu pro komunikaci pomocí socketu, třídu pro práci se soubory a třídu, jež definuje a drží instance tříd využívajících platformě závislých funkcí. V knihovně Device.Droid jsou tyto funkce implementovány pro platformu Android v rámci tříd dědících od tříd knihovny Device. Při spuštění aplikace jsou tyto třídy inicializovány do statické instance knihovny Device. To mi umožňuje používat platformě závislé funkce i z platformě nezávislé (portable) knihovny ENVIS. Stejným způsobem jsou implementované funkce pro práci s binárními soubory. Do těchto funkcí patří načtení souboru, uložení souboru a výpis složky se soubory.

Knihovna ENVIS implementuje veškerou business logiku. Pracuje s třídami vybranými ze vzorové aplikace pro Windows a využívá k tomu platformě závislých funkcí definovaných v knihovně Device. V této knihovně jsem také implementoval sadu třídy, které zprostředkovávají práci s univerzální datovou strukturou. Tyto třídy dekódují a zpracovávají hodnoty této struktury a tím ji zapouzdřují. Jednotlivé dekódované konfigurační hodnoty jsou směrem ven z knihovny implementované jako veřejné vlastnosti, které jsou implementované jako nullable hodnoty. V případě, že konkrétní konfigurace není pro dané zařízení přístupná, tak příslušné vlastnosti jsou nastaveny na hodnotu null.

Při návrhu grafického uživatelského rozhraní pro tuto aplikaci jsem se inspiroval v původní aplikaci pro Windows, ale některé prvky jsem musel s ohledem na zvyklosti

prostředí Android upravit, případně přemístit vzhledem k displeji. Většinu stránek s konfiguračními daty bylo kvůli velkému množství dat potřeba rozdělit do záložek. Díky tomu, že je veškerá logika implementována v knihovně ENVIS, tak jednotlivé controllery aplikace v podstatě jen zobrazují veřejné vlastnosti managerů v jednotlivých view.

Díky způsobu, jakým jsem navrhnul knihovnu Device a jak ji knihovna ENVIS využívá, by byla případná implementace aplikace pro jinou platformu velmi jednoduchá. Bylo by jen potřeba doimplementovat platformě závislé funkce knihovny Device a napojit samotnou aplikaci na knihovnu ENVIS. Díky tomu tento projekt vybízí k rozšíření na další mobilní platformy jako je iOS, Windows Phone, popřípadě UWP. Další prací na této aplikaci by mohlo být také grafické doladění. Postranní menu by bylo vhodné přepracovat na klasické, vysouvací Hamburger menu. Na obrazovce by tím vzniklo na šířku více místa a aplikace by poté mohla podporovat i orientaci na výšku.

Seznam použité literatury

[1] *Activity. Android Developers. [online]. [cit. 2016-03-15]. Dostupné z: http://developer.android.com/reference/android/app/Activity.html*

[2] *Android Support Library v7 AppCompat. Xamarin. [online]. [cit. 2016-05-14]. Dostupné z: https://components.xamarin.com/view/xamandroidsupportv7appcompat*

[3] *Ing. Andrea Ronešová. Přehled protokolu MODBUS [online]. 2005. [cit. 2016-05-13]. Dostupné z: http://home.zcu.cz/~ronesova/bastl/files/modbus.pdf*

[4] *Dashboards. Android Developers. [online]. 4. 3. 2015 [cit. 2016-03-15]. Dostupné z: https://developer.android.com/about/dashboards/index.html*

[5] *ENVIS ver. 1.2. KMB systems. [online]. [cit. 2016-03-15]. Dostupné z: http://www.kmb.cz/index.php/cs/aplikace/envis*

[6] *Fragment. Android Developers. [online]. [cit. 2016-03-15]. Dostupné z: http://developer.android.com/reference/android/app/Fragment.html*

[7] *How mobile is your .NET?. Xamarin. [online]. [cit. 2016-05-14]. Dostupné z: https://scan.xamarin.com/*

[8] *Introduction to Mobile Development. Xamarin. [online]. [cit. 2015-03-15]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/*

[9] *Material Pager Sliding Tab Strip for Xamarin.Android. github. [online]. [cit. 2016-05-14]. Dostupné z: https://github.com/jamesmontemagno/PagerSlidingTabStrip-for-Xamarin.Android*

[10] *McConnell Steve. Dokonalý kód: umění programování a techniky tvorby software. Brno: Computer Press, a.s., 2006. ISBN 80-251-0849-X.*

[11] Microsoft Developer Network. *Dependency Injection.* [online]. [cit. 2016-05-11]. Dostupné z: *https://msdn.microsoft.com/en-us/library/ff921152.aspx*

[12] Microsoft Developer Network. *Inversion of Control.* [online]. [cit. 2016-05-11]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff921087.aspx

[13] Microsoft Developer Network. *Model-View-Controller.* [online]. [cit. 2016-05-11]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff649643.aspx

[14] Microsoft Developer Network. *Structural Patterns: Adapter and Façade.* [online]. [cit. 2016-05-11]. Dostupné z: https://msdn.microsoft.com/en-us/library/orm-9780596527730-01-04.aspx

[15] *Mobile Application Development to Build Apps in C#. Xamarin. [online]. [cit. 2016-04-27]. Dostupné z: http://xamarin.com/platform*

[16] *PA 144, SMC 144, SMV, SMVQ, SMP, SMPQ Multifunctional Panel Meters & Power Quality Analyzers Communication Protocol Manual* [online]. 2012. [cit. 2016-05-11]. Dostupné z: *http://www.kmb.cz/index.php/cs/component/phocadownload/category/14-dokumenty-komunikace?download=213:popis-komunikacnich-protokolu-modbus-kmblong*

[17] *Part 1 – Understanding the Xamarin Mobile Platform. Xamarin. [online]. [cit. 2015-03-15]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/*

[18] *Say hello to Xamarin Profiler Preview.. Xamarin. [online]. [cit. 2016-05-14]. Dostupné z: https://www.xamarin.com/profiler*

Seznam příloh

Přiložené CD

Na přiloženém CD je kromě tohoto dokumentu i vlastní program.

Popis jednotlivých adresářů:

- Kořenový adresář
    - Readme.txt – popis adresářů
    - Install.txt – návod k instalaci programu
- ENVIS – adresář s programem
    - ENVIS.sln – soubor projektu pro Visual Studio
    - Device – adresář s implementací knihovny Device
    - Device.Droid – adresář s implementací knihovny Device.Droid
    - ENVIS – adresář s implementací knihovny ENVIS
    - ENVIS.Droid – adresář s implementací aplikace ENVIS
- package
    - cz.kmb.envis – Signed – instalační balíček aplikace pro Android
- text
    - BP_Eisler_David.pdf tato práce ve formátu PDF

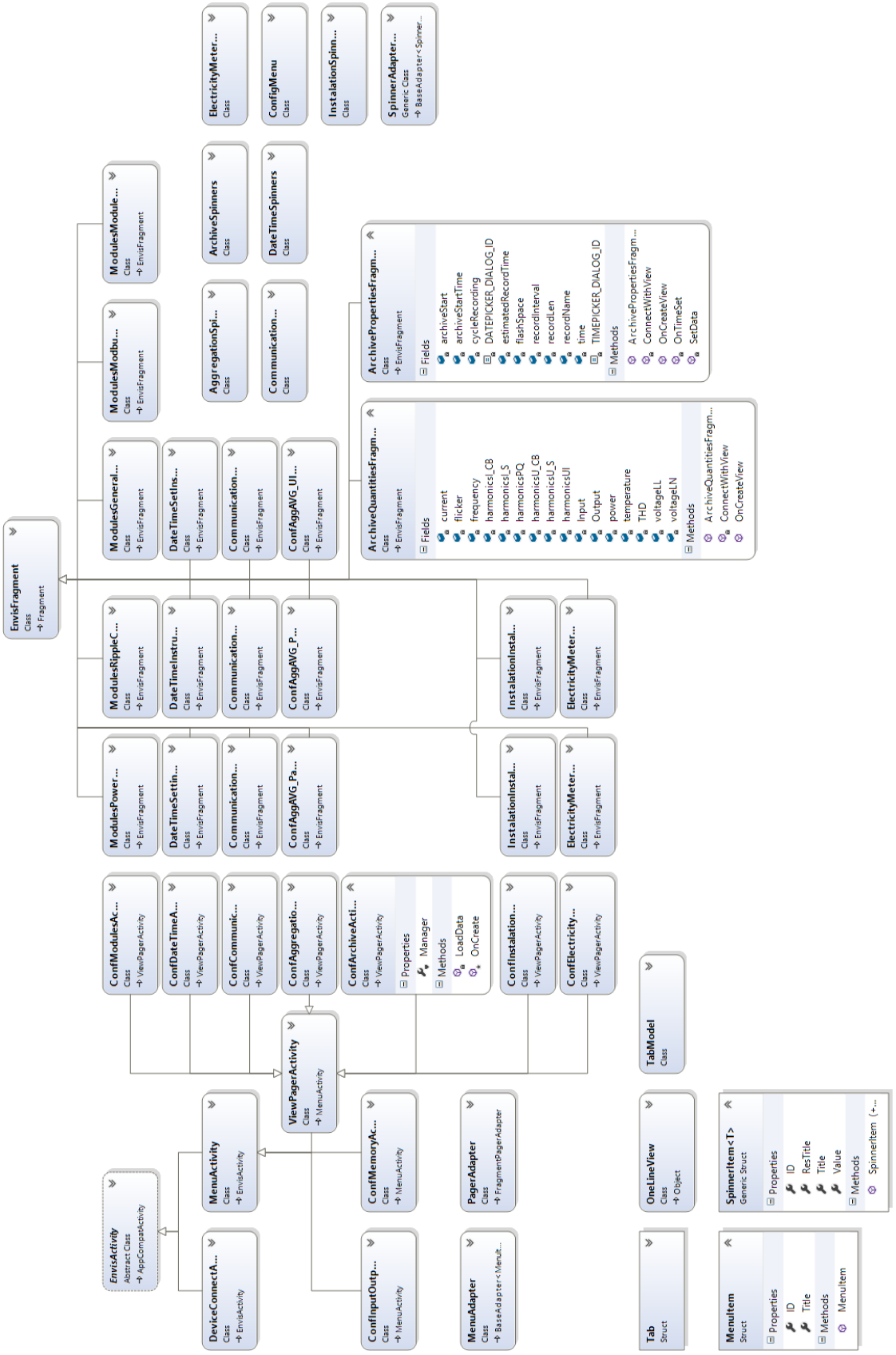
Příloha 1 – Instalace aplikace

Na přiloženém CD je k dispozici instalační balíček výsledné aplikace s příponou APK. K jeho instalaci je zapotřebí mít zařízení s operačním systémem Android 4.1 Jelly Bean až Android 6.0 Marshmallow. Verzi Androidu na svém zařízení zjistíte v Nastavení -> Informace o telefonu -> Verze systému Android.

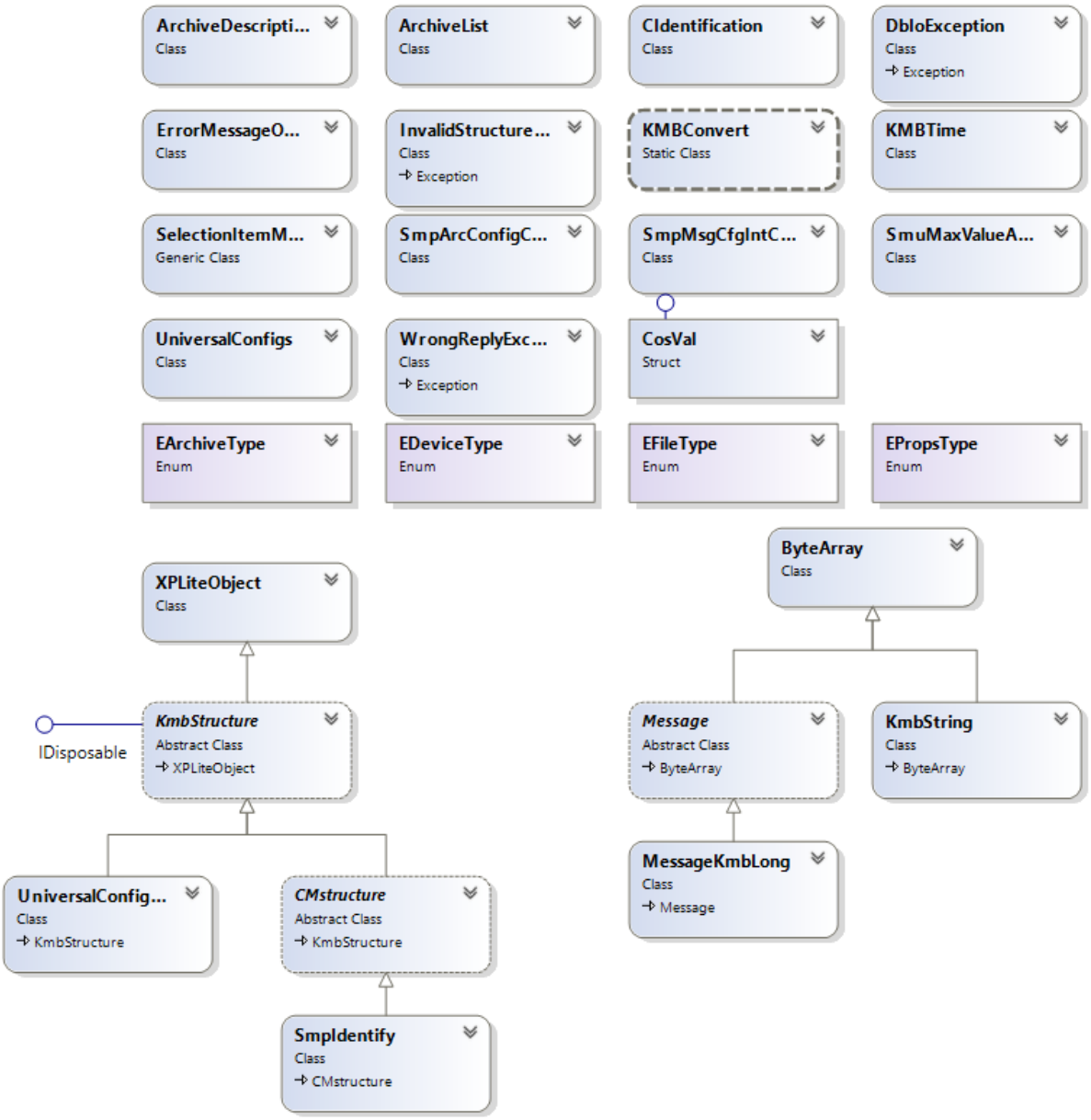
Dále je potřeba na vašem zařízení povolit instalaci aplikací z neznámých zdrojů. To provedete v Nastavení -> Zabezpečení -> Neznámé zdroje. Tato položka musí být zaškrtnutá.

Nakonec už je potřeba nakopírovat balíček do nějakého přístupného adresáře v telefonu přes USB (nebo poslat e-mailem apod.), poté pomocí nějakého prohlížeče souborů daný balíček nalézt, kliknout na něj a zvolit instalovat. Tímto by měla být instalace dokončena a aplikaci je možné spustit.

Příloha 2 – Class diagramy

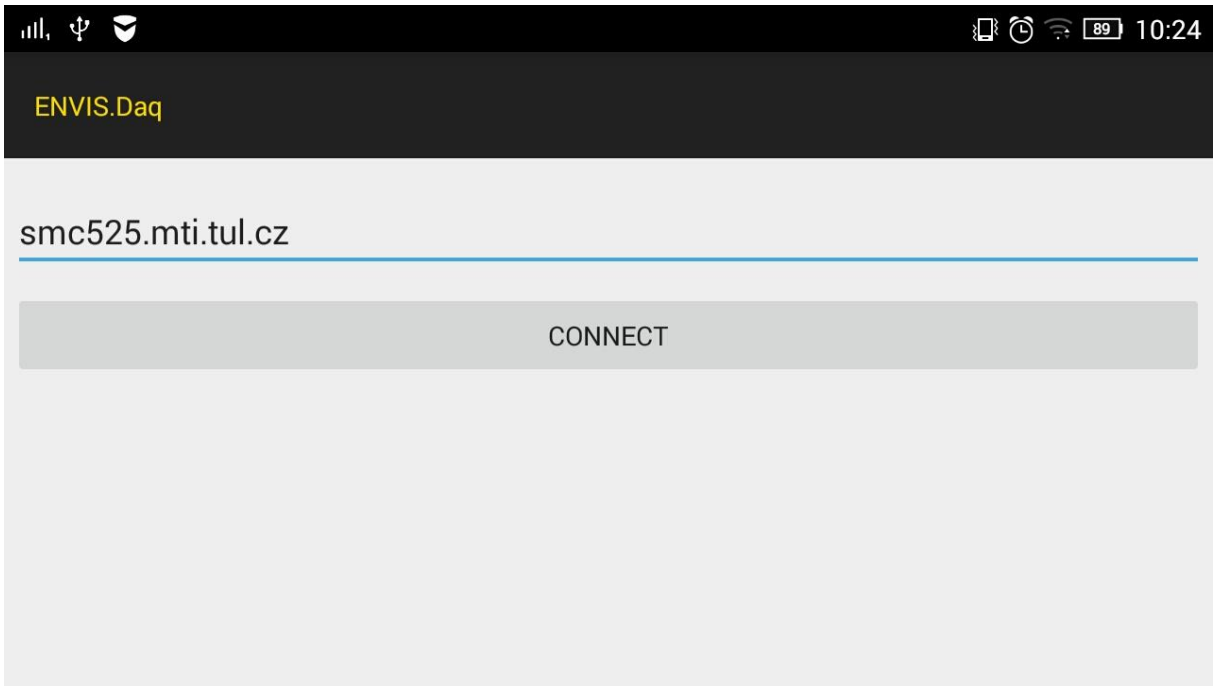


Obrázek 13: Struktura aplikace ENVIS pro Android. V diagramu je vidět hierarchie Activit a Fragmentů. Dále sada struktur, statických definicí a nástrojů.

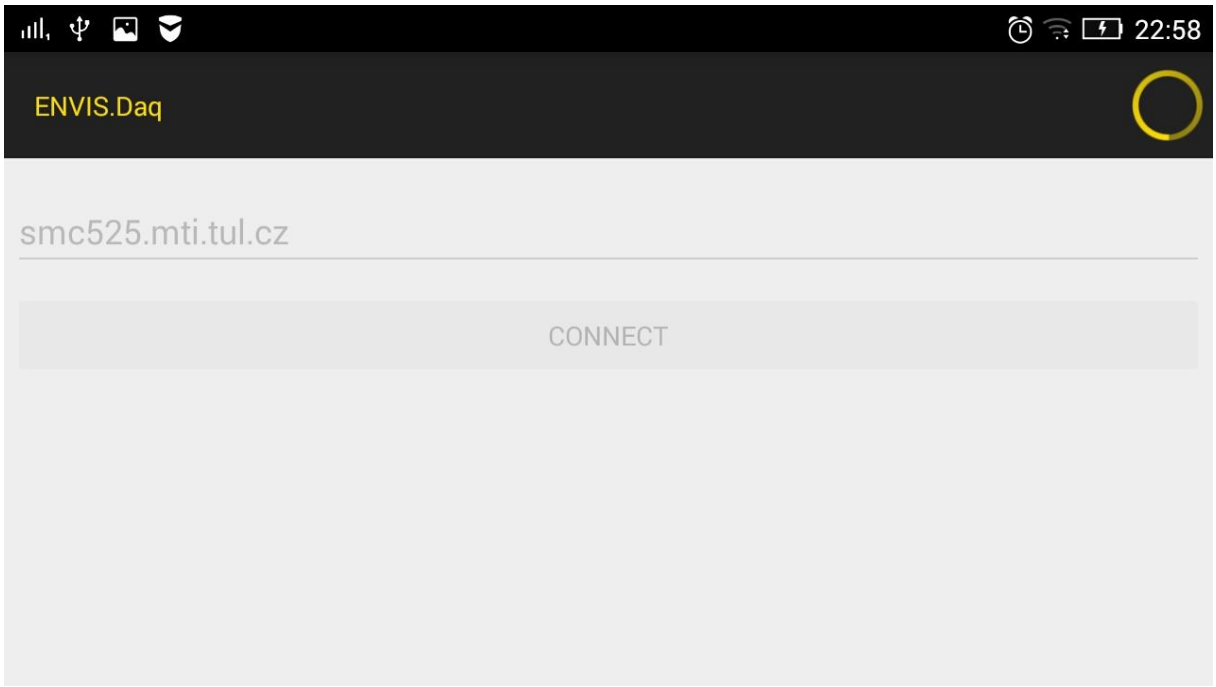


Obrázek 14: Třídy a datové typy vykopírované z původní aplikace pro OS Windows.

Příloha 3 – Screenshoty



Obrázek 15: Stránka pro připojení k zařízení - DeviceConnectActivity



Obrázek 16: Probíhá připojení k zařízení - interakce s grafickými prvky je vypnutá