VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

BRÁNA INTELIGENTNÍ DOMÁCNOSTI

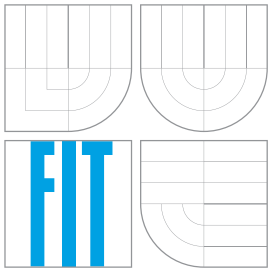
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ČECHMÁNEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

BRÁNA INTELIGENTNÍ DOMÁCNOSTI

GATEWAY FOR SMART HOME

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ČECHMÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ NOVOTNÝ

BRNO 2015

Abstrakt

V současné době jsou v oblasti inteligentního řízení domácnosti kladeny požadavky na miniaturizaci, zabezpečení a pokročilou logiku řídicích členů. Výzvou je také způsob sběru dat z koncových členů a jejich zpracování, včetně jejich distribuce dalším uzlům v řetězci inteligentní domácnosti. Cílem práce je navrhnout a vytvořit člen systému pro řízení inteligentní domácnosti – bránu mezi senzorickou sítí a databázovým serverem, která bude implementována na vestavěném systému a se specifickými HW prostředky. Funkce brány by měly zastřešovat i chybové stavy, například uchování zpráv ze senzorů při přerušení spojení se serverem či řešení výpadku synchronizace času. Práce se zabývá jednak teoretickým základem pro vytvoření této brány, jednak i praktickou implementací na určené platformě.

Abstract

In area of Smart-home control, there are requirements for miniaturisation, security and advanced control logic of individual member of system at present. One of the challenges is the way of data collecting from terminal nodes and data processing, including their distribution to the others nodes of Smart-home. The goal of this thesis is to design and to produce Smart-home control node, which will be a gateway between sensoric net and database and will be developed and produced on embedded system with specific HW boards. Functionality of this gateway should include error states, e.g. should cache sensor messages when the connection between gateway and server interrupts, or take care about time synchronising. The thesis describes theoretical side of gateway design and also practical side of programming and producing SW for the gateway in Smart-home system.

Klíčová slova

adaptér, inteligentní domácnost, senzor, vestavěný systém

Keywords

adapter, smart home, sensor, embedded system

Citace

Martin Čechmánek: Brána inteligentní domácnosti, diplomová práce, Brno, FIT VUT v Brně, 2015

Brána inteligentní domácnosti

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Tomáše Novotného.

.....
Martin Čechmánek
21. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu této práce Ing. Tomáši Novotnému za poskytnutou pomoc a rady, Ing. Josefu Hájkovi za pomoc a konzultace při vytváření komunikačního protokolu s rozšiřující deskou, své rodině a všem, kteří mě v průběhu tvorby práce podporovali.

Obsah

Kapitola 1

Úvod

Pojem inteligentní domácnosti není ve světě vestavěných systémů žádnou novinkou, naopak – od inteligentní správy nemovitostí malých i velkých korporací se pozvolna dostává inteligence i do běžných domácností. S rozmachem tohoto odvětví se inteligentní správa domu stává čím dál více dostupnější širší základně uživatelů. V současné době je vyvíjen tlak na snižování pořizovacích nákladů spojených s integrací systému pro monitorování a řízení domácnosti, spolu s požadavkem na jednoduchou instalaci systému, snadné ovládání a zabezpečení tohoto systému.

1.1 Cíl práce

Cílem této práce je představit systém inteligentní domácnosti se zaměřením na jeho specifickou část – bránu mezi senzorickou sítí a databázovým serverem, implementovanou na open-hardware platformě **OLinuXino** [20] od firmy *Olimex*. Součástí práce je vytvoření aplikace pro bránu inteligentní domácnosti, návrh a implementaci modulu pro otestování komunikace brány se senzorovou sítí a aplikování vytvořeného SW na cílovou platformu.

1.2 Struktura práce

V kapitole 3 jsou představeny vývojové desky, které by mohly sloužit pro reálné nasazení budoucí aplikace. Podrobněji bude popsána platforma OLinuXino a spolu s ní model *A10-OLinuXino-LIME*, na kterém bude v praktické části práce demonstrována aplikace pro *bránu inteligentní domácnosti*, která bude v rámci této práce označována jako **adaptér**. Aplikace, která na tomto adaptéru poběží, bude souhrnně označována jako **AdaApp**.

Návrh architektury AdaApp a modulu, který usnadní otestování komunikačního protokolu AdaApp a nahradí reálnou senzorovou síť její virtuální podobou, bude podrobně popsán v kapitole 4.2. Bude zde vysvětlena struktura modulu, jeho integrace v budoucí aplikaci a možnosti jeho nastavení pro potřeby testování.

Kapitola 5 popisuje konkrétní podobu praktické části této práce. V úvodu této kapitoly je nastíněn postup implementace aplikace, spolu plánem vývoje. Dále jsou zde popsány konkrétní podoby jednotlivých modulů, tak i příprava AdaApp pro distribuci zpracovávaných dat pro externí aplikace. Je zde také nastíněna komunikace mezi uzly, stejně jako vnitřní architektura AdaApp a výčet používaných struktur.

Závěr obsahuje shrnutí práce, splnění cílů a úvahu nad možnými rozšířeními vyvinutého řešení.

Kapitola 2

Inteligentní domácnost

Inteligentní domácnost (někdy také jako **Inteligentní dům** či **Smart-home**) je souhrnný pojem označující budovy vybavené jistým druhem inteligence či řízení, který má majiteli zajistit vyšší komfort bydlení, vyšší zabezpečení, nižší spotřebu energií nebo se jinak podílet na zlepšení domácnosti. Do této kategorie se v tom nejširším slova smyslu řadí domy, které mají jen základní kamerový systém až po ukázkové domy budoucnosti protkané technologiemi, senzory a inteligentními prvky. Myšlenka inteligentního domu však není nová – koncept automatizovaného domu už existuje od 50. let 20. století s myšlenkou využít prostředky pro ovládání topení, audio/video systémy a další [26].

Pokud bereme „běžný dům“ jako dům, kde máme určité komponenty inteligentní domácnosti, pak tyto prvky většinou mají každý jiný tvar, styl řízení a nekomunikují spolu. Inteligence domácnosti spočívá právě ve vzájemném propojení komponent systému a umožnění využití všech dostupných inteligentních prvků pro zajištění výše zmíněných přínosů pro dům a jeho majitele [26].

Díky tomuto propojení pak lze využít všech řiditelných komponent a umožnit nejen vzdálené řízení inteligentní domácnosti, ale také řízení automatizované. To pak skrývá velký potenciál ke zvýšení komfortu uživatele, jelikož je možné určité operace dopředu plánovat, nastavovat jejich trvání i intervaly a v neposlední řadě vytvářet tzv. *scény*, což jsou režimy přednastavených činností, které lze vyvolat např. stiskem jednoho tlačítka. Tyto scény se pak uplatní pro odchod poslední osoby z domu, kdy se při stisknutí tlačítka aktivace scény například zavřou všechna okna, zhasnou světla a zapne zabezpečovací systém. Jelikož jsou v ideálním případě všechny ovládací prvky i vypínače konfigurovatelné, tedy nemají pevně přiřazený účel, lze scény spínat odkudkoliv, měnit jejich posloupnost i funkci [26].

Z hlediska současných trendů pak řízení inteligentní domácnosti využívá chytrých telefonů, televizí a obecně moderních technologií. To umožňuje uživateli ještě větší mobilitu pro ovládání domácnosti, čtení hodnot ze senzorů aj. Dále se také využívá více dotykových panelů a ovládacích prvků před běžnými mechanickými spínači. V neposlední řadě je velmi praktické využívání bezdrátových technologií pro řízení domácnosti, jelikož nevyžadují zásadní zásahy do infrastruktury budovy a poskytují stejnou funkcionalitu jako drátové řešení.

2.1 Přehled možností v řízení inteligentní domácnosti

Systém pro řízení inteligentní domácnosti tedy skrývá několik úrovní hierarchie, různě zatížené komponenty o různých velikostech, různé požadavky na odolnost, stálost, výdrž apod.

a další faktory, které je potřeba při návrhu tohoto systému zvážit. Komponenty inteligentní domácnosti mohou být různé:

- *signalizační* – zobrazovací prvky,
- *snímací* – senzory nejrůznějších veličin, kamery,
- *s možností vykonat nějakou činnost* (neboli **aktory**) – ovládání garážových vrat, domovních oken, čerpadlo pro zavlažování aj.,
- *řídicí* – řízení sběrnice, centrála,
- *pro uchování dat* – pevné disky, databázový server,
- *komunikační prvky* – bezdrátové/drátové technologie

a jiné. Z těchto komponent se pak výsledný systém skládá do komplexního systému, který umožňuje ovládat/snímat každou část domu a s dodaným SW i vykonávat funkce pro usnadnění chodu domácnosti, snížit energetické náklady domu a nebo také zvýšit zabezpečení domu.

V případě, že je systém pro řízení domácnosti univerzální, lze jej využívat nejen jako prostředek pro ovládání inteligentní domácnosti, ale i jako jiné užitečné komponenty (části inteligentní domácnosti). Ku příkladu je možné systém využít jako ([11]) :

- **domácí meteostanice** – umožní zobrazovat informace ze senzorů v různých místností,
- **termoregulace** – umožní inteligentní řízení kotle v závislosti na okolních podmínkách,
- **elektronický zabezpečovací systém** – spustí zaznamenávací zařízení při detekci pohybu, ohlásí pohyb osob přes GSM síť na mobil majitele domu nebo se postará o automatické zavření všech oken a dveří při odchodu z domu pryč,
- **inteligentní zalévání** – systém spustí zavlažování trávníku/kytek/zahrady/skleníku v určitých časových intervalech, nebo v případě využití senzoru vlhkosti zalévá v závislosti na půdní vlhkosti,
- **simulace osob v domě** – systematickým rozsvěcováním žárovek v domě lze simulovat přítomnost osob v domě, což by mělo odradit zloděje i když fyzicky doma nejsme.

Takovýto systém pak pochopitelně nabírá na univerzalitě, lze ho využít na různé způsoby, využít komponent původního systému, což znamená také výslednou finanční úsporu. A pokud uživatel přestane systém potřebovat, může ho převést na jiný systém s nulovými finančními a minimálními časovými náklady.

2.2 Existující řešení pro řízení domácnosti

Existujících řešení je s pokrokem vývoje integrovaných obvodů, vývojových platforem a větší integrací stále více. Na českém trhu je jeden z hlavních představitelů společnost **ELKO EP**, která se zabývá domovními i průmyslovými elektroinstalacemi, včetně těch inteligentních. Jejich produkt **iNELS** pro řízení inteligentní domácnosti, který existuje ve sběrnicovém

i bezdrátovém provedení, získal mnoho ocenění doma i v zahraničí. Jejich předváděcí místnost je možné po stažení jejich promo aplikace ovládat a sledovat dálkově odkudkoliv a vyzkoušet tak potenciál jejich produktů [8].

Dalším českým představitelem je například společnost **HAIDY**, která nabízí komplexní systém pro inteligentní domy v různých úrovní produktů. Stejně jako ELKO EP nabízí demo verzi svého produktu, nicméně pouze jako virtuální variantu ovládaného domu. Podobně je na tom i společnost **Cannynet**, která kromě inteligentní domácnosti nabízí i inteligentní ovládání domu bez potřeby zásahu do stávající elektroinstalace. Taktéž nabízí demo ukázku svého produktu na svých stránkách [9][7].

Ze zahraničních představitelů je to například společnost **Belkin**, která taktéž nabízí SW pro ovládání inteligentní domácnosti zvaný **WeMo** spolu se spoustou HW komponent pro inteligentní domy. Na svých internetových stránkách nabízí tyto produkty přímo k zakoupení, avšak narozdíl od českých konkurentů na svých stránkách nenabízí možnost vyzkoušení aplikace pro řízení domácnosti [5].

Jako zástupce open-source varianty je například společnost **Open Source Automation**, která poskytuje volně dostupný automatizační software pro ovládání domácnosti na platformě Windows. Umožňuje ovládat světla, audio systémy, zabezpečit dům aj., přičemž je jednoduchý k použití a uživatelsky přizpůsobivý. Jejich stránky obsahují tutoriály, návody a popisy jejich řešení, které je neustále ve vývoji, takže se možnosti Open Source Automation nadále rozrůstají [21].

2.3 Popis IoT projektu a jeho architektury

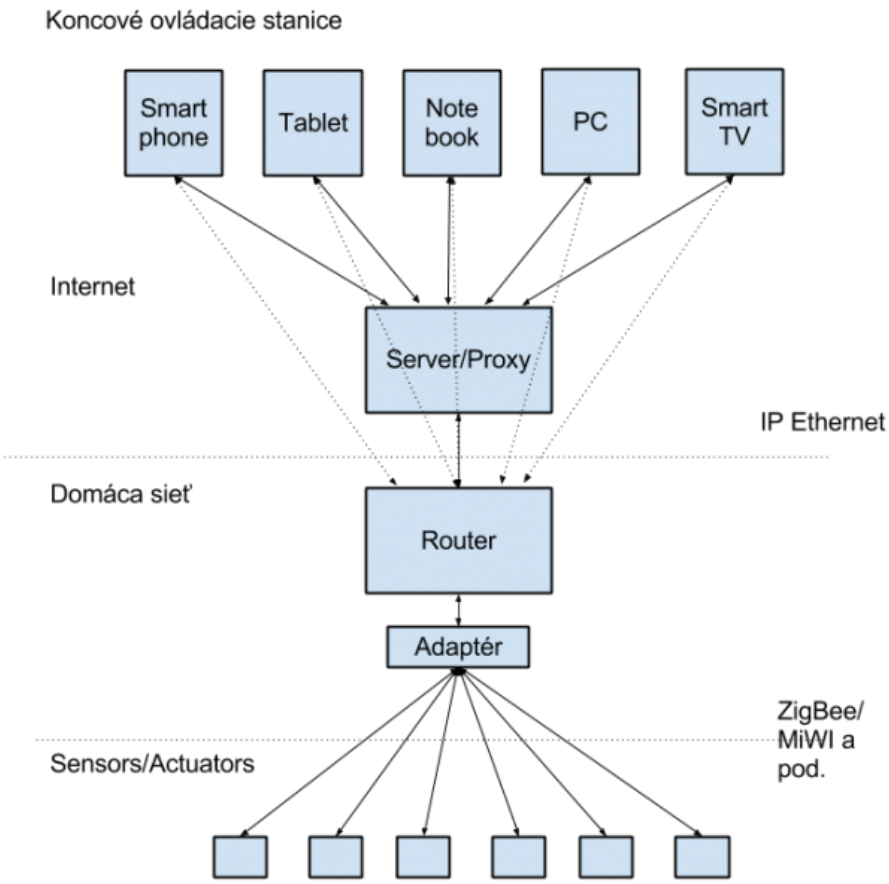
Projekt **Internet of Things** (zkráceně IoT), který je řešen na Fakultě informačních technologií na VUT v Brně a ve kterém je řešena také tato práce, má za cíl vyvinout systém pro řízení inteligentní domácnosti s důrazem na širokou škálu použití za nízkou cenu výsledného řešení. Struktura systému je znázorněna na obrázku 2.1 a obsahuje několik základních vrstev, jejichž funkce bude stručně vysvětlena v následujícím textu. Informace pro následující text jsou čerpány z [10].

2.3.1 Koncové ovládací stanice

Jedná se o souhrn všech zařízení, ktere mohou ovládat inteligentní domácnost. Patří mezi ně hlavně chytré mobilní zařízení, PC, a v neposlední řadě i chytré televize. Pro mobilní zařízení platformy Windows phone a Android existuje aplikace pro inteligentní domácnost, která umožňuje zobrazovat hodnoty senzorů v domě, graficky zobrazovat jejich vývoj, ovlivňovat intervaly výčtu hodnot ze senzorů a také provádět úkony pomocí ovládaní aktorů. Pro stolní počítače s OS Windows 8.1 tato aplikace existuje též, pro chytré televize se aplikace teprve bude připravovat.

2.3.2 Server

Vrstva nezbytná pro ukládání dat do databáze, lokální i vzdálenou komunikaci mezi mobilním zařízením a zbytkem systému a generování notifikací pro koncové ovládací stanice. Pracuje nad IP protokolem a TCP vrstvou, přenosy mezi mobilními zařízeními a serverem probíhají zabezpečeně a šifrovaně, totéž platí i pro komunikaci s protější stranou systému. Tou je adaptér (brána inteligentní domácnosti), který slouží jako spojení mezi senzorovou sítí a serverem. Server tedy komunikuje jak s koncovým ovládacím zařízením, od kterého



Obrázek 2.1: Architektura IoT projektu, jež je brána inteligentní domácnosti součástí. Obrázek převzat z [10].

sbírá požadavky a odesílá jejich odpovědi, tak i s adaptérem, od kterého přijímá data ze senzorů a odesílá požadavky na ovládání domácnosti.

2.3.3 Brána inteligentní domácnosti

Vrstva, která je řešena v rámci této práce. Jak již bylo zmíněno výše, jedná se o prvek komunikující se serverem – dostává od něj požadavky a ty pak přeposílá dále na senzory. Z těch pak přijímá data, zpracovává je a odesílá na server, kde probíhá dále jejich zpracování a uložení do databáze. K tomu bude využívat zvolenou vývojou platformu, ke které bude připojena rozšiřující DPS s modulem pro bezdrátovou komunikaci přes protokol MiWi.

Brána by také měla umět uchovávat příchozí zprávy ze senzorů v případě výpadku spojení se serverem a hlídat správnou časovou synchronizaci zařízení. Více o tomto uzlu bude popsáno v dalších kapitolách této práce.

2.3.4 Senzory a aktory

Koncový prvek inteligentní domácnosti, který plní funkci sběru dat z domácnosti či jejích akčních členů. Existuje několik typů snímaných veličin, které v současné době desky, na nichž jsou senzory umístěny, podporují, stejně tak jako ovládání akčních členů. Tyto desky

budou komunikovat s adaptérem přes bezdrátový protokol MiWi, který umožňuje komunikovat v řežimu *point-to-point* i jako *senzorická síť*. U těchto uzlů je potřeba při konstrukci i programování dbát na spotřebu energie, protože jsou napájené z baterií (týká se pouze senzorů).

2.4 Výhled do budoucnosti

V oblasti řízení inteligentních domácností je široké spektrum možností zdokonalení jednotlivých komponent systému. Současný trend aplikací pro mobilní zařízení se začíná projevovat i v oblasti nositelné elektroniky, inteligence domů začíná používat *učení* jako prostředek pro zlepšení spotřeby domu, naučení se opakovaných akcí v určitou dobu a jejich předvídání a další [26].

Také z hlediska HW jsou prováděny neustálé pokroky a s využitím bezdrátových technologií lze zavést inteligenci do všech myslitelných oblastí, ať už inteligentních domácností, budov či výrobních podniků. Výrobci však nesmějí zapomínat na cenu, která je důležitým faktorem při zvažování nasazení jejich systému do zákazníkem vybrané nemovitosti.

Velmi populárním trendem je v dnešní době přidávání inteligence do bílého zboží. Ať už se jedná o pračky, ledničky, kávovary a jinou techniku, výrobci integrují inteligenci do těchto druhů produktů pro zvýšení komfortu potenciálních zákazníků. Například chytrá lednice pak může monitorovat svůj obsah, vytvářet majiteli seznam, co by měl v obchodě nakoupit pro doplnění zásob, nabízet recepty pro vaření z dostupných potravin a v neposlední řadě upozorňovat na potraviny, kterým se blíží konec trvanlivosti [13].

Další výzvou, kterou tato nová inteligentní zařízení přinášejí, je jejich vzájemné propojení včetně systému pro řízení inteligentní domácnosti, který by pak mohl po probuzení uživatele automaticky vyslat příkaz pro uvaření ranního teplého nápoje ke snídani, vypnutí zapnutých plotýnek sporáku či troubu při odchodu z domu, automaticky spínat praní prádla při přechodu na nízký tarif energií apod.

Kapitola 3

Prostředky pro vývoj

Tato kapitola poskytne přehled o nástrojích pro vytvoření brány inteligentní domácnosti. Budou zde zmíněny vývojové platformy, knihovna POCO a další prostředky pro vývoj aplikace AdaApp.

3.1 Přehled vývojových platforem

V současné době je pro vývojáře výzvou realizovat své projekty s minimálními náklady a rozměry výsledku. Na to reagují i výrobci vývojových desek, a díky velkému rozmachu této oblasti je v současnosti dostupná velká škála vývojových platforem, které nabízejí různé modely s různým vybavením pro pokrytí velké části trhu.

Pro výslednou realizaci bylo potřeba vybrat jednoho z nich, který bude poskytovat dostatečný výkon za přijatelnou cenu desky. Všichni představitelé výběru musejí splňovat následující podmínky:

- cena do $55,
- linuxový operační systém,
- rozšiřující vstupy a výstupy s rozhraním SPI,
- existující síťové rozhraní (Ethernet/Wi-Fi).

Dle výše zmíněných kritérií bylo vybráno několik představitelů, jejich vzájemné porovnání je v Tabulce 3.1. Konkrétně jsou to **Raspberry Pi (model B)**, **BeagleBone Black**, **CubieBoard**, **A10-OLinuXino-LIME** a **pcDuino2**.

	Raspberry Pi	BeagleBone Black	CubieBoard	A10-OLinuXino	pcDuino2
CPU	ARM11	ARM Cortex A8	ARM Cortex A8	ARM Cortex A8	ARM Cortex A8
takt CPU	700 MHz	1 GHz	1 GHz	1 GHz	1 GHz
RAM	512 MB	512 MB	1 GB	1 GB	1 GB
USB	2	2	2	2	0
Ethernet	✓	✓	✓	✓	✓
HDMI	✓	✓	✓	✓	✓
SPI	✓	✓	✓	✓	✓
SATA	✗	✗	✓	✓	✗
cena	$35	$54,95	$49	$36,54[1]	$49,95

Tabulka 3.1: Porovnání linuxových vývojových desek dle jejich parametrů. Informace byly čerpány z [14], [6] a [17]. Aktualizovány byly ceny desek, které byly zjišteny z domovských stránek výrobce (byly-li dostupné).

Pro výslednou realizace byla zvolena platforma OLinuXino, konkrétně model **A10-OLinuXino-LIME**, jelikož je jedna z nejlevnějších ze všech porovnávaných platforem, avšak nabízí srovnatelné či lepší vlastnosti než konkurence.

3.2 Platforma OLinuXino

OLinuXino je open-source, open-hardware[2] vývojová platforma pro vestavěné systémy s možností operačního systému Linux pro procesory ARM. Běžné použití této platformy je například jako řídicí člen pro 3D tiskárny, nízkonákladové PLC, či pro domácí automatizaci [20].

3.2.1 Vývojové desky

OLinuXino nabízí několik druhů svých vývojových desek, vždy rozdělené do kategorií dle procesoru, který je srdcem dané desky. Společnost Olimex postupně vyvíjela své produkty dle následujícího pořadí a s následujícími vlastnostmi [19]:

- **iMX233** – počáteční model platformy, dostatečný výkon pro Linux, procesor řady ARM9 na frekvenci 454 MHz a pamětí 64 MB,
- **A13** – nástupce iMX233, procesor řady Cortex-A8 na frekvenci 1 GHz s pamětí 512 MB a jako vstupy a výstupy obsahuje: 4 USB porty, USB-OTG, vstup/výstup pro audio, SD kartu, VGA port, ovládací tlačítka, konektor se 72 vstup/výstupními vývody a konektor pro připojení externího LCD displeje,
- **A10** – podobně jako A13 má procesor řady Cortex-A8 1 GHz, ale dokáže adresovat až 2 GB, má navíc SATA, HDMI a VGA konektor, 100 MBit Ethernet a více vstup/-výstupních GPIO vývodů,
- **A10S** – novější verze A10, ale redukovaná – jen s 512 MB RAM, 1 USB port, bez SATA konektoru aj.,
- **A20** – zatím poslední inovací vývoje desek OLinuXino bylo nasazení dvoujádrového procesoru řady Cortex-A7 do desek kategorie A20.

Kromě samotných vývojových desek Olimex nabízí příslušenství ke svým deskám, jako jsou paměťové karty, baterie, LCD displeje včetně dotykové vrstvy a další.

3.2.2 A10-OLinuXino-LIME

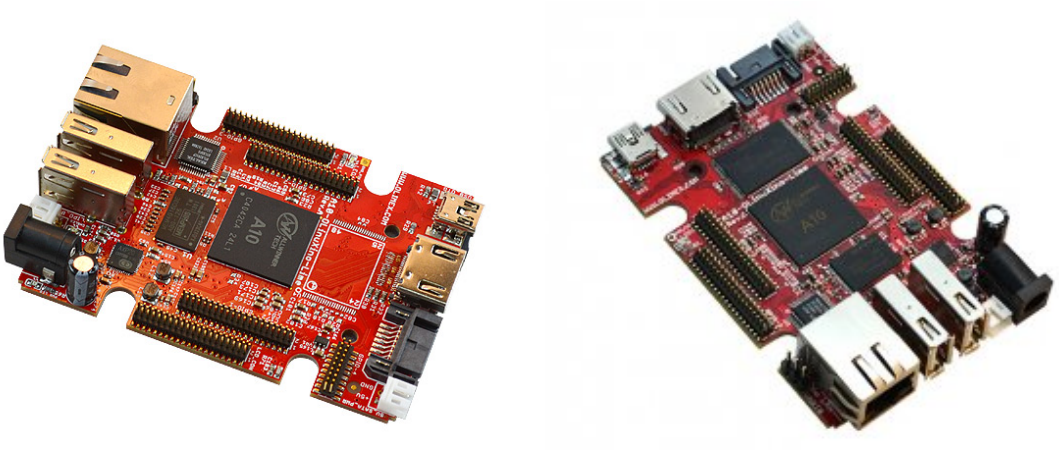
Jak již bylo zmíněno, řada A10 má ve svých útrobách procesor Cortex-A8 s taktem 1 GHz, obsahuje 512 MB DDR3 RAM paměti, dále umožňuje připojit periferie přes své konektory SATA, HDMI (s podporou FullHD rozlišení), 2× USB port + 1 USB-OTG, 100 Mbit Ethernet a konektor pro microSD paměťovou kartu. Dále pak umožňuje připojení LiPo baterie s možností jejího nabíjení, připojení externího Olimex LCD displeje a 160 GPIO pinů pro

[1] Oficiální cena je 30 €. Přepočet byl proveden ke dni 30. 12. 2014 při kurzu 1,218 $/€.

[2] Open-source a open-hardware dohromady vyjadřují souhlas tvůrce s použitím všech zdrojových kódů, včetně schématů plošných spojů k výrobcově tvorbě a umožňuje svobodně tyto informace využít či je použít k vlastní výrobě, pokud je v dokumentaci/webových stránkách uveden odkaz na původního autora [19].

ovládání externích zařízení. Ladění a vývoj na této desce umožňuje DEBUG-UART konektor pro konzolové ladění přes USB kabel, SW ovládaná LED dioda, 3 tlačítka včetně resetovacího tlačítka a 2 KB EEPROM paměti pro uložení MAC adresy zařízení a další [17].

V tomto složení stojí vývojová deska na oficiálních stránkách výrobce 30 €[3]. Existuje rozšířená verze této desky, A10-OLinuXino-LIME-4GB, která je o 10 € dražší a má navíc 4 GB NAND paměti pro rozšíření dostupného úložiště, s kterým může uživatel pracovat [17].



Obrázek 3.1: Vývojová deska A10-OLinuXino-LIME. Vlevo v základní variantě, vpravo pak s osazenou 4 GB NAND pamětí. Fotografie převzaty z [17] resp. z [18].

3.2.3 Linux pro OLinuXino

Základem OS je projekt open-source SW komunity **linux-sunxi**, která vyvíjí OS pro procesory od společnosti *Allwinner Technology*. Sunxi reprezentuje rodinu ARM procesorů od zmíněné společnosti, jejíž součástí je i procesor ze série A10, který je na vývojové desce A10-OLinuXino-LIME. Díky velmi aktivní komunitě je dostupná podpora a balíčky nástrojů pro velmi širokou škálu zařízení, které nabízí pod licencí GPLv2, ale SDK, které poskytl Allwinner pro tvorbu OS, tuto licenci v několika ohledech nedodržuje. Situace se však s postupujícím časem zlepšuje [25].

3.3 Nástroje pro vývoj

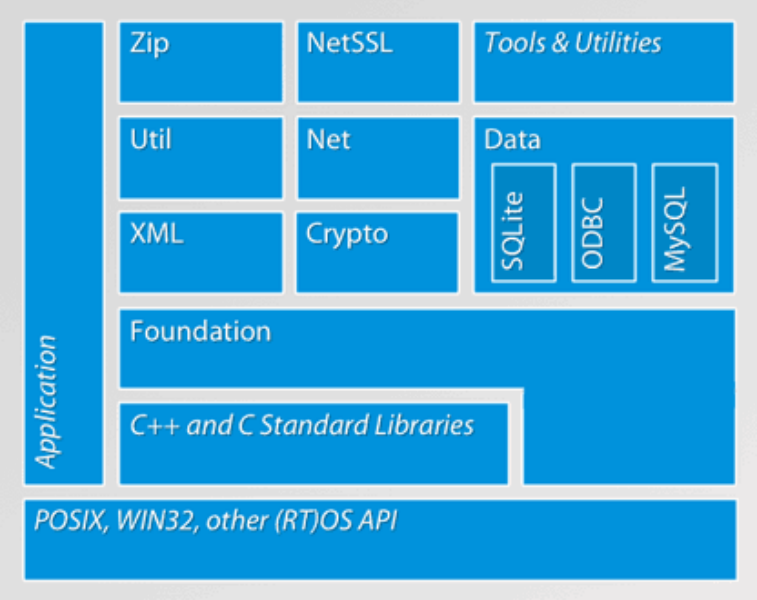
Jelikož je na zvolené vývojové desce procesor architektury ARM, je potřeba přizpůsobit vývoj pro tuto platformu. Z tohoto důvodu bylo nutné vytvořit křížový překladač, který slouží pro překlad programu pro (často) jinou platformu, než na které je vyvíjen. Ten je získán díky překladu OS pro vývojovou desku (viz výše) překladovým systémem **OpenEmbedded**, který je používán v rámci projektu IoT a slouží nejen k vytvoření SDK pro křížový překlad (včetně potřebných hlavičkových souborů, knihoven aj.), ale i k vytvoření obrazu systému se všemi potřebnými balíčky, systémovými knihovnami, zavaděčem atd.

[3] Stav k 21. 12. 2014.

3.3.1 Knihovna POCO

Součástí práce bylo prozkoumat knihovnu **POCO** a její možnosti. Ve skutečnosti se jedná o komplexní balík open-source knihoven pro jazyk C++ usnadňující práci v oblasti sítí, zpracování konfiguračních souborů, logování, (multi)vláknových aplikací a spoustu dalších utilit pro programování v jazyce C++ na různých platformách. Jejím zakladatelem je *Günter Obiltschnig* a tato knihovna je od založení v roce 2004 dále rozšiřována komunitou vývojářů C++. Kromě volně dostupné verze existuje i placená verze knihovny s rozšířenými možnostmi. Záštitu nad tímto projektem zprostředkovává společnost **Applied Informatics** [2].

Na obrázku 3.2 je vyobrazen přehled modulů knihovny POCO. Celý výčet jejich možností je dostupný na [3], stejně jako podporované platformy. Knihovna je dostupná pod licencí *The Boost Software License 1.0*, která umožňuje komukoliv využívat, upravovat či vytvořit kopii SW, pokud je přidáno znění licence k danému SW (netýká se programů, které jsou dále distribuovány jako strojový spustitelný objektový soubor).



Obrázek 3.2: Přehled oblastí knihovny POCO, které v sobě integruje. Obrázek převzat z [3].

Možnost využití knihovny POCO jak pro PC platformu, tak i pro vestavěné systémy s procesory ARM činí z této knihovny mocný nástroj pro C++ aplikace. Této skutečnosti může být využito v práci např. pro testování aplikace AdaApp jak na cílové platformě OLinuXino, tak i na PC a sloužit tak jako náhrada HW uzlu brány inteligentní domácnosti.

Pro práci s POCO byla tvůrci vytvořena dokumentace všech tříd, které se zde vyskytují, včetně jejich funkcí a vlastních datových typů. Dále se zde nachází ke každé oblasti prezentace, kde jsou popsány možnosti jednotlivých funkcí/objektů a jednoduché příklady přímo ve zdrojovém kódu C++ s použitím těchto funkcí. Díky těmto prezentacím pak uživateli nabízí základní kostru svého programu s užitím požadovaných objektů či funkcí, spolu s popisem funkce programu.

Celá knihovna je rozdělena na několik dílčích částí dle oblasti, kterou zastřešují [4] :

- **Foundation Library** – jádro POCO, obsahuje nejpoužívanější funkce, datové typy apod.,

- **XML Library** – obsahuje nástroje pro práci s XML formátem – zpracování, generování, aj.,
- **Util Library** – název lehce zavádějící, obsahuje framework pro vytváření konzolových a serverových aplikací, zpracování parametrů příkazové řádky a spol.,
- **Net Library** – usnadňuje práci při vytváření aplikací, které využívají ke komunikaci síť – podporuje TCP/UDP/ICMP/raw sokety, tvorbu HTTP serverů aj.,
- **NetSSL Library** – rozšíření **Net Library**, umožňuje používat zabezpečené připojení založené na SSL a TLS,
- **Data Library** – obsahuje nástroje pro práci s SQL databázemi,
- **Crypto Library** – často použitá spolu s **NetSSL Library**, umožňuje šifrovat spojení, práci s X509 certifikáty apod.,
- **Zip Library** – umožňuje vytvářet, editovat a extrahovat Zip archívy.

Díky tomuto rozdělení není potřeba při překladu linkovat celou POCO knihovnu jako celek, nýbrž jen ty části, v nichž se nachází námi požadovaná funkce či třída.

3.3.2 Meziprocesová komunikace

Pro komunikaci mezi procesy (*IPC – Inter-Process Communication*) je možné využít jednoho z následujících komunikačních prostředků [12][15]:

- **signály** – zasílání (číselných) signálů mezi procesy (např. *SIGINT*, *SIGKILL*),
- **roury** – nejstarší komunikační prostředek v Unixu, lze použít jen v případě, že mají procesy společného předchůdce, data putují jen jedním směrem,
- **zprávy** – 2 a více procesů může komunikovat pomocí zpráv, které se ukládají do fronty a mohou být těmito procesy čteny či zapisovány,
- **sdílená paměť** – společný ukazatel do fyzické paměti umožňuje číst či zapisovat data (=komunikovat) více procesům,
- **sockety** – komunikují obousměrně, mohou být různých druhů (SOCK_STREAM, SOCK_DGRAM, RAW aj.),
- **RPC** – volání vzdálených procedur.

Jako pokročilejší nástroje pro komunikaci procesů lze využít například sběrnicový systém **D-Bus**, který umožňuje posílat zprávy mezi procesy, spouštět aplikace a démony na vyžádání, informovat o stavu jiných procesů či případně reagovat na připojení nového HW zařízení apod. [22].

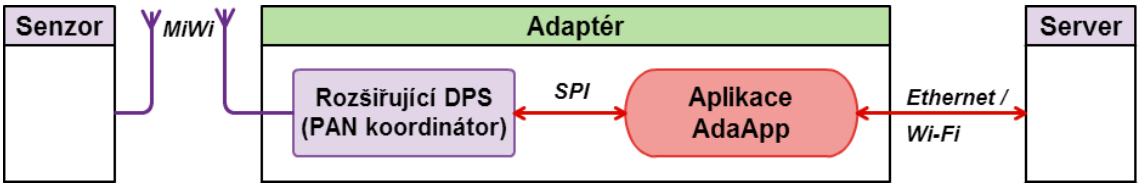
Jiným představitelem pokročilejších nástrojů je **Mosquitto**, což je open-source zprostředkovatel MQTT[4] zpráv mezi zařízeními (například mezi mobilním telefonem a vestavěným systémem). Je proto vhodný k použití pro domácí automatizaci a monitorování, nicméně je ho možné využít díky jeho vlastnostím i pro komunikaci mezi procesy [1].

[4]MQTT je *machine-to-machine/internet of Things* komunikační protokol, který je navržen jako extrémně odlehčený protokol pro distribuci zpráv mezi zařízeními. Informace čerpány z `http://mqtt.org/`

Kapitola 4

Návrh brány inteligentní domácnosti

Dle výše uvedených poznatků je vytvořen návrh pro budoucí aplikaci AdaApp. Tato aplikace bude spouštěna na adaptéru a bude komunikovat s rozšiřující deskou a okolním světem, nejčastěji tedy se serverem, kam se budou odesílat všechny zprávy s údaji ze senzorů a naopak přijímat řídicí zprávy ze serveru. Tato architektura je znázorněna na obrázku 4.1. Níže je problematika AdaApp rozebrána hlouběji, kde budou popsány náležitosti jak samotné aplikace, tak i modulu virtuálních senzorů.



Obrázek 4.1: Existující architektura adaptéru, umístění AdaApp a komunikace jednotlivých komponent s okolím.

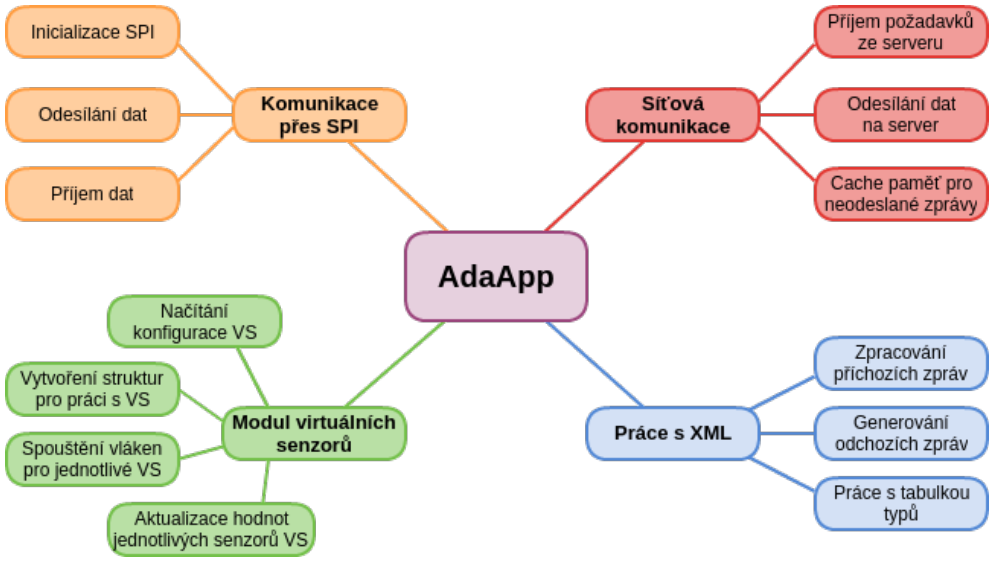
4.1 Návrh architektury AdaApp

Pro návrh aplikace je potřeba nejprve předem určit, co všechno od ní bude vyžadováno, jaké komunikační kanály bude používat a jestli se bude konfigurace aplikace či celého adaptéru obecně lišit. Toto rozvržení je zobrazeno na obrázku 4.2. Abychom byli schopni znovupoužít existující procedury např. pro práci se senzory, je vhodné navrhnout klíčové komponenty jako moduly, které bude možné pro každou instanci zapnout/vypnout dle potřeby. Existence nezbytných komponent, jako např. síťová komunikace, bude brána jako jádro aplikace, které nelze vypnout, jelikož tato komunikace bude součástí každé instance brány inteligentní domácnosti.

Cílem návrhu je tedy postavit architekturu AdaApp tak, aby bylo možné si zvolit, které komponenty pro danou konfiguraci mají být spuštěny a které nebudou využity, tudíž budou ve funkci *zásuvných modulů*. Toto pak umožní dle konfigurace měnit složení modulů v AdaApp pro různé instance, dále pak jednoduchou aktivaci požadovaného modulu při změně požadavků a také bude tato architektura přístupná k rozšiřování AdaApp dalšími moduly.

Dále by také měla mít aplikace možnost nastavení jejího chování bez nutnosti znovu překládat zdrojový kód. Toto bude dosaženo s využitím knihovny POCO a jejími utilitami pro práci s konfiguračními soubory. Tyto znalosti budou využity taktéž pro zpracování konfigurace virtuálních senzorů.

K dalším komponentám, které nejsou klíčové pro funkci AdaApp, ale umožňovaly by uživateli/vyvojáři lépe kontrolovat běh programu apod., patří například nástroje pro logování informací o běhu aplikace a chyb vzniklých za běhu aplikace. K tomuto účelu může být využito například nástrojů z knihovny POCO, konkrétně třídy **Logger**, která umožňuje vytvářet logovací kanály, které mohou být ukládány do souborů, vypisovány na standardní výstup apod. Taktéž umí vytvářet rotace logovacích souborů tak, aby nepřekročily definovanou velikost či časovou trvanlivost.



Obrázek 4.2: Návrh architektury AdaApp aplikace pro adaptér a určení základních operací pro každou komponentu zvlášť.

Další pomocnou komponentou by mohl být například modul, který by data, která se ze senzorů přijímají, poskytoval externím aplikacím. Taktéž by tyto data mohla jiná komponenta zobrazovat na externím monitoru, připojenému přes HDMI port.

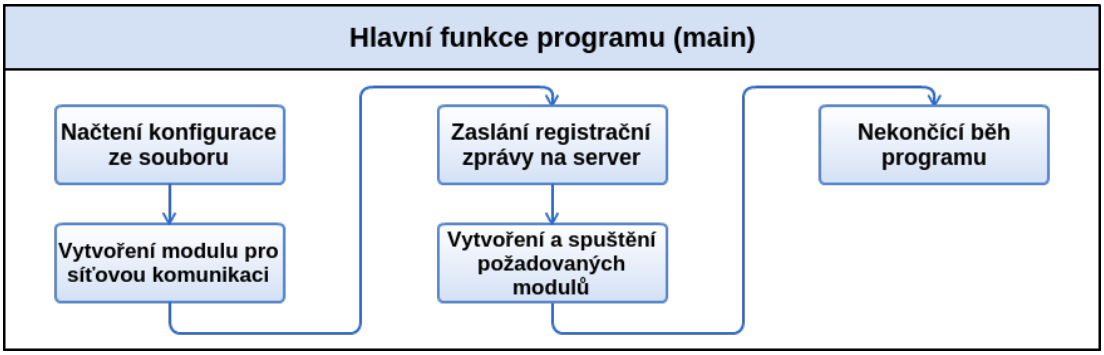
Níže jsou popsány požadavky, které by měly jednotlivé oblasti splňovat. Komplexní shrnutí a návrh je pak nastíněn v podkapitole 4.3

4.1.1 Hlavní funkce programu

Jelikož je cílovým programovacím jazykem C++, je hlavní funkcí myšlena funkce **main()**. Hlavní funkce by měla umět jednak provést inicializaci všech komponent aplikace, jednak spouštění komponent v závislosti na konfiguraci AdaApp a v neposlední řadě umět reagovat a zotavit se z chybových stavů, které mohou během činnosti aplikace nastat. Zjednodušené schéma posloupnosti vykonaných akcí hlavní funkcí je na obrázku 4.3.

4.1.2 Rozhraní pro komunikaci s rozšiřující DPS

Jelikož je komunikační rozhraní mezi vývojovou a rozšiřující deskou SPI, je potřeba pro něj mít v systému zpřístupněny příslušné balíky a nástroje pro přístup k tomuto rozhraní.



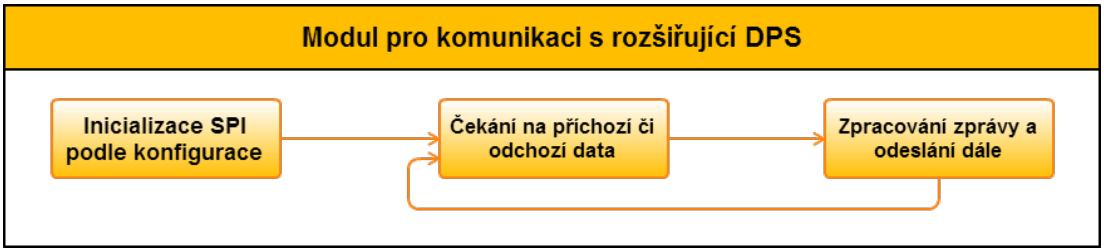
Obrázek 4.3: Zjednodušené schéma úkonů pro hlavní funkci AdaApp.

To zajišťuje jeden z modulů operačního systému s názvem **spidev**, který umožňuje číst a odesílat data přes toto rozhraní. Pro C++ aplikace jsou díky tomuto balíku dostupné knihovny pro přístup k SPI rozhraní a jeho konfigurace. Tohoto ovladače pak bude využito v práci při návrhu a vývoji modulu pro komunikaci s rozšiřující DPS [24] .

4.1.3 Modul pro komunikaci s rozšiřující DPS

Modul pro komunikaci s rozšiřující DPS je připojen k adaptéru na GPIO porty. Dřívější verze byla připojena pomocí speciálního propojovacího kabelu, nová verze je připojena již přes 2 dvouřadé konektory přímo na desku A10-OLinuXino-LIME. Z hlediska architektury je umístění rozšiřující desky znázorněno na obrázku 4.1. Tento modul bude mít tedy za úkol komunikovat s HW (rozšiřující deskou) přes sériovou sběrnici SPI. Ta slouží nejčastěji pro sériovou komunikaci s nějakou periferií, z nichž jeden je v režimu *master* a druhý v režimu *slave*. Využívá 2 datové vodiče (MISO a MOSI[1]), hodinový signál na vodiči SPSCK a umožňuje také vybírat zařízení, se kterým chce *master* komunikovat, pomocí vodiče SS [23].

Je tedy potřeba, aby se tento modul staral o inicializaci SPI rozhraní, komunikaci přes SPI a uměl interpretovat data od HW senzorů. Dále také musí umět odesílat požadavky, které přichází ze serveru, směrem k senzorům. Pro vyšší sploholivost aplikace (a pro zajištění zotavení z chyb) by měl modul umět reagovat i na přijetí poškozené či nevalidní zprávy z rozšiřující desky. Zjednodušené schéma činnosti je znázorněno na obrázku 4.4.



Obrázek 4.4: Zjednodušený průběh činnosti modulu pro komunikaci s rozšiřující deskou.

[1]MISO = Master In Slave Out, což značí směr přenosu po tomto vodiči. Analogicky pak totéž platí pro MOSI (Master Out Slave In).

4.1.4 Modul pro síťovou komunikaci

Jednou z nejdůležitějších součástí AdaApp je jistě síťová komunikace. Zde je nutno spojení zabezpečit, aby komunikaci nemohl nikdo odposlouchávat či zneužít, dále je potřeba umět přijmout, zpracovat a interpretovat příchozí zprávy ze serveru, totéž platí pro odchozí zprávy. Při ztrátě spojení je potřeba si umět s touto situací poradit – zálohovat neodeslaná data a pokusit se je odeslat později. Dále pak bude tento modul komunikovat s dalšími moduly pro zajištění správné interpretace zpráv jejím adresátem. Zde by bylo možné využít prostředků knihovny POCO, která nabízí nástroje pro zabezpečené spojení, HTTP server pro přijímání a odesílání požadavků aj. Návrh základní funkcionality je na obrázku 4.5.

Obrázek 4.5: Návrh modulu pro síťovou komunikaci. Červené bloky značí funkce, které bude implementovat tento modul, zelené bloky bude implementovat modul pro práci s XML a fialový blok značí spolupráci s dalšími moduly.

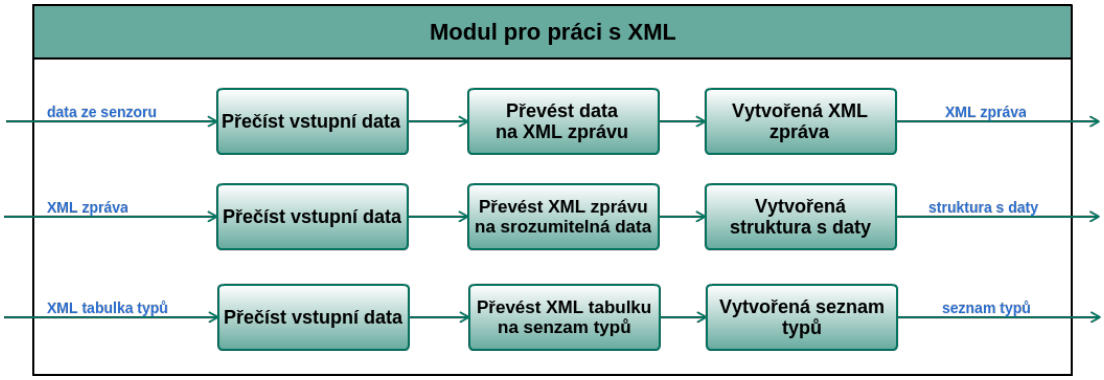
4.1.5 Modul pro práci s XML

Úzce souvisí s modulem pro síťovou komunikaci – jelikož všechny zprávy přicházejí/odcházejí ve formátu XML, je potřeba příchozí zprávy zpracovat, uložit do (programu) srozumitelné formy a interpretovat. Naopak odchozí zprávy je potřeba z této srozumitelné formy převést do XML zprávy, a tuto odeslat. Právě zmíněné konverze by měl zajišťovat modul pro XML, který zde opět může využít dostupných parserů z knihovny POCO. Přehled požadavků na funkce modulu pro práci s XML je na obrázku 4.6.

S tímto modulem souvisí tzv. **tabulka typů**, která bude uložena rovněž ve formátu XML. Jelikož existuje velké množství typů senzorů, které však podporu v aplikaci mohou získat až časem, je vhodné aplikaci navrhnout tak, aby nemusela být překládána při zajištění podpory nového typu senzoru. Řešení tohoto problému bude zajišťovat tabulka typů, o jejímž návrh bude více hovořeno v podkapitole 4.2.

4.1.6 Modul virtuálních senzorů

Jelikož je tento bod důležitost součástí AdaApp, která bude sloužit pro otestování velké části aplikace a komunikace s okolím, bude návrhu virtuálních senzorů věnována samostatná



Obrázek 4.6: Zjednodušený návrh rozhraní a vnitřní funkcionality XML modulu.

podkapitola 4.2.

4.1.7 Doplňkové moduly

Jak již bylo zmíněno, dalšími moduly je myšleno například logování zpráv a stavů aplikace tak, aby vývojáři či externím aplikacím umožnilo zjistit stav AdaApp, vypisovat vzniklé chyby či případně ověřit funkčnost jednotlivých komponent AdaApp. Mezi doplňkové moduly může v budoucnu patřit i například komunikace právě s externími aplikacemi na adaptéru, které by pak tato data mohly dále analyzovat, zpracovávat a případně i zobrazovat či poskytovat dále.

4.1.8 Další možné rozšiřující moduly

V budoucnu by mohla AdaApp obsahovat např. modul pro komunikaci s mobilním zařízením přes rozhraní Bluetooth, umět komunikovat s mobilním zařízením přes rozhraní Wi-Fi (kvůli zrychlení odezvy), zobrazovat data ze senzorů na obrazovce přes rozhraní HDMI apod.

4.2 Návrh modulu virtuální senzorové sítě

Tento modul by měl velmi věrohodně kopírovat chování reálného HW (multi-)senzoru[2]. Jelikož dostupné HW senzory nabízí (=odesílají) následující informace, stejné informace budou dostupné a konfigurovatelné i ve VS[3]:

- **RSSI** – síla přijatého signálu,
- **battery** – napětí baterií,
- **LQI** – doplňující hodnota o kvalitě spojení,
- **dvojice – typ senzoru + hodnota** – informace o měřené veličině a její hodnota, těchto dvojic může být více.

[2]Multisenzorem se rozumí jedna DPS, na které je vícero senzorů stejné či různé měřené veličiny. Lze tak mít například 2 senzory (teplota, vlhkost) na jednom zařízení.

[3]Dále se v textu budou označovat zkratkou VS virtuální senzory.

Všechna prvotní spojení mezi adaptérem a senzorem vyvolávají příkazy ze serveru. Tyto příkazy umožňují **spárovat nový senzor** a **upravit čas odeslání dalších hodnot ze senzorů**.

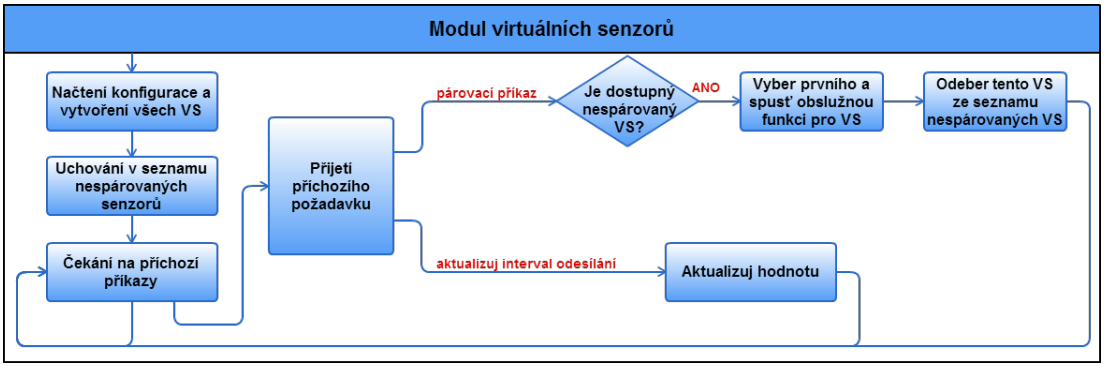
Jelikož VS neměří žádnou reálnou veličinu, je také potřeba jim nějakou hodnotu přiřadit. Aby tento modul sloužil i pro testování celého systému, je vhodné implementovat generování hodnot z virtuálních senzorů náhodně, či dle nějakého průběhu v určitém rozsahu a umožnit tak uživateli nastavit způsob generování hodnot z VS.

Jelikož je pravděpodobné, že počet typů veličin HW senzorů časem poroste, taktéž VS musí být schopny tyto typy používat a generovat pro ně hodnoty. Aby nebylo potřeba pro každý nový typ senzoru aktualizovat AdaApp, je potřeba navrhnout systém, který umožní dynamickou práci s rozrůstajícím se portfoliem typů senzorů. Souhrnně je tato oblast označována jako **tabulka typů**, která je popsána v podkapitole 4.2.2.

4.2.1 Návrh virtuálního senzoru

Dle výše zmíněných vlastností reálných senzorů je tedy potřeba, aby chování virtuálního senzoru vykazovalo požadované chování na příchozí akci. VS tedy musí umět:

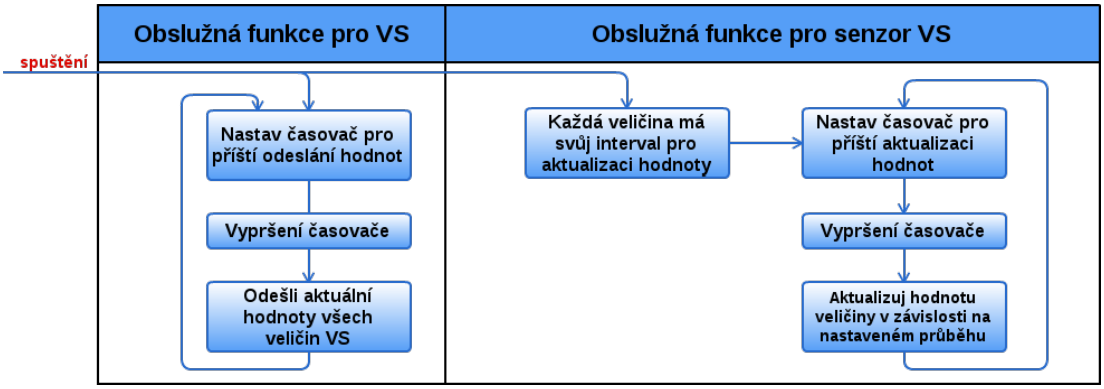
- **vzniknout** – načíst svoji konfiguraci a vytvořit objekt/vlákno, ve kterém bude existovat,
- **spárovat se** – po příchozím požadavku na párování musí začít odesílat hodnoty na server,
- **aktualizovat *refresh-time*** – uživatel může dynamicky měnit interval mezi odesláním zpráv (změna provedena přes mobilní zařízení),
- **aktualizovat svou hodnotu** – měnit hodnoty veličiny dle požadovaného průběhu a v požadovaném intervalu z konfigurace.



Obrázek 4.7: Zjednodušený návrh vnitřní funkcionality modulu pro VS.

Jelikož může mít každý VS více veličin, které odesílá, je potřeba zajistit, aby se každá z veličin chovala nezávisle na ostatních (čas aktualizace, průběh atd.). Může také nastat situace, kdy bude použito obou modulů pro komunikaci se senzory – HW senzory i VS. V tom případě je potřeba vyřešit otázku párování – který modul bude mít přednost, kdo bude řešit rozhodování apod. Jelikož by toto rozhodování mělo příslušet modulu, který bude těmto dvěma nadřazený, bylo by vhodné tuto logiku přenést do některého z výše popsaných modulů (například modulu pro síťovou komunikaci). Zjednodušený návrh tohoto modulu

je na obrázku 4.7. Nástin toho, jak by mohl modul VS pracovat s časovými intervaly pro odesílání hodnot a aktualizací hodnot senzorů je na obrázku 4.8.



Obrázek 4.8: Návrh doplňující funkcionality VS pro zajištění aktualizace hodnot a odesílání hodnot ve stanovených intervalech.

4.2.2 Návrh tabulky typů

Tabulka typů by měla obsahovat především číselnou informaci o typu snímané veličiny, kterou bude celý systém interpretovat stejně. Kromě toho by měla obsahovat doplňující informace například o názvu této veličiny, jednotky, datový typ hodnoty veličiny, počet bajtů, na kterých bude hodnota uchovávána (platí pro komunikaci přes SPI s rozšiřující deskou), a případně i výraz, kterým je potřeba hodnoty získané ze senzoru upravit tak, aby vyjadřovaly skutečnou hodnotu veličiny. Příkladem může být situace, kdy snímací část senzoru není schopná interpretovat teplotu jako desetinné číslo, a tedy svoji hodnotu násobí koeficientem 100, což přijímací strana musí dopředu vědět a upravit tuto hodnotu na reálnou hodnotu teploty. V neposlední řadě by samotná tabulka měla obsahovat informace o její verzi, časovém razítku apod.

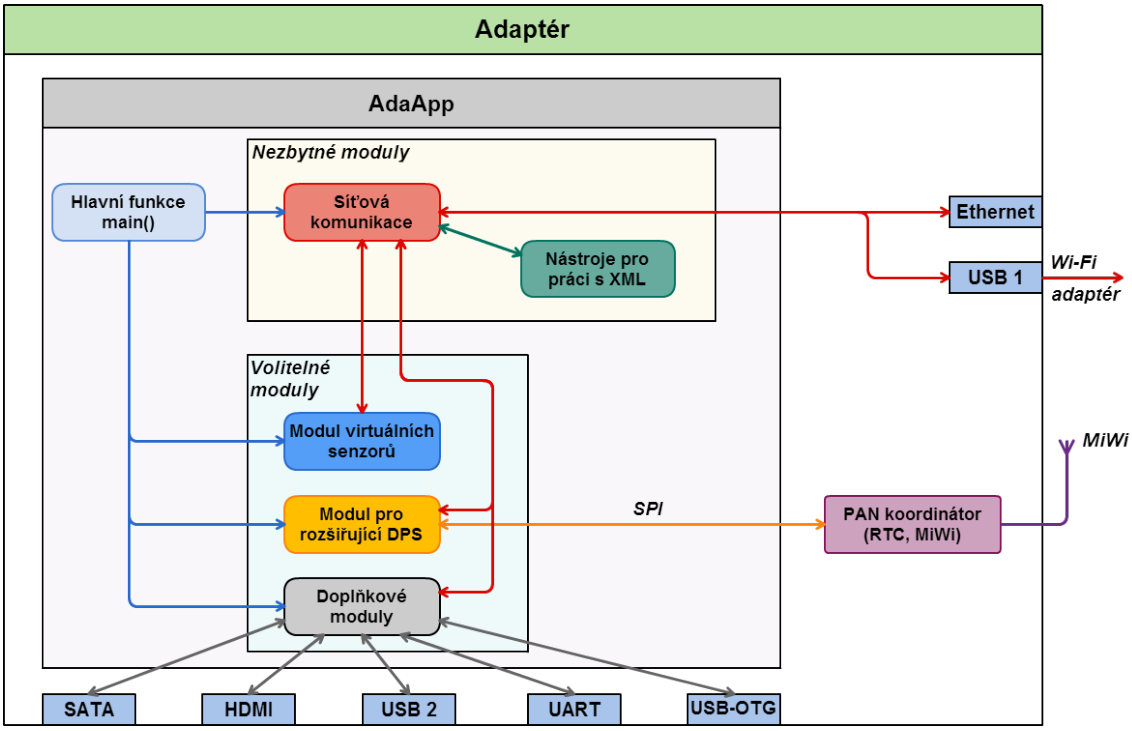
Pro implementaci tabulky typů byl vybrán formát XML, jelikož umožňuje uchovat všechny výše zmíněné informace ve své struktuře, jedná se o velmi rozšířený formát a jeho forma bude stejná pro celý systém. Příklad toho, jak by tabulka typů mohla vypadat, je znázorněn na obrázku 4.9.

```
<?xml version="1.0" encoding="UTF-8"?>
<types_table version="1.0" timestamp="1419973722">
  <sensor_type name="temperature" type_num="0" unit="°C" modifier="/100" />
  <sensor_type name="pressure" type_num="1" unit="hPa" size="2B" />
  <sensor_type name="humidity" type_num="2" unit="%" />
</types_table>
```

Obrázek 4.9: Příklad XML tabulky typů.

4.3 Shrnutí návrhu

Dle výše uvedených informací byl vytvořen finální návrh celé AdaApp, který může být v reálné implementaci upraven, nicméně dle získaných teoretických znalostí je kompletní návrh znázorněn na obrázku 4.10, včetně návrhu komunikace mezi moduly. Celý program využívá komponent knihovny POCO, je psán v C++ a pro vývoj je použita platforma PC, z které je program překládán křížovým překladem na desku A10-OLinuXino-LIME.



Obrázek 4.10: Kompletní návrh celé AdaApp včetně komunikace mezi moduly.

Kapitola 5

Implementace navrženého řešení

Z výše získaných poznatků a předběžného návrhu z kapitoly 4 bylo nutné rozvrhnout si plán implementace aplikace. Inspirací pro implementaci mi bylo řešení v jazyce Python[1], které využívalo rozhraní **spidev** pro komunikaci se sběrnicí SPI a získanou posloupnost bajtů odesílalo na server. Tento předchůdce aplikace AdaApp bylo tedy nutné přepsat do jazyka C++, implementovat konverzi zpráv z pole bajtů na formát XML, vytvořit modulární aplikaci a rozšířit ji o příslušné moduly. Taktéž bylo potřeba do AdaApp zavést meziprocesovou komunikaci pro komunikaci s dalšími aplikacemi na adaptéru (např. AdaMan aj.).

5.1 Plán vývoje

Plán postupu při vývoji aplikace AdaApp, seřazený sestupně dle priorit úkolů před začátkem řešení, byl sestaven následovně:

- **Příprava před samotnou implementací programu**
  - seznámit se s architekturou systému inteligentní domácnosti,
  - seznámit se s vývojovou platformou a překladovým systémem,
  - vytvořit základní aplikaci v C++ pro platformu ARM.
- **Vytvořit modul pro zpracovávání XML zpráv a XML dat obecně**
  - nastudovat chování a práci s modulem *POCO::XML*,
  - implementovat funkce pro vytvoření XML zprávy dle definovaného formátu z dat ze senzorů,
  - připravit funkce pro zpracování XML zpráv ve formátu dle definovaného protokolu,
  - testování vytvořeného modulu před implementací modulu pro komunikaci se serverem.
- **Vytvořit modul virtuálních senzorů**
  - vytvoření prvotního návrhu tohoto modulu, s orientací na jednoduchou změnu konfigurace a dodržení modularity systému,

[1] Toto řešení vzniklo jako pilotní verze aplikace AdaApp ve spolupráci Bc. Martina Douděry, Ing. Tomáše Novotného a Ing. Josefa Hájka.

  - seznámit se s modulem *POCO::Thread* pro implementaci VS jako samostatných vláken,
  - testování modulu z hlediska generování hodnot dle definovaného protokolu, konverze do XML formátu a příprava pro odesílání hodnot na server,

- **Vytvořit modul pro komunikaci se serverem**
  - nastudovat protokol zpráv využívaných v projektu inteligentní domácnosti,
  - seznámit se s modulem *POCO::Net*, který nabízí prostředky pro snadnější práci se sokety apod.,
  - implementovat modul pro síťovou komunikaci, který bude využívat modul pro práci s XML,
  - testování základní komunikace mezi serverem a AdaApp, včetně zpracování a generování zpráv z/do XML formátu.

- **Vytvořit modul pro rozhraní SPI**
  - seznámit se s ovladačem **spidev** pro rozhraní SPI,
  - implementovat vyčítání a zápis na SPI sběrnici přes zmíněný ovladač,
  - zpracovávat informace ze sběrnice a ukládat je do datového typu, který umožní jednodušší práci s nimi.

Výše uvedený výčet byl prvotním plánem, který bylo nutné implementovat, aby mohla AdaApp plnit svou funkci a být schopná komunikovat s okolním prostředím. V průběhu práce na AdaApp došlo vlivem vylepšování aplikace, vývoje a měnících se potřeb celého systému inteligentní domácnosti ke změnám, které budou zmíněny při podrobném popisu vytvořených komponent. S vývojem rozšiřující desky (označován pracovně jako tzv. *PAN koordinátor*) došlo také k vytvoření dalších modulů pro ovládání nových komponent na desce, případně byly doplněny funkce do stávajících modulů.

5.2 Příprava prostředí pro vývoj

Vyvíjená aplikace byla v počátcích vyvíjena výhradně pro platformu ARM tak, aby bylo možné binární soubor s aplikací spouštět na vývojové desce A10-OLinuXino-LIME. Jak již bylo zmíněno v kapitole 3.3, pro tento překlad je zapotřebí křížového překladače – tedy do překladového systému natáhnout konfiguraci pro překlad pro jinou platformu. Tato konfigurace (SDK) je vytvořena pomocí překladového systému **OpenEmbedded**.

Zdrojový kód byl verzován pomocí systému **Git**, který usnadnil sdílení zdrojových souborů a jejich verzování. Vývojové IDE bylo zvoleno **Netbeans IDE 8.0.1** s podporou zvýrazňování syntaxe pro C++ projekty, spolu s podporou pro verzovací systém **Git**.

Pro komunikaci se samotným vývojovým kitem bylo využito více způsobů spojení – jedním z nich je připojení přes síť pomocí ssh tunelu (známe-li IP adresu zařízení v síti), případně připojení přes UART pomocí převodníku *USB-UART*. Pro druhý způsob je potřeba např. utility **kermit**, kterou je potřeba nakonfigurovat s následujícími parametry:

```
set line /dev/ttyUSB0
set speed 115200
set flow-control none
set carrier-watch off
robust
```

5.3 Vývoj aplikace AdaApp

Dle výše uvedeného plánu a s připraveným prostředím pro vývoj byla započata práce na jednoduchém prototypu aplikace, přeložitelným pro cílovou platformu. Postupně byla vytvořena aplikace, která komunikuje se serverem pomocí XML zpráv, tvořených z dat z VS či HW senzorů z rozhraní SPI. Tuto vývojovou verzi bylo třeba rozšířit o doplňkové moduly a zajistit, aby byla aplikace schopná fungovat stabilně (bez pádu či nutnosti zásahu do programu) při nasazení do domácností.

Iterativním vývojem byly vyvíjené další moduly – prvním z nich byl modul XML, který slouží pro zpracovávání příchozích i odchozích zpráv z, resp. na server. Po předložení struktury s daty ze senzoru je dle definovaného protokolu vygenerován příslušný XML řetězec, který je odeslán na server, resp. z příchozí zprávy ze serveru je tato struktura naplněna přijatými informacemi. Kromě toho slouží tento modul pro zpracovávání tabulky typů, která bude popsána níže.

Dalším vyvíjeným modulem byl modul VS, jehož návrh je popsán v kapitole 4.2. Tento modul umožňuje snadnou konfiguraci VS pro potřeby testování celého systému inteligentní domácnosti. Testována byla samotná AdaApp, která je modulem VS otestována na konverzi dat z VS do XML zprávy a odeslána na server, párování VS aj., přes server, který je díky VS testován na zpracování XML zpráv od VS, které mohou reprezentovat dosud neznámé nebo nevyrobené typy HW senzorů (např. tlak, vlhkost, hluk atd.). Taktéž je možné testovat mobilní aplikaci, která díky VS může být testována na netradiční typy senzorů, případně na neobvyklé průběhy hodnot ze senzorů apod.

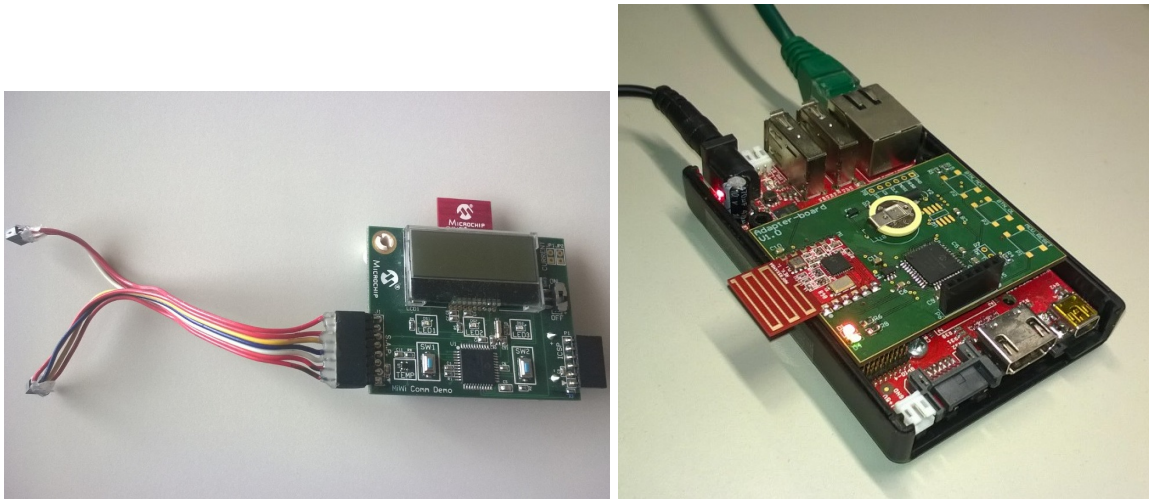
Po tomto modulu byla nasazena zabezpečená komunikace přes VPN pomocí utility **openVPN**, která umožňuje jednak zabezpečit komunikaci pomocí klientských certifikátů podepsaných certifikační autoritou, jednak umožňuje řešit problém s umístěním brány inteligentní domácnosti v domácí síti uživatele (při umístění adaptéru za NAT není možné připojit se na adaptér ze strany serveru). Toto řešení však vytváří zabezpečenou rouru mezi serverem a adaptérem, což vytváří zranitelné/zneužitelné místo pro potenciálního útočníka, který by chtěl napadnou serverovou část, či podvrhnout data apod.

Celý tento průběh se nesl v duchu předchozího návrhu z kapitoly 4, konkrétně obrázku 4.2, až na vynechání ukládání neodeslaných zpráv, což ve výsledku nebyl triviální úkol a bylo potřeba se v tuto chvíli soustředit na funkční základní komunikaci AdaApp se svým okolím. Diagram implementovaných funkcí tohoto prototypu je tedy podobný jako na obrázku 4.2, ale chybí zde podpora pro vyrovnávací paměť pro neodeslané zprávy.

Modul pro komunikaci s rozšiřující DPS je stěžejním bodem celé aplikace, jelikož zajišťuje komunikaci mezi senzory a AdaApp. Jelikož v této části došlo k velkým změnám v oblasti rozšiřující desky, bylo nutné tomuto vývoji přizpůsobovat i aplikaci. To však díky konfiguraci modulu nezapříčinilo žádné problémy v nahrazení. Příklad vývoje rozšiřující desky je znázorněn na obrázku 5.1.

V tomto okamžiku už bylo možné mít adaptéry na neveřejných IP adresách, tedy umístěné potenciálně kdekoli. Díky této skutečnosti a díky tomu, že aplikace byla navržena modulárně a byla přizpůsobovaná i pro PC platformu, bylo možné AdaApp testovat na domácích sítích jak na vývojovém kitu OLinuXino, tak i na běžném linuxovém stroji. Díky modulu VS pak bylo možné nasadit AdaApp i přímo na server a po vygenerování VS a jejich spárování mohl vzniknout stálý adaptér pro testování systému a hlavně sběru dat pro následné analýzy apod.

Pro načítání konfigurace byly vytvořeny konfigurační soubory s příponou **.ini**, jejichž



Obrázek 5.1: Vývoj HW rozšiřující desky. Vlevo je verze desky, která byla pro AdaApp používána mezi 6/2014-12/2014. Vpravo pak aktualizovaná destička, využívaná pro vývoj v roce 2015. V současné době (5/2015) je již ve výrobě další verze této rozšiřující desky.

zpracování podporuje knihovna POCO. Díky tomu pak bylo možné jednoduše měnit konfiguraci aplikace změnou atributů v konfiguračních souborech, což značně usnadňovalo a urychlovalo vývoj aplikace i její testování. Ve stejném období vznikly i logovací soubory, resp. byly implementovány nástroje pro logování informací z jednotlivých modulů pomocí modulu **POCO::Logger**. Díky konfiguračním souborům pak existuje možnost zapnout/vypnout logování informací jednotlivým modulům a sledovat tak pouze vyžádané informace, které AdaApp generuje.

V této fázi bylo nutné řešit případy, kdy vznikne nějaký chybový stav adaptéru – hlavně tedy stav aplikace při výpadku síťového spojení. Při této situaci je nutné jednak řešit znovuobnovení spojení se serverem a jednak ukládat hodnoty přicházející z HW/virtuálních senzorů, aby nedošlo ke ztrátě těchto důležitých dat. Pro tento účel byla aplikace upravena, konkrétně architektura odesílání dat. Vznikl modul s názvem **Aggregator**, který působí jako kritická sekce pro průchod veškerých dat z/do adaptéru. Díky tomu je možné v tomto modulu provádět veškeré potřebné operace se zprávami, jako je například jejich ukládání, distribuce aj. V rámci toho modulu byla vytvořena vyrovnávací paměť pro uchovávání zpráv, které se nepodařilo odeslat na server. Tyto zprávy se ve vyrovnávací paměti řadí podle jejich časových razítek a priority a modul se pokouší v pravidelných intervalech tyto data z paměti znovu odesílat na server (s cílem vyprázdnit frontu zpráv k odeslání). Tento úkol však není triviální, což bude detailněji popsáno později.

V rámci menší přestavby AdaApp pak vznikl modul **Distributor**, který rozšiřuje *Aggregator* o možnost ukládat data do historie (pro distribuci dalším aplikacím, jako například **AdaVis**), možnost distribuovat data na pojmenovanou rouru, přes MQTT zprávy apod. Obecně modul slouží na distribuci dat svému blízkému okolí (na adaptéru), včetně nástrojů pro konverzi senzorových zpráv na potřebné formáty či výstupy (např. do formátu CSV, XML, aj.).

V neposlední řadě byly vyvinuty testovací skripty pro ověření funkčnosti AdaApp – ať už z hlediska ověření úspěšného překladu a spuštění při aktualizaci zdrojových souborů aplikace na *Git* repozitáři, či pro ověření správné funkce odesílání dat. Tyto skripty byly nasazeny do serveru průběžné integrace **Jenkins**, který byl nasazen v projektu pro průběž-

nou kontrolu vyvíjených aplikací a podporu pro automatizovaný překlad celého systému inteligentní domácnosti.

Za zmínku stojí drobné funkce adaptéru, které nejsou pro aplikaci stěžejní, ale slouží jako přidaná hodnota pro celou AdaApp – je to snímání stisku tlačíka na adaptéru, které může mít různé reakce na různě dlouhé stisky. Dále bylo pro vývoj použito ovládání vestavěných LED diod na rozšiřující desce pro signalizaci stavů AdaApp.

5.4 Popis jednotlivých implementovaných komponent AdaApp

V rámci následující podkapitoly budou popsány jednotlivé komponenty celé brány inteligentní domácnosti, které prošly v průběhu vývoje řadou změn, které budou v podkapitolách zmíněny. Výsledná podoba aplikace je popsána v kapitole 7.

5.4.1 Překlad

Pro překlad byl vytvořen *Makefile* pro PC platformu a samostatný skript pro křížový překlad pro platformu ARM. SDK, potřebné pro křížový překlad, bylo vytvořeno vedoucím práce Ing. Tomášem Novotným, v překladovém systému OpenEmbedded (viz kapitola 5.2). Při vývoji byl proveden přechod z překládací utily **make** na **cmake** a v rámci sjednocení překladu jednolivých částí inteligentní domácnosti byl i patřičný *CMakeLists.txt* vytvořen vedoucím práce. K tomuto účelu pak vznikly ještě další pomocné skripty pro rozdělený překlad (nastavení prostředí) aplikace AdaApp pro PC nebo ARM platformu. Do zmíněných skriptů byly následně přidány direktivy pro preprocesor (jedna pro zjištění verze firmwaru aplikace z tagu v Gitu, druhá pro určení, zda-li se překlad provádí pro PC platformu či nikoli). Těchto direktiv je pak ve zdrojovém kódu využito pro specifikaci verze FW skrze celý systém, resp. pro selekci úseků zdrojového kódu, které pro PC platformu nejsou potřeba.

5.4.2 Spouštění aplikace

Jelikož je na cílové platformě linuxová distribuce s manažerem služeb **systemd**, bylo této vlastnosti využito pro návrh systémového démona pro tohoto manažera. Ten spustí skript, který v sobě obsahuje přípravu prostředí pro AdaApp, inicializuje GPIO pro SPI sběrnici (pokud ještě neproběhla) a spustí samotnou AdaApp.

5.4.3 Modul pro práci s XML

V tomto modulu jsou dle návrhu v kapitole 4.1.5 implementovány funkce pro zpracování XML zpráv, včetně tabulky typů. Tento modul je vytvořen jako třída pro zpracování dat a v závislosti na druhu operace je nutné vytvořit instanci této třídy s patřičným konstruktorem (2 hlavní typy – buď zpracování zprávy ze serveru či její generování, nebo zpracování tabulky typů). Pak je možné prostřednictvím této instance provádět zpracování či generování potřebných dat pro další zpracování.

Pro práci s XML protokolem je využito modulu *XML* z knihovny **POCO**, díky kterému není třeba implementovat vlastní zpracování řetězce v XML formátu. Formát XML zpráv, využívaných pro přenos dat z/do AdaApp, je v příloze C.

5.4.4 Modul virtuálních senzorů

Modul VS vznikl ze dvou hlavních důvodů: pro potřebu testovat odesílání dat z aplikace na server a pro ladění. Kromě modulu pro komunikaci s rozšířenou DPS (zkráceně modul SPI), který pro odesílání dat musí získat data z SPI sběrnice, bylo potřeba jiným způsobem získávat data ze senzorů a pracovat na vývoji nových podporovaných senzorů a zařízení. Pro tyto požadavky byl vytvořen modul VS, který umožňuje (uživateli) konfigurovat virtuální zařízení, které obsahuje různé typy senzorů s definovanými hodnotami či průběhy hodnot. Příklad výsledku vytvořeného konfiguračního souboru pro VS je na obrázku 5.2.

```
# povoleni modulu v aplikaci a sdileni logu
[Virtual_sensor]
enabled = true
enable_logging = true

# definovani prvniho zarizeni - obsahuje senzor teploty (0x0a) a vlhkosti (0x01)
[Sensor_1]
battery = 2500
rssi = 90
lqi = 90
type_1 = 0x0a
value_1 = linear:5->100|step:5|time:2m
offset_1 = 0
type_2 = 0x01
value_2 = triangle:25->75|step:0.1|time:5s
offset_2 = 0

# definovani druheho zarizeni - obsahuje 2 senzory typu 0x06, oddeleny offsetem
[Sensor_2]
type_1 = 0x06
offset_1 = 0
value_1 = random:10->100|time:1h
type_2 = 0x06
offset_2 = 1
value_2 = const:55
```

Obrázek 5.2: Ukázka možnosti konfigurace VS pro AdaApp. Je možné si nakonfigurovat více různých senzorových veličin pro jedno zařízení, ale také více stejných s jiným indexem (pro jednoznačnou identifikaci senzoru na zařízení).

Tento modul byl vyvíjen dle návrhu v kapitole 4.2 a jak již bylo zmíněno, umožňuje ověřit připravenost celého systému na nově podporovaná zařízení, která zatím nemají fyzickou podobu. U každého VS lze konfigurovat parametry zařízení (hodnota baterie, úrovně kvality signálu atd.) a samotné veličiny, které zařízení měří a získává z nich data pomocí jednotlivých senzorů. Lze také definovat rozpětí jejich hodnot, granularitu, průběh a periodu generování nových hodnot. Pro potřeby vývoje byly implementovány 4 řežimy průběhu veličiny:

- *const* – senzor poskytuje stanovenou (konstantní) hodnotu po celý čas její existence,
- *random* – senzor poskytuje náhodné hodnoty z definovaného intervalu a po zvolené periodě,
- *linear* – senzor poskytuje lineární průběh dle zvoleného kroku ve specifikovaném intervalu a po zvolené periodě – při dosažení stanovené cílové hodnoty z intervalu je průběh převeden na konstantní,

- *triangle* – stejný jako linární, ale při dosažení cílové hodnoty je velikost kroku převedena na opačné číslo a lineární průbeh pokračuje zpět k počáteční hodnotě z intervalu – tzv. průběh pily.

Modul VS byl s výhodou využit pro funkční testy aplikace AdaApp (popsány detailně v kapitole 6), kde díky definované konfiguraci VS a pojmenované rouře (popsána v podkapitole 5.4.7) lze zjistit, zda-li v nové verzi aplikace nevznikla chyba a neporušil se řetěz odesílání dat. Pro tuto funkcionalitu byl do modulu VS implementován mechanismus, který umožňuje znovu spárovat určitý počet VS dle přečtení této informace z volatilní paměti adaptéru. Toto řešení plyne z virtuality všech VS, které jsou uloženy pouze v paměti programu a po úvodním spuštění/restartu aplikace by musel uživatel znovu zaslat potřebný počet žádostí o párování senzoru, což je při automatizovaném testování nežádoucí.

5.4.5 Modul pro síťovou komunikaci

Tento modul se stará o veškerou síťovou komunikace z adaptéru na server a zpět. K tomu slouží definované XML zprávy (jejich syntaxe je v příloze C), jež se od sebe liší zejména svým typem příkazu, který reprezentují. Pro současný stav systému jsou implementovány tyto typy zpráv:

- *registrační zpráva* – touto zprávou z adaptéru na server konkrétní adaptér žádá o jeho registraci na serveru,
- *párování zařízení* – pokyn pro PAN koordinátor, aby na bezdrátové síti pro senzory zahájil párovací režim,
- *zpráva s daty* – zpráva obsahující data ze senzorů daného zařízení,
- *nastavení aktoru* – zpráva pro nastavení hodnot aktorů,
- *čas probuzení* – odeslání času pro zařízení, za jak dlouho se má probudit ze spánku a odeslat následující data,
- *návrat do továrního nastavení* – zpráva, která inicializuje celý adaptér do továrního nastavení,
- *smazání zařízení* – smazání záznamu o daném spárovaném zařízení v senzorické síti.

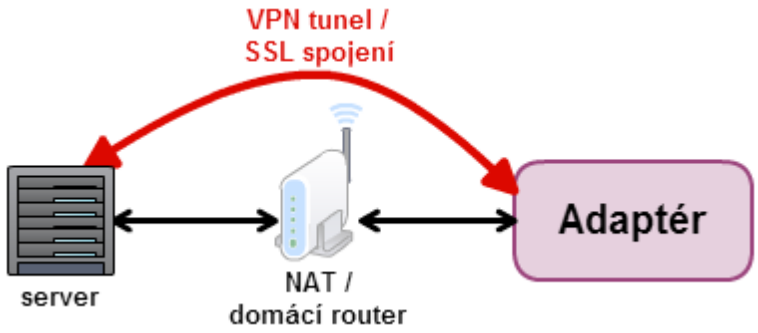
Síťová komunikace během vývoje aplikace prošla řadou změn. Z těch velkých bych zmínil 3 fáze vývoje aplikace:

- *prototyp komunikace* – prvotní verze komunikace se serverem dle definovaného protokolu zpráv a práce na zpracování potřebných typů zpráv,
- *přechod na VPN* – potřeba jak zabezpečené komunikace, tak i řešení problému, kdy je adaptér umístěn v domácí síti, tudíž není možné navázat nové spojení ze serveru na adaptér, ale pouze opačně,
- *přechod na SSL* – nahrazení VPN pomocí zabezpečené SSL komunikace.

Konečná podoba síťové komunikace tak byla rozdělena na 2 části – přijímače zpráv ze serveru a odesílatele. Toto rozdělení umožňuje mít pro příjem zpráv ze serveru separátní objekt, který bude naslouchat na síťovém rozhraní a přijímat zprávy ze serveru. Po jejich konverzi (pomocí modulu pro práci s XML) je pak tato zpráva předána k dalšímu zpracování.

Základ pro příjemce byl vytvořen z modulu *Net* knihovny *POCO*. Vlivem výše zmíněného vývoje pak bylo nutné nahradit VPN řešení, implementovaného pomocí nástroje *openVPN*, pomocí SSL komunikace. Pro ni bylo taktéž využito téhož modulu knihovny *POCO*, který v sobě obsahuje nástroje i pro práci s SSL spojením. Tuto náhradu spojení implementoval v rámci své bakalářské práce student Lukáš Köszegy, který vzal moje původní řešení a přepsal jej na nový typ komunikace pomocí SSL spojení.

Tato náhrada umožňuje zakrýt potenciálně zneužitelný zabezpečený tunel, který byl vytvářen pomocí VPN spojení. Nevýhoda tohoto řešení, kdy je potřeba se serverem udržovat neustále navázané spojení, je řešena kontrolou, zda-li soket, na kterém komunikace probíhá, nebyl zavřen a tudíž nebyla komunikace z nějakého důvodu ukončena. Pokud tato situace nastane, je na server odeslána nová registrační zpráva, která způsobí otevření nového SSL spojení a obnovy komunikace. Schéma spojení aplikace se serverem je zobrazeno na obrázku 5.3.



Obrázek 5.3: Znázornění spojení se serverem pomocí zabezpečené komunikace. Pro lepší názornost je zabezpečený komunikační kanál znázorněn mimo nezabezpečenou rouru, nicméně fyzicky je veden přes stejný komunikační kanál.

Při výpadku spojení se serverem byla implementována vyrovnávací paměť, která slouží pro ukládání zpráv ze senzorů, jenž nemohou být odeslány na server. V této paměti se data ukládají po celou dobu výpadku spojení, přičemž výpadek může být libovolně dlouhý. Z tohoto důvodu je zapotřebí vyrovnávací paměť průběžně ukládat na paměť adaptéru, ale je nutné brát ohled na počet zápisů na SD kartu, na které je OS pro adaptér. Mé řešení umožňuje definovat jak cestu k souboru s uloženými zprávami, tak periodu ukládání zpráv a počet zpráv, po jehož překročení se zprávy ukládají. Po obnovení spojení jsou zprávy díky časovým razítkům odesílány chronologicky dle jejich příchodu a priority, ale postupně (s časovými rozestupy) aby nedošlo k nadměrnému zatížení serveru. V případě, kdy je při výpadku spojení adaptér vypnut z napájení a poté znovu zapnut, dojde k načtení zpráv ze souboru s perzistentními daty do paměti programu a ty jsou pak odesílány na server stejně jako ve výše zmíněném postupu.

V případě, že dojde k detekci nesprávného (neaktuálního) času v systému na adaptéru (ať už při spuštění nebo při náhlé desynchronizaci), jsou všechny aktuální zprávy zneplatněny a nejsou odeslány. Bez tohoto opatření by totiž na server chodily zprávy s nesprávným časovým razítkem, je tedy potřeba tyto zprávy uchovat a přiřadit jim správný čas příchodu dle délky výpadku časové synchronizace. Tento problém je řešen tak, že jakmile dojde k detekci nesprávného času, je spuštěn časovač, který měří délku od výpadku spojení. Při detekci opravy problému s časem jsou vypočteny časové intervaly mezi aktuálním časem

a délkou čekání zprávy ve frontě, z nichž je odvozena správná doba příchodu zprávy dle aktuální časové známky. Tato zpráva je opět označena jako platná a je jí umožněno odeslání na server zpětně ale se správnou časovou známkou.

5.4.6 Modul pro komunikaci s rozšiřující DPS

Modul pro komunikaci s rozšiřující DPS (zkráceně modul SPI) byl jedním z prvních implementovaných modulů a základů samotné AdaApp. Tento modul totiž zprostředkovává komunikaci mezi rozšiřující deskou a aplikací AdaApp. Pro aplikaci se tedy tato komunikace jeví jako přísun dat do/ze senzorové sítě prostřednictvím sběrnice SPI. Na její realizaci byl využit ovladač *spidev*, díky kterému je možné pracovat se sběrnicí SPI jako s běžným souborem pro čtení a zápis. Taktéž je pro komunikaci nutno sledovat hodnotu přerušovacího výstupu, který je nastaven na logickou hodnotu 1 při příchodu dat na SPI rozhraní.

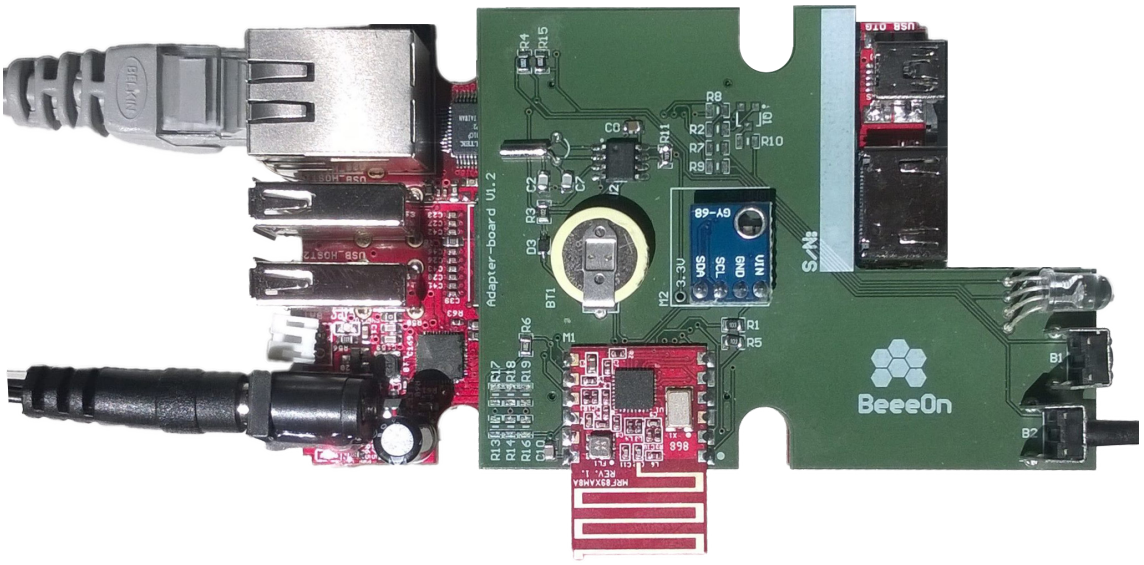
Tento modul byl, podobně jako modul pro síťovou komunikaci, rozdělen na 2 části – příjemce a odesílatel. Odesílatel obsahuje frontu požadavků směrem na SPI, kterou plní zprávy ze serveru. Příjemce pak při zjištění dat na SPI zprávu vyčte, zpracuje a předá modulu pro síťovou komunikaci pro další zpracování. Důvodem pro rozdělení bylo co nejvíce urychlit zpracování požadavků z SPI a tuto kritickou sekci zabírat jen pro potřebnou dobu obsluhy. Pro výlučný přístup k tomuto rozhraní bylo využito semaforů z knihovny *POCO*, tudíž není možné, aby nastala situace, kdy příjemce i odesílatel zároveň komunikují přes sběrnici. Toto řešení však nevyřešilo problém režimu komunikace **half-duplex** ovladače *spidev*, kdy docházelo (i přes výlučný přístup příjemce a odesílatele) k nežádoucímu příjmu dat během transakce odesílání. Tato data pak v rámci SPI rozhraní sice z rozšiřující desky odeslána byla (samotné SPI je **full-duplex**), nicméně ovladač *spidev* tato data nedokázal zpracovat a příjemce v modulu SPI tedy neobdržel žádnou informaci o nově příchozích datech. Tento problém vyřešila detekce nastavení přerušovacího vstupu při odesílání – jakmile je nastaven, vyšší prioritu má příjemce, který data přijme a uvolní tím sběrnici pro odesílání.

Při zpracování dat z SPI je zpráva dle specifikovaného protokolu zpracována, převedena do vnitřní struktury pro zprávy v rámci AdaApp a předána výše zmíněnému síťovému modulu pro odeslání na server. Pro zpracování SPI dat, které jsou reprezentovány jako pole bajtů, je zapotřebí tabulky typů (její podoba je v příloze C.2), která definuje sémantiku jednotlivým typům měřených veličin. Pokud typ senzoru/aktoru z přijatých dat není v tabulce typů nalezen, znamená to, že je tento typ zařízení pro aplikaci neznámý, tudíž je jeho odeslání na server potlačeno, protože chybí dodatečné informace o zpracování jeho veličin.

V současné době (5/2015) byl vytvořen nový bezdrátový protokol pro senzorovou síť, který nahrazuje původní řešení přes **MiWi** protokol a tato změna se dotkla také modulu pro komunikaci s rozšiřující deskou. Nebude již potřeba některých součástek, které převádí MiWi protokol na posloupnost bajtů, s čímž souvisí také nový návrh rozšiřující desky (pohled na jeho první reálný prototyp je na obrázku 5.4). Jelikož už tento nový formát komunikace nebude využívat SPI rozhraní pro předávání dat z bezdrátové sítě do AdaApp, bude potřeba modul upravit tak, aby nový formát podporoval. Jelikož však nové rozhraní zatím nebylo důkladně odzkoušeno, tato novinka zatím implementována nebyla.

5.4.7 Modul pro distribuci dat

Tento modul slouží jako nadstavbový modul pro šíření dat, procházejících přes AdaApp, na další rozhraní. Motivace pro vznik této komponenty byla umožnit okolí získávat data ze senzorů prostřednictvím rozhraní dostupného na adaptéru. Takovým rozhraním se stala pojmenovaná roura, přes kterou při aktivaci distribučního modulu proudí data ze senzorické



Obrázek 5.4: Prototyp nové rozšiřující desky (5/2015) pro nový bezdrátový protokol senzorové sítě. V pravém dolním rohu lze vidět tlačítko pro reset do továrního nastavení či LED diodu pro signalizaci stavu aplikace pro uživatele, ve spodní části pak anténu pro nový bezdrátový protokol (malá červená DPS). Tento PAN koordinátor obsahuje kromě vestavěného bateriově zálohovaného RTC obvodu (který byl už i na jejím předchůdci) nově také vestavěný senzor tlaku (modrá DPS uprostřed desky).

sítě ve formátu CSV. Na tuto rouru se může připojit libovolný uživatel a číst zmíněná data a využít je na vlastní analýzy aj. Této vlastnosti bylo využito pro implementaci testovacích skriptů, které díky pojmenované rouře ověřují, zda-li z aplikace proudí data (a) ve správném formátu.

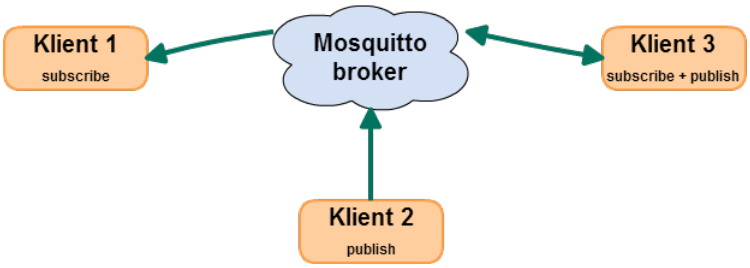
Tento modul kromě pojmenované roury obsahuje také historii všech dat ze senzorů, která distribučním modulem prošla. To nabízí možnost distribuovat tato historická data dalším aplikacím na adaptéru (AdaVis, AdaMan apod.) pro další zpracování. V současném stavu jsou možnosti tohoto modulu otevřené, připravené jsou však funkce pro převod zprávy ze senzorické sítě na formát CSV, XML a prostý text. Těchto funkcí je využito v modulu pro komunikaci přes MQTT protokol, který je popsán v podkapitole 5.4.8.

5.4.8 Modul pro komunikaci přes MQTT protokol

Modul pro komunikaci přes MQTT protokol byl původně navrhován jako součást modulu pro distribuci dat, nicméně po uvážení jeho potenciálu a rozsáhlosti jsem zvolil jeho architekturu jako další samostatný rozšiřující modul. Tento modul využívá knihovnu **mosquittopp** [16], což je tzv. *wrapper* knihovny **mosquitto.h** pro C++. S jeho využitím jsem implementoval celý modul pro komunikaci přes MQTT protokol, zahrnující připojení na zprostředkovatele, odebírání témat, publikování na zvolená témata aj. Celý modul je tvořen tak, aby byl vzorovou předlohou pro kohokoli, kdo by chtěl využívat v jazyce C++ komunikaci přes mosquitto.

Tento modul se tedy stal dalším nástrojem pro distribuci dat, který je dostupný z adaptéru komukoliv, kdo se připojí na patřičné téma a bude z něj odebírat zprávy – příklad takového systému s klienty je na obrázku 5.5. U komunikace přes mosquitto lze nastavit

různé způsoby zabezpečení pro připojování klientů na zprostředkovatele, na témata aj., taktéž umožňuje vytvořit tzv. *mosty*, které spojují zprostředkovatele na více zařízení pomocí spojování stromů témat tak, že je možné propojit více MQTT sítí dohromady (příklad takového mostu je níže na obrázku 5.7). Žádné z těchto nastavení však v současnosti zatím nebylo využito.



Obrázek 5.5: Ukázka principu komunikace pomocí mosquitto. Na zprostředkovatele zpráv (broker) se připojují jednotliví klienti, kteří mohou zprávy odebírat (subscribe) z libovolných témat a/nebo na ně svými zprávami přispívat (publish).

V rámci distribuce dat skrze tento modul bylo vytvořeno několik témat, které distribuují data ze senzorové sítě v různých formátech (CSV s vybranými atributy, CSV se všemi atributy, XML) a taktéž několik servisních témat, které slouží pro vnitřní komunikaci mezi aplikacemi na adaptéru. Díky tomuto modulu je tedy možné zasílat servisní informace aplikaci **AdaMan**, jejímž autorem je student Peter Malina, a informovat tak okolí o nastálé chybě, nekonzistentních stavech apod. Taktéž AdaMan může kontaktovat AdaApp s požadavkem na vypnutí se kvůli aktualizaci systému aj. Konkrétní formát protokolu zpráv mezi aplikacemi v současné chvíli nebyl specifikován, jelikož druhá strana zatím nepodporuje komunikaci přes moquitto, nicméně byla tato varianta vybrána jako meziprocesová komunikace na adaptéru mezi těmito aplikacemi.

S výhodou byl tento modul taktéž využit při testování nového druhu zařízení senzorové sítě, kde dle stejného protokolu zpráv, jako implementuje modul pro komunikaci s rozšiřující deskou, jen přes rozhraní mosquitto, byla vytvořena komunikace se senzorovou sítí namísto komunikace přes SPI sběrnici. Velkou výhodou tohoto modulu je to, že je pak možné pro účely testování vytvářet a ověřovat funkčnost virtuálních zařízení dříve, než budou fyzicky vyrobena (podobně jako u VS), nicméně dle protokolu, který využívá modul pro komunikaci s PAN koordinátorem.

5.4.9 Modul pro ovládání vstupů/výstupů

Tento modul se jménem *IOControl* vznikl pro účely ovládání vstupních a výstupních vývodů a zařízení, jenž jsou na ně připojena. Konkrétně se na adaptéru jedná o kombinaci tlačítek a LED diod. Tlačítka slouží například pro manuální uvedení adaptéru do továrního nastavení, případně pro jinou dodatečnou funkcionalitu. Konkrétní funkce pro tlačítka zatím nebyla specifikována, nicméně lze rozpoznat 2 úrovně stisku – krátký (<4 s) a dlouhý (>4s) stisk. Toto rozpoznávání je inspirováno utilitou **evtest**, která snímá události na definovaných vstupech a generuje záznam o této události, který obsahuje informace o typu událost, času a původci této události.

Ovládání LED diod slouží pro účely ladění (vestavěná LED na adaptéru) i pro základní signalizaci stavu adaptéru uživateli (LED na rozšiřující desce). Ve vytvořeném modulu lze

nastavit LED diodě stavy zapnuto/vypnuto, případně blikání ve zvoleném počtu s pevně stanovenou periodou.

5.5 Plán budoucí funkcionality

Pro blízkou budoucnost aplikace AdaApp, jakožto součásti projektu inteligentní domácnosti, vyvíjeném na fakultě FIT VUT v Brně, je plánováno několik dílčích úprav a nových funkcí. Jednou z nich je komunikace s programem AdaVis, jehož autorem je student Martin Sakin, a to tak, že si AdaVis bude nejen odebírat data z pojmenované roury a zobrazovat je na výstup, ale i žádání si delší historie a obecně dodatečných informací k danému senzoru (či všem senzorům) přes adaptér.

Dalším vývojem projde také komunikace mezi AdaApp a AdaMan, kde by měly vzniknout pokročilejší schémata komunikace – reakce na nekonzistence systému, aplikace, konfigurace nebo nesprávný či neznámý formát přijímaných zpráv ze serveru.

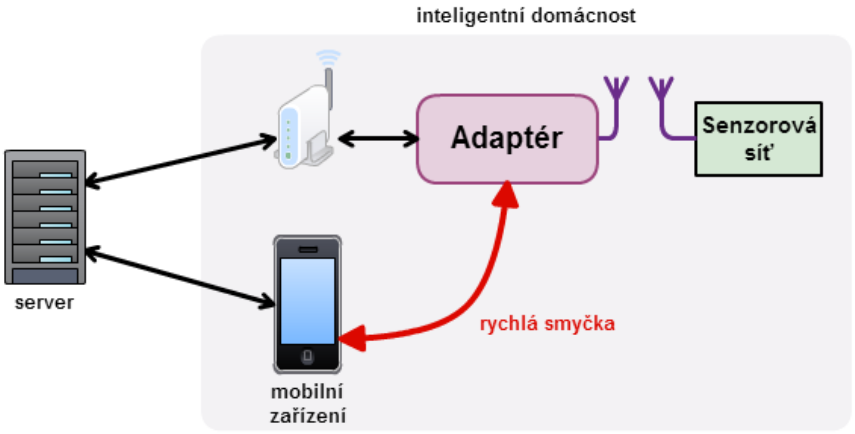
Jelikož na adaptéru existuje tzv. *message broker* pro MQTT zprávy **mosquitto**, který umí pomocí tzv. *mostů* spojovat brokery na různých zařízení a propojit tak celou řadu zařízení spolu do jedné sítě, je pak možné této vlastnosti využít na sbírání statistik na jednom centrálním místě (např. server) od všech adaptérů a dostávat tak například servisní informace o chodu aplikace AdaApp na adaptérech v domácnostech pro snadnější identifikaci a rychlejší odstranění případných chyb.

5.6 Možnosti rozšíření

Jelikož je aplikace navržena jako modulární, s využitím konfiguračních souborů lze jednoduše implementovat nový rozšiřující modul aplikace a tento pak povolit/zakázat a konfigurovat v souboru s příponou *.ini*. Taktéž se lze při návrhu modulu inspirovat v hlavní funkci **main** AdaApp, jakožto příklad pro integraci modulu do aplikace a její spouštění.

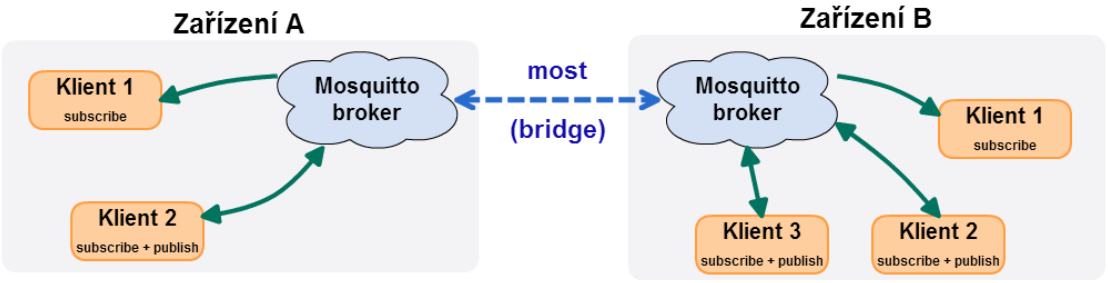
Možnost rozšíření aplikace vidím v implementaci tzv. *rychlé smyčky* mezi mobilním zařízením a adaptérem, čímž by se mohlo předcházet ztrátu možnosti kontroly domácnosti v případě výpadku spojení mezi adaptérem, serverem a mobilním zařízením. Uživatel by díky rychlé smyčce mohl sepnout aktory či získávat data ze senzorů rychleji, napřímo a bez nutnosti serveru. Toto spojení by mohlo být realizováno s pomocí bezdrátových protokolů Wi-Fi, či Bluetooth. Toto rozhraní by mohlo být zprostředkováno pomocí externích USB modulů, připojených k desce OLinuXino přes USB port. Rozšíření o možnost rychlé smyčky však není triviální úkol, jelikož je potřeba z pohledu aplikace převzít řízení domácnosti a o vzniklých událostech (požadavcích) z mobilního zařízení informovat server a taktéž by si AdaApp musela udržovat seznamy zařízení, které s adaptérem komunikují (senzory a aktory), což přidává stavovost a při restartu aplikace také potenciální riziko neaktuálních informací v aplikaci. Znázornění principu rychlé smyčky je na obrázku 5.6.

Další možností rozšíření by mohlo být například vytvoření abstraktní třídy pro rozšiřující moduly, které by dováželo rozhraní společné pro všechny rozšiřující moduly (například odkaz na *Aggregator* apod.) a obsahovalo nástroje společné pro celý projekt (například vytvoření *Loggeru* pro výpisy do souboru, načtení příslušného konfiguračního souboru atd.). Z této třídy by pak dědily všechny rozšiřující moduly a rozšiřovaly by svoji třídu o specifické vlastnosti daného modulu. Toto by pak značně zjednodušilo vývoj nových modulů pro AdaApp a zvýšilo využití OOP v navržené aplikaci.



Obrázek 5.6: Znázornění možnosti rozšíření aplikace pomocí rychlé smyčky. Ta umožňuje ovládat inteligentní domácnost bez nutnosti připojení k serveru.

Za možnost rozšíření je možné brát i rozšíření modulu *Distributoru* (popsaný v podkapitole 5.4.7), který umožňuje veškeré odchozí zprávy na server jak sbírat do historie, tak ihned odesílat do svého okolí. V současné době je možné zprávu sdílet pomocí Distributoru na pojmenovanou rouru ve formátu CSV, přes modul *MosqClient* zprávy šířit přes protokol *MQTT* (jednak mezi lokální odebíratele na adaptéru a jednak s možností distribuovat zprávy přes tzv. *mosty* do okolních zařízení), přičemž možnosti distribuce dat vyčerpány nejsou. Příklad toho, jak by vypadalo spojení více zařízení pomocí mostů je na obrázku 5.7. Jako rozšíření bych viděl distribuovat zprávy na sběrnici D-Bus, USB, nebo např. na lokální síťové sokety. S využitím historie zpráv by bylo možné také provádět lokální statistiky a analýzy nad historií dat ze senzorů a vnést umělou inteligenci do brány inteligentní domácnosti.



Obrázek 5.7: Ukázka spojování zprostředkovatelů zpráv mosquitto pomocí mostů.

S předchozím návrhem se nabízí myšlenka řídit pomocí adaptéru domácnost autonomně. Například by mohl adaptér sledovat povolené limity senzorových veličin a při jejich překročení by autonomně vykonal řídicí akci[2] bez zásahu uživatele (velmi užitečné například při výpadku spojení se serverem, kdy informace o potenciální havárii nebo překročení limitů by neměla žádnou odezvu, případně by o ní nebyl server ani informován). Velkou výhodou by byla jistě vysoká přidaná hodnota pro uživatele, který by tak spoléhal na domovní automatizaci i v případě nekonzistentního či jinak nepovoleného stavu domácnosti a obecně celého systému. Nicméně je pochopitelně tento úkol velmi složitý, jelikož je potřeba specifi-

kovat (nejlépe dynamicky) konfiguraci, s jakou se bude adaptér chovat (jaké veličiny mají vliv na které aktory, jaký je práh a délka překročení dané hodnoty atd.) a také se tato konfigurace může měnit na základě ročního období, týdenních či denních intervalů.

[2]Modelový příklad této události by mohlo být hlídání pokojové teploty v domě a při jejím překročení by adaptér autonomně vykonal akci pro vypnutí domovního topení.

Kapitola 6

Testování a nasazení v domácnostech

Testování aplikace AdaApp pro ověření její funkčnosti a stability bylo prováděno v iteracích po celou dobu jejího vývoje. Vývoj aplikace inkrementálně zvyšoval celkovou funkcionalitu aplikace, která byla jednou měsíčně uvedena do *produkčni*[1] verze, která plnila funkci nasazené stabilní verze programu na reálném HW pro potřeby testování systému.

Kromě samotné *produkční* verze aplikace (označována v systému inteligentní domácnosti jako beta verze) byla nasazena také vývojová verze aplikace na reálném HW (alfa verze) s nejaktuálnější verzí aplikace, novou funkcionalitou a aktualizovanou během celého měsíce. Z této verze se pak po ukončení měsíčního vývoje stala beta verze a alfa verze byla uvolněna pro nový vývoj. Tyto iterace ve vývoji velmi usnadnil verzovací systém Git – v repozitáři s aplikací byly vytvořeny příslušné větve verzí, v kterých se vždy nacházely aktuální verze zdrojových kódů tak, aby bylo možné okamžitě nasadit požadovanou verzi (alfa, beta, testovací…) na požadovanou platformu.

V pokročilejších fázích vývoje byla produkční verze aplikace pravidelně každý měsíc nasazována na adaptéry do domácností v počtu cca 8 kusů. Zde adaptéry běžely neustále po dobu 3 týdnů, byly k nim připojeny senzory, které snímaly dané veličiny v rámci domácnosti, a připomínky ze zpětné vazby z tohoto nasazení jsem využil pro zvýšení spolehlivosti aplikace AdaApp. Z adaptérů byly také čteny informace o spotřebovaných zdrojích, kde i při uložení desítek tisíc zpráv do historie paměť adaptéru narůstala takřka neznatelně (cca 1 % z 512 MB RAM). Také vytížení procesoru bylo po chybném hospodaření s vlákny sníženo na jednotky %. Z tohoto přímého nasazení do domácností mi byla poskytována zpětná vazba slovně, ale také údaje o době chodu aplikace, kdy bylo možné zjistit, zda-li se aplikace během jejího nasazení dostala do nekonzistentního stavu a musela být znovu spuštěna.

Pro automatizaci testování pomocí systému Jenkins jsem vytvořil testovací skripty na funkčnost aplikace, které jednak testují, zda-li aplikace v aktuální vývojové verzi je možné spustit bez pádu (tzn. z instalační podsložky lze aplikaci spustit – vše je nakonfigurováno tak jak má) a jednak jestli aplikace poskytuje z definovaného vstupu správný výstup. Tento skript na test komunikace využívá (jak již bylo zmíněno dříve) modulu VS a sleduje výstup z pojmenované roury. Toto automatizované testování umožňuje díky funkčním testům rychle upozornit na chybu, která při vývoji nastala a poškodila funkční komunikaci, případně na jiné chyby způsobené poslední změnou ve zdrojových souborech aplikace.

[1] Produkční verzí byla označována vývojová verze aplikace, u které však celý měsíc nedocházelo ke změnám nebo aktualizacím.

Kapitola 7

Závěr

Po teoretickém rozboru možností řízení inteligentní domácnosti, výběru cílové platformy a návrhu jednotlivých komponent proběhlo seznámení s deskou A10-OLinuXino-LIME, na které byly nainstalovány nástroje potřebné pro vývoj. Také proběhlo seznámení s komponentami rozšiřující DPS pro komunikaci s reálnými senzory. Dále byla vytvořen základní prototyp aplikace, který byl přeložen křížovým překladem, nahrán na desku a spuštěn pro ověření funkčnosti. Dále proběhlo seznámení s knihovnou POCO, byly vyzkoušeny nástroje pro práci s XML a napsán XML parser pro příchozí zprávy ze serveru.

Další vývoj pokračoval vytvořením síťového modulu pro komunikaci mezi serverem a AdaApp, implementací modulu pro HW senzory i VS. Současná verze aplikace umí interpretovat zprávy jak ze senzorů (HW + SW), tak i ovládat aktory (podporované typy jsou v tabulce typů) a je nasazena v provozu na desce OLinuXino. Během vývoje rozhraní pro komunikaci s HW senzory se vlivem změn u senzorové části projektu inteligentní domácnosti změnil i tento modul, který je popsán v kapitole 5.4.6.

Během vývoje byla nasazena taktéž zabezpečená komunikace se serverem přes SSL, včetně zotavení se při výpadku spojení a ukládání dat na nevolatilní paměť adaptéru. Taktéž bylo implementováno řešení výpadku času či jeho nesprávné synchronizaci a s tím související priority zpráv.

Pro sdílení dat s dalšími aplikacemi a komunikaci na adaptéru byl implementován modul Distributor, který v sobě skýtá možnosti zasílat veškeré zprávy, které chodí z AdaApp na server, taktéž na pojmenovanou rouru, ukládání zpráv do historie pro AdaVis, modul MQTT pro komunikaci (nejen) s AdaManem aj. Modul lze snadno rozšířit pro komunikaci s dalšími entitami v systému. Pro doplňkové funkce, jako jsou zobrazování stavu adaptéru (pomocí vestavěné LED diodě na desce OLinuXino a rozšiřující desce pro HW senzory), nebo čtení událostí na tlačítku (opět na obou deskách) byl implementován modul *IOcontrol*.

Během vývoje byla aplikace testována na reálném HW v domácnostech, což mi poskytlo užitečnou a důležitou zpětnou vazbu k dalším úpravám pro zvýšení stability a spolehlivosti aplikace. Taktéž byly vytvořeny funkční skripty pro automatizované testování aplikace pomocí systému průběžné integrace Jenkins. Díky tomuto systému je možné získat informaci o porušení funkčnosti aplikace při aktualizaci zdrojových souborů v Git repozitáři aplikace. Výsledky testování jsou popsány v kapitole 6.

Jelikož je aplikace stále ve vývoji, jsou určité funkce pouze předchystány pro pozdější úpravy, až danou funkčnost bude podporovat celý systém. Do budoucna by bylo vhodné aplikaci rozšířit o abstraktní třídu pro rozšiřující moduly tak, aby při implementaci nového modulu bylo možné zdědit společné vlastnosti každého modulu, a rozšířit tuto abstraktní třídu o specifické vlastnosti. Dále by bylo možné uvažovat o propojení telefonu a adaptéru

pomocí Bluetooth rozhraní, případně Wi-Fi sítě. V rámci vývoje bude v blízké budoucnosti navázána komunikace s aplikací AdaVis pro získávání historických dat z modulu **Distributor**, implementace funkcí pro pro reset adaptéru do továrního nastavení a v neposlední řadě také neustálý vývoj stability a spolehlivosti AdaApp tak, aby nebylo potřeba žádného zásahu do aplikace od uživatele či vývojáře po celou dobu běhu systému v domácnostech.

Literatura

[1] Mosquitto. [online], [cit. 7. 1. 2015].
URL http://mosquitto.org/

[2] Applied Informatics Software Engineering GmbH: POCO C++ libraries. [online], 2006-2014, [cit. 21. 12. 2014].
URL http://pocoproject.org

[3] Applied Informatics Software Engineering GmbH: POCO C++ libraries - Features. [online], 2006-2014, [cit. 21. 12. 2014].
URL http://pocoproject.org/features.html

[4] Applied Informatics Software Engineering GmbH: A Guided Tour Of The POCO C++ Libraries. [online], 2014, [cit. 21. 12. 2014].
URL http://pocoproject.org/docs/00100-GuidedTour.html

[5] Belkin: Home Automation. [online], [cit. 2. 1. 2015].
URL http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/

[6] Bellamy, B.: Arduino vs. Raspberry Pi vs. CubieBoard vs. Gooseberry vs. APC Rock vs. OLinuXino vs. Hackberry A10. [online], 2013, [cit. 30. 12. 2014].
URL http://techwatch.keeward.com/geeks-and-nerds/arduino-vs-raspberry-pi-vs-cubieboard-vs-gooseberry-vs-apc-rock-vs-olinuxino-vs-hackberry-a10/

[7] Cannynet: Chytré a úsporné řešení pro Vaši firmu. [online], 2011-2015, [cit. 2. 1. 2015].
URL http://www.cannynet.com/

[8] ELKO EP: ELKO EP — Představení společnosti. [online], [cit. 2. 1. 2015].
URL http://www.elkoep.cz/o-nas/

[9] HAIDY: HAIDY HOME. [online], [cit. 2. 1. 2015].
URL http://haidy.cz/cs/produkty/haidy-home/

[10] IoT: Architektura. [online], 2014, [cit. 3. 1. 2015].
URL https://ant-2.fit.vutbr.cz/projects/iot/wiki/Architektura

[11] IoT: Příklady použití. [online], 2014, [cit. 2. 1. 2015].
URL https://ant-2.fit.vutbr.cz/projects/iot/wiki/Příklady_použití

[12] Kašpárek, T.; Kočí, R.; Peringer, P.; aj.: Operační systémy IOS - studijní opora, 2006.

[13] Kilián, K.: Samsung T9000: chytrá lednička s Androidem. [online], 2013, [cit. 14. 1. 2015].
URL http://www.svetandroida.cz/samsung-t9000-chytra-lednicka-s-androidem-201301

[14] Landoni, B.: A Comprehensive Comparison of Linux Development Boards. [online], 2013, [cit. 30. 12. 2014].
URL http://www.open-electronics.org/a-comprehensive-comparison-of-linux-development-boards/

[15] Marshall, A. D.: Programming in C - UNIX System Calls and Subroutines using C. [online], 1994-2005, [cit. 7. 1. 2015].
URL http://www.cs.cf.ac.uk/Dave/C/

[16] Mosquitto: mosquittopp. [online], 2014, [cit. 13. 5. 2015].
URL http://mosquitto.org/api/files/cpp/mosquittopp-h.html

[17] Olimex: A10-OLinuXino-LIME. [online], 1997-2014, [cit. 30. 11. 2014].
URL https://www.olimex.com/Products/OLinuXino/A10/A10-OLinuXino-LIME/open-source-hardware

[18] Olimex: A10-OLinuXino-LIME-4GB. [online], 1997-2014, [cit. 26. 12. 2014].
URL https://www.olimex.com/Products/OLinuXino/A10/A10-OLinuXino-LIME-4GB/

[19] Olimex: OLinuXino - Open Source Hardware Boards. [online], 1997-2014, [cit. 20. 12. 2014].
URL https://www.olimex.com/Products/OLinuXino/open-source-hardware

[20] Olimex: A10-OLinuXino-LIME User's manual. [online], 2014, [cit. 20. 12. 2014].
URL https://www.olimex.com/Products/OLinuXino/A10/A10-OLinuXino-LIME/resources/A10-OLinuXino-LIME_manual.pdf

[21] Open Source Automation: Open Source Automation Wiki. [online], 2014, [cit. 2. 1. 2015].
URL http://www.opensourceautomation.com/wiki/index.php?title=Main_Page

[22] Rayhawk, J.: D-Bus. [online], 2014, [cit. 7. 1. 2015].
URL http://www.freedesktop.org/wiki/Software/dbus/

[23] Růžička, R.; Schwarz, J.; Strnadel, J.: Mikroprocesorové a vestavěné systémy - studijní opora, 2006.

[24] sunxi: sunxi Wiki - SPIdev. [online], 2014, [cit. 14. 1. 2015].
URL http://linux-sunxi.org/SPIdev

[25] sunxi: sunxi Wiki - Main Page. [online], 2015, [cit. 14. 1. 2015].
URL http://linux-sunxi.org/Main_Page

[26] Valeš, M.: *Inteligentní dům*. 21. století, ERA, 2006, ISBN 9788073660628.

Příloha A

Obsah CD

V této příloze je výčet adresářové struktury přiloženého CD disku. Kromě jiného jsou zde uloženy všechny zdrojové soubory potřebné pro překlad, spuštění a instalaci aplikace AdaApp na požadované cílové zařízení. Taktéž je na disku uložena elektronická verze této práce ve formátu PDF s hyperlinky.

- **fig/** – obrázky a fotografie použité v písemné práci
- **pdf/** – výsledný PDF dokument této práce
- **src/** – zdrojové soubory pro AdaApp (viz příloha B)
- **tex/** – zdrojové soubory pro LaTeX k sestavení výsledného PDF souboru

Příloha B

Adresářová struktura Git repozitáře se zdrojovými kódy

Tato příloha obsahuje výčet adresářové struktury se zdrojovými soubory spolu s popisem jejich obsahu. Do tohoto výčtu není zahrnut adresář **.git** potřebný pro verzovací nástroj Git.

- **adapter_files/** - Soubory pro práci s AdaApp - například spouštěcí skript pro manažera služeb aj.
- **etc/** – všechny konfigurační soubory pro aplikaci, včetně rozšiřujících modulů a tabulky typů
- **jenkins/** – funkční skripty pro server průběžné integrace **Jenkins**
- **scripts/** – skripty pro snadnější ovládání aplikace z příkazové řádky
- **tools/** – pomocné utility pro vývoj AdaApp
- **/** – kořenová složka repozitáře obsahující všechny zdrojové soubory (*.cpp a *.h) včetně překladových skriptů

Příloha C

Formát XML zpráv

Tato příloha obsahuje definovanou strukturu všech využívaných XML dat v implementované aplikaci AdaApp.

C.1 Formát zpráv z/na AdaApp

Pro přenos zpráv z aplikace na server je využíván následující formát[1] zprávy:

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter_server adapter_id="<adapter_id>" fw_version="<fw_version>"
                protocol_version="<proto_version>" state="<command_type>"
                time="<timestamp>">
  <device id="<device_id>">
    <battery value="<bat_value>"/>
    <rssi value="<rssi_value>"/>
    <lqi value="<lqi_value>"/>
    <values count="<values_count>">
      <value offset="<sensor_offset>" type="<sensor/actuator_quantity_type>">
        <sensor/actuator_actual_value>
      </value>
      ...
    </values>
  </device>
</adapter_server>
```

Pro přenos zpráv ze serveru do aplikace je využíván následující formát[1]:

```
<?xml version="1.0" encoding="UTF-8"?>
<server_adapter protocol_version="<proto_version>" state="<command_type>"
                id="<device_id>" adapter_id="<adapter_id>">
  <value type="<sensor/actuator_quantity_ID>" offset="<sensor_offset>">
    <wanted_value>
  </value>
</server_adapter>
```

[1] Hodnoty ve špičatých závorkách symbolizují substituci hodnoty za její význam.

C.2 Tabulka typů

Tabulka typů, jejíž význam je popsán v kapitole 4.2.2, využívá následujícího formátu[1]:

```
<?xml version="1.0" encoding="UTF-8"?>
<table version="<version>" timestamp="<timestamp>">
  <sensors>
    <value type="<sensor_quantity_ID>" name="<type_description>"
           data_type="<data_type>" unit="<unit>"
           transform="<operation+modifier>">
    </value>
    ...
  </sensors>
  <actuators>
    <value type="<actuator_quantity_ID>" name="<type_description>"
           data_type="<data_type>" unit="<unit>"
           transform="<operation+modifier>" size="<sizeof_actuator_value>">
    </value>
    ...
  </actuators>
</types_table>
```