VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SIMULÁTOR ELEKTROMĚRU S DLMS PROTOKOLEM
ELECTRIC METER SIMULATOR WITH DLMS PROTOCOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Kateryna Tsymbal

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Tomáš Lieskovan

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Studentka: Kateryna Tsymbal ***ID:*** 209239

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Simulátor elektroměru s DLMS protokolem

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je vytvoření simulátoru chytrého elektroměru (smart metru) komunikujícího pomocí protokolu DLMS. Virtuální elektroměr by měl být napsán pomocí programovacího jazyka Java nebo Python. Implementace by měla být realizována v hardwarově omezeném prostředí (Raspberry Pi).

DOPORUČENÁ LITERATURA:

[1] Gurux DLMS Server: https://www.gurux.fi/Gurux.DLMS.Server

[2] JIRKA, Bc Matěj. Framework DLMS/COSEM pro sběr dat v AMM systémech.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. Tomáš Lieskovan

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce je zaměřena na seznámení se s protokolem DLMS/COSEM a jeho funkcemi. Dále je stručně představen programovací jazyk Java a minipočítač Raspberry Pi. Pro simulaci měření pomocí chytrého měřiče je využit protokol DLMS/COSEM, který zajišťuje komunikaci mezi simulátorem elektroměru a koncentrátorem, čehož se hojně využívá v chytrých sítích (tzv. smart grids).

V první části práce jsou rozebrány důležité informace ohledně energetických sítí a jejich souvislost s chytrými měřiči, důležitost chytrých měřičů v moderních sítích a úloha chytrých sítí, které slouží k efektivnímu měření energií v určité oblasti (např. elektrické energie). Data naměřená v chytrých sítích lze jednoduše analyzovat a pomocí tohoto zefektivnit spotřebu. Dále jsou v první části zmíněny důležité informace ohledně protokolu DLMS/COSEM, programovacího jazyku Java a minipočítače Raspberry Pi.

V druhé části práce je vytvořeno testovací prostředí pro vyzkoušení funkčnosti simulátoru chytrého měřiče, který po svém spuštění komunikuje pomocí protokolu DLMS/COSEM s koncentrátorem a předává mu naměřené hodnoty. Naměřené hodnoty jsou pro testovací účely ručně nadefinovány pomocí změn v kódu ve vývojovém prostředí Eclipse.

Cílem práce bylo vytvořit simulátor chytrého měřiče, který vypisuje předem definované hodnoty a předává je koncentrátoru, čehož bylo docíleno pomocí knihovny Gurux.DLMS. Nakonec byla provedena analýza této komunikace pomocí programu Wireshark. Tato bakalářská práce je užitečná pro jednoduché porozumění protokolu DLMS/COSEM a jeho využití v chytrých sítích.

KLÍČOVÁ SLOVA

COSEM, DLMS, Eclipse, Gurux, Chytrá síť, Chytrý měřič, Java, OBIS, Raspberry Pi, Wireshark.

ABSTRACT

This bachelor thesis is focused on getting acquainted with the DLMS/COSEM protocol and its functions. Furthermore, the Java programming language and the Raspberry Pi minicomputer are briefly introduced. The DLMS/COSEM protocol is used to simulate measurements using a smart meter, which ensures communication between the meter simulator and the concentrator. Communication provided by the DLMS/COSEM protocol is widely used in smart grids.

The first part of the thesis discusses important information about energy networks and their relationship with smart meters, the importance of smart meters in modern networks and the role of smart networks, which are used for effective energy measurement in a particular area (e.g. for measuring electricity). Data measured in smart grids can be easily analyzed and used to make consumption more effective. The first part also mentions important information about the DLMS/COSEM protocol, the Java programming language and the Raspberry Pi minicomputer.

In the second part of the thesis, a test environment is created for testing the smart meter simulator, which communicates using the DLMS/COSEM protocol with the concentrator and transmits the measured values to it. Measured values are manually defined for testing purposes using code changes in the Eclipse IDE.

The aim of the work was to create a smart meter simulator that lists predefined values and passes them to the concentrator, which was achieved using the Gurux.DLMS library. Finally, an analysis of this communication was performed using Wireshark. This bachelor thesis is useful for a simple understanding of the DLMS/COSEM protocol and its use in smart grids.

KEYWORDS

COSEM, DLMS, Eclipse, Gurux, Java, OBIS, Raspberry Pi, Smart grid, Smart meter, Wireshark.

TSYMBAL, Kateryna. *Simulátor elektroměru s DLMS protokolem*. Brno, 2021, 55 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Tomáš Lieskovan

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Simulátor elektroměru s DLMS protokolem“ jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autorky

PODĚKOVÁNÍ

Děkuji vedoucímu práce panu Ing. Tomáši Lieskovanovi za odbornou pomoc, cenné rady, trpělivost a obrovskou ochotu při zpracování mé bakalářské práce.

Obsah

Úvod

V lidském životě lze energii definovat jako primární faktor existence. Průběh historického formování společnosti přímo souvisí s povahou využívání energie. Rozvoj lidstva je neoddělitelný od rozmachu energetického průmyslu. S postupem doby ovšem rostou požadavky na efektivitu využití těchto energií. S tímto se pojí snaha o zavádění tzv. „smart gridů“, neboli chytrých/inteligentních sítí. S pojmem „smart grid“ je úzce spjat pojem „smart meter“ (chytrý měřič), který je, mimo jiné, schopen měřit např. elektrickou energii a také plní funkci vyhodnocování údajů z měření. Mezinárodně nejuznávanějším standardem pro výměnu dat z měřičů je DLMS/COSEM (Device Language Message Specification/Companion Specification for Energy Metering). Dobrá spolupráce mezi dodavateli energie a výrobci měřidel, DLMS UA (User Association), umožňuje neustále rozšiřovat rozsah DLMS/COSEM a zajistit pokrytí nových aplikací a nových komunikačních médií [1].

DLMS (Device Language Message Specification, původně Distribution Line Message Specification, IEC 62056-5-3:2017 [2]) je specifikace aplikační vrstvy DLMS/COSEM AL (Application Layer) určená k podpoře zasílání zpráv do a z (energetických) distribučních zařízení. Podporovány jsou aplikace jako dálkový odečet měřidel, dálkové ovládání a služby schopné měřit jakýkoliv druh energie, jako je např. elektřina, voda, plyn a teplota, ale i ostatní měřitelné veličiny, jako např. čas.

Protokol DLMS/COSEM je následně využit k sestrojení vlastního simulátoru měřiče, který je naprogramován pomocí programovacího jazyku Java a realizován v hardwarově omezeném prostředí, tedy na Raspberry Pi. Jako koncentrátor slouží open-source software Gurux DLMS Director, který shromažďuje hodnoty vygenerované simulátorem chytrého měřiče. Komunikace mezi měřičem a koncentrátorem je nakonec analyzována pomocí programu Wireshark pro snadnější pochopení dané problematiky.

1 Teoretická část

V této části práce bude rozebráno všechno podstatné ohledně energetických sítí, které po implementaci chytrých měřičů, využívajících protokol DLMS/COSEM, mohou měřit důležitá data pro pozdější zpracování ke zvýšení efektivity spotřeby nejen elektrické energie. V následujících kapitolách bude rovněž analyzován programovací jazyk Java, pomocí kterého byl naprogramován simulátor chytrého měřiče, který byl následně, po otestování funkčnosti, přesunut na Raspberry Pi (hardwarově omezené prostředí).

1.1 Elektrické sítě

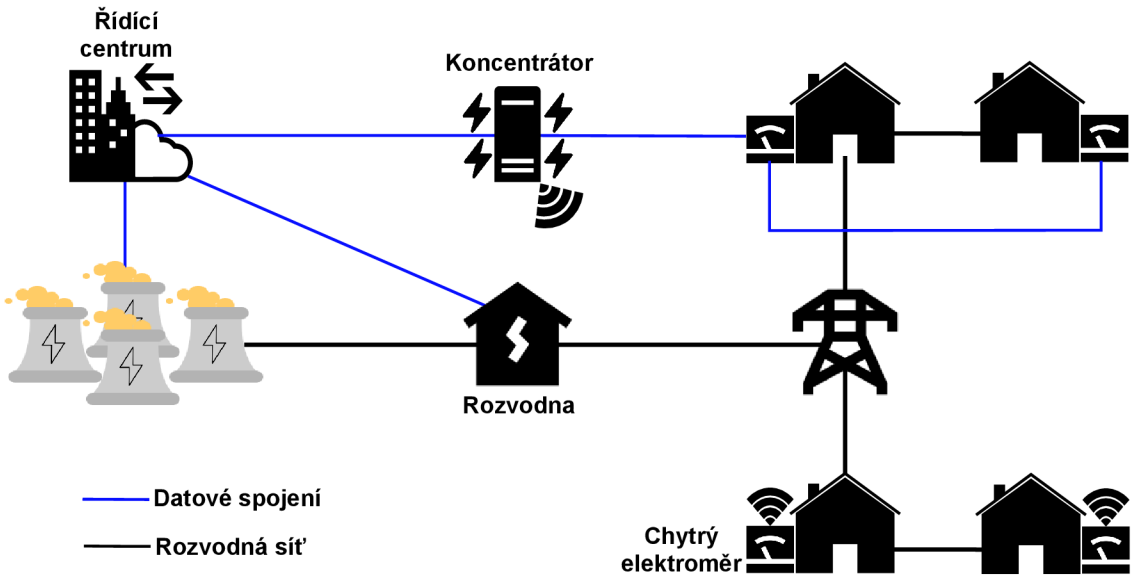
Nejčastější oblastí pro implementaci chytrých měřičů jsou elektrické sítě. Elektrické sítě jsou propojené sítě pro poskytovaní elektřiny od producentů ke konzumentům. Elektrická síť je zpravidla formována následujícími položkami:

- elektrárny vyrábějící elektrickou energii;
- elektrické rozvodny ke zvýšení elektrického napětí pro přenos rozvodnými sítěmi nebo k snížení elektrického napětí k distribuci ke konzumentům;
- vedení vysokého napětí, kterým je energie přenášena ze vzdálených zdrojů do středisek poptávky;
- distribuční linky spojující jednotlivé spotřebitele.

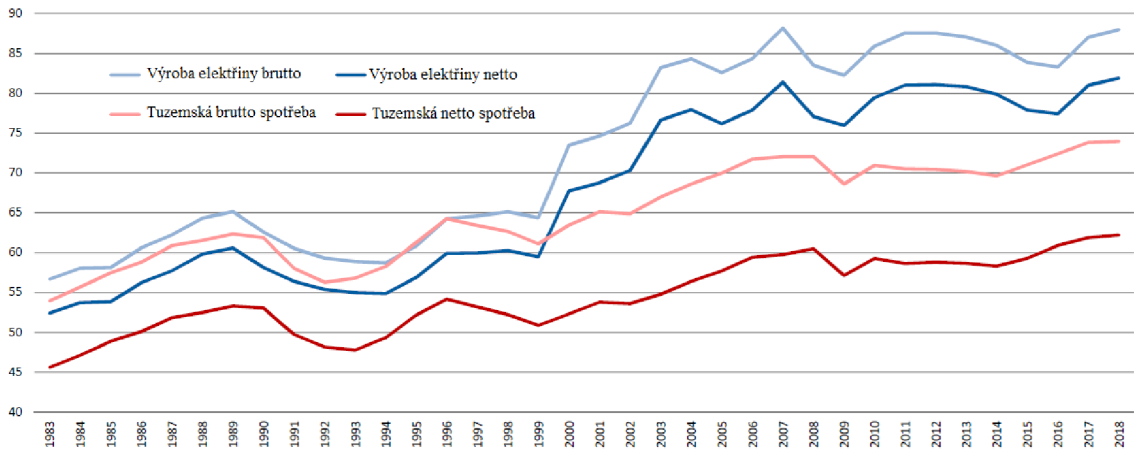
Normalizované napětí – zvyšováním napětí se zvyšuje přenášený výkon, aniž by se zvyšovaly ztráty přenosem. Velikosti elektrických sítí se mohou lišit od pokrytí jedné budovy přes národní sítě, které pokrývají celé země, až po mezinárodní sítě, které mohou procházet kontinenty.

Síť, kde výrobci i spotřebitelé využívají chytré měřiče se dá označovat jako „Smart Grid“ (chytrá síť – příklad chytré sítě na Obr. 1.1). Tyto typy sítí mohou automaticky, chytře a účinně ovládat celý distribuční systém. V České republice probíhá projekt (Smart region Vrchlabí), jehož cílem je dosáhnout automatizace a online monitorování chytré elektrické sítě.

I přes to, že jsou elektrické sítě velice rozšířené, v roce 2017 bylo stále 1,2 miliard lidí bez připojení k elektrické síti, globální míra přístupu k elektrické energii tedy dosáhla 89 % (v ČR 100 % přístup k elektrické síti) [3]. U elektrických sítí samozřejmě také existuje potřeba zabezpečení, mohou totiž být náchylné k neoprávněnému vniknutí nebo útoku. S technologickým vývojem elektrických sítí a rozmachem IoT (Internet of Things) se začínají stávat bezpečnostním rizikem také kybernetické útoky. Specifické problémy se týkají komplikovanějších počítačových systémů, které jsou nezbytné pro správu chytrých elektrických sítí.



Obr. 1.1: Příklad chytré sítě pro měření elektrické energie



Obr. 1.2: Roční spotřeba elektrické energie v ČR v TWh [4]

1.2 Chytrý měřič

Chytrý měřič je elektronické zařízení, které zaznamenává informace, jako je spotřeba elektrické energie, vody, plynu nebo měření elektrického napětí, elektrického proudu, teploty atd. Informace naměřené chytrými měřiči jsou pomocí systémů pro dálkový (drátové či bezdrátové připojení) odečet a sběr dat v reálném čase posílány do tzv. koncentrátorů, které tyto data vyhodnocují (vyhodnocuje se spotřeba a průtok energií). Zároveň jsou měřiče opatřeny IHD (In-Home Display), který umožňuje

spotřebitelům mít v reálném čase přehled o jejich aktuální spotřebě a zároveň mohou vidět cenu za spotřebu energií v daném období. Shromažďováním informací o spotřebě se může využít pro větší přehled chování zákazníků při spotřebě a pro monitorování systému a fakturaci zákazníků od dodavatele energie. Chytré měřiče jsou druhem vylepšených měřičů, které určují ukazatele spotřeby podrobněji než tradiční měřicí zařízení [5][6].

Chytré měřiče obvykle zaznamenávají energii v reálném čase a jsou vybaveny komunikačními prostředky pro přenos nahromaděných informací, které posléze pravidelně hlásí v určitých časových intervalech (čímž se liší od běžných měřičů, které nemají jak předat naměřené informace na dálku) za účelem monitorování spotřeby nebo zpracovávání údajů o spotřebě pro vyúčtování spotřebitelů [7].

Chytré měřiče jsou podobné běžným měřičům spotřeby, ale zahrnují řadu různých technologií, jako je čtení, shromažďování a ukládání naměřených údajů, upozornění na ztrátu energie a sledování kvality komunikačních zdrojů. Chytré měřiče umožňují obousměrnou komunikaci mezi měřičem a centrálním systémem. Tato AMI (Advanced Metering Infrastructure) se liší od AMR (Automatic Meter Reading) v tom, že umožňuje obousměrnou komunikaci mezi měřičem a dodavatelem energie [8]. Komunikace sítě může být buď bezdrátová (mobilní komunikace, Wi-Fi) nebo prostřednictvím pevných kabelových připojení (např. PLC – Programmable Logic Controller).

Je tedy mnoho výhod, kterými chytré měřiče disponují oproti běžným měřičům, ale samozřejmě jsou tu i nevýhody [6]:

- Mohou nastat problémy se synchronizací. Pokud domácnost vlastní starší chytrý měřič a změní dodavatele energie, je možné, že tento starý měřič, sice bude stále měřit, ale nebude schopen data posílat novému dodavateli;
- U bezdrátového spojení měřiče s koncentrátorem, které bývá zpravidla realizováno pomocí mobilních sítí, může dojít k inkonzistenci přenášených dat z důvodů špatného signálu v dané oblasti;
- Chytrý měřič, v případě nadměrného užívání energie, nemůže sám o sobě do spotřeby nijak zasahovat. Spotřebitel si musí sám analyzovat jeho spotřebu kontrolováním měřiče a vyvodit důsledky, jak šetřit energiemi (spotřebitelé s vědomím nadměrné spotřeby většinou mají sklony ke snížení spotřeby);
- Ne všichni dodavatelé podporují chytré měřiče;
- Náklady na chytrý měřič a na sběr dat pomocí mobilních sítí.

Chytré měřiče mohou mít do budoucna obrovské využití a benefity, ale v dnešní době zatím převyšují náklady nad výhodami.

Obr. 1.3: Příklad chytrého měřiče [31]

1.3 DLMS/COSEM

DLMS/COSEM je celosvětový standard, který slouží k měření elektřiny, plynu, vody, tepla a také slouží pro komunikaci s chytrými měřiči. Definuje objektově orientovaný datový model, aplikační protokol a komunikační profily specifické pro média.

DLMS/COSEM zahrnuje tři klíčové komponenty [9]:

- DLMS (Device Language Message Specification) – protokol aplikační vrstvy, který mění informace uchovávané v objektech na zprávy. DLMS je specifikace aplikace určená ke standardizaci zpráv přenášených přes distribuční linky. Reguluje: dálkové odečítání naměřených hodnot z měřicích zařízení, dálkové ovládání a také další služby pro měření jakéhokoli typu energie.
- COSEM (Companion Specification for Energy Metering) – objektový model rozhraní komunikačního zařízení pro měření energie, schopný popsat prakticky jakoukoli aplikaci. COSEM je specifikace, která odráží model rozhraní měřicích zařízení, který poskytuje reprezentaci jejich funkčnosti. Model rozhraní

používá objektově orientovaný přístup. Instance třídy rozhraní COSEM se nazývá „COSEM interface object“. Sada objektů konkretizovaných v logických zařízeních fyzického zařízení modeluje funkčnost měřicího zařízení, jak je vidět prostřednictvím jeho komunikačních rozhraní.

Model COSEM představuje měřič jako server používaný klientskými aplikacemi, které načítají data a poskytují řídicí informace. Klient může podporovat obchodní procesy veřejných služeb, zákazníků, provozovatelů měřidel nebo výrobců měřičů. Třídy rozhraní COSEM a jejich instance (objekty) lze snadno použít jak pro modelování případů použití měření, tak i pro modelování jakékoli aplikace. Objektové modelování je užitečný nástroj k formální reprezentaci jednoduchých nebo složitých dat. Každý aspekt dat je modelován pomocí atributu. Objekty mohou mít několik atributů a také metody k provádění operací s atributy. Objekty lze použít v kombinacích k modelování jednoduchých případů použití (např. čtení registrů), anebo složitějších případů použití, jako jsou schémata tarifů a fakturace nebo správa zátěže. V dnešní době je specifikováno 89 tříd rozhraní, ale je možné specifikovat až 65 535 tříd.

- OBIS (Object Identification System) – systém pojmenování objektů. OBIS definuje identifikační kódy (ID-kódy, čímž poskytuje jedinečný identifikátor pro všechna data v měřícím systému) pro běžně používané datové položky v zařízeních na měření energií. Tato část IEC 62056 specifikuje celkovou strukturu identifikačního systému a mapování všech datových položek na jejich identifikační kódy, které se používají k identifikaci:
  - logických názvů různých instancí integrovaných obvodů nebo objektů;
  - dat přenášených komunikačními linkami;
  - údajů zobrazených na měřicím zařízení.

  ID-kódy OBIS identifikují datové položky používané v zařízeních na měření energie v hierarchické struktuře pomocí šesti hodnotových skupin (skupiny A až F, kde OBIS kód se píše následovně – *A-B:C.D.E*F* a *A.B.C.D.E.F* ve zjednodušeném zápisu, který využívá Gurux DLMS Director – Obr. 2.7) [10][28]:
  - **skupina A** – identifikuje médium (typ energie), se kterým měření souvisí. S informacemi, které nesouvisejí s médii, se zachází jako s abstraktními daty (např. 0 = abstraktní objekt, 1 = elektřina, 6 = teplo, 7 = plyn, 8 = voda atd...);
  - **skupina B** – obecně identifikuje číslo měřicího kanálu (číslo vstupu měřicího zařízení, které má několik vstupů pro měření energie stejného typu nebo různých typů. Lze tak identifikovat údaje z různých zdrojů. Může také identifikovat komunikační kanál. A v některých případech

může identifikovat další prvky;
  - **skupina C** – identifikuje abstraktní nebo fyzické datové položky související s příslušným zdrojem informací, tedy fyzikální veličinu (např. elektrický proud, elektrické napětí, výkon, objem, teplota);
  - **skupina D** – identifikuje typy nebo výsledky zpracování fyzických veličin identifikovaných hodnotami ve skupinách A, C podle specifických algoritmů. Algoritmy mohou dodávat množství energie, poptávky atd.;
  - **skupina E** – určuje další zpracování nebo klasifikaci veličin identifikovaných hodnotami ve skupinách A až D (např. přepínání rozsahů);
  - **skupina F** – Identifikuje historické hodnoty dat podle skupin A až E v rámci různých zúčtovacích období. Pokud to není relevantní, lze tuto skupinu hodnot použít i pro další klasifikaci.

DLMS/COSEM je distribuován hlavně do zahraničí. Jedná se o protokol založený na koncepcích modelu OSI (Open Systems Interconnection), který reguluje výměnu dat mezi měřicími zařízeními a systémy sběru dat na základě architektury klient-server.

Výhody DLMS/COSEM [11]:

- široká nabídka rozhraní pro přenos dat: např. RS 232/485, PPP, PLC;
- definuje model rozhraní, který je platný pro jakýkoli typ zdroje energie. Systém založený na protokolu DLMS/COSEM je otevřený pro rozšíření přidáním nových funkcí beze změny stávajících služeb;
- vylepšení funkčnosti měřicích zařízení: zaznamenávání spotřeby, plánování tarifů, měření kvality elektřiny atd.;
- kontrolovaný a bezpečný přístup k naměřeným informacím (ověřený přístup pomocí hesla a s autentizací). Informace přenášené po komunikačních linkách mohou být dodatečně šifrovány;
- umožňuje vytvářet jednotné ovladače, díky nimž je možné komunikovat s měřicími zařízeními různých typů od různých výrobců;
- rozšíření mezi zahraničními měřicími zařízeními.

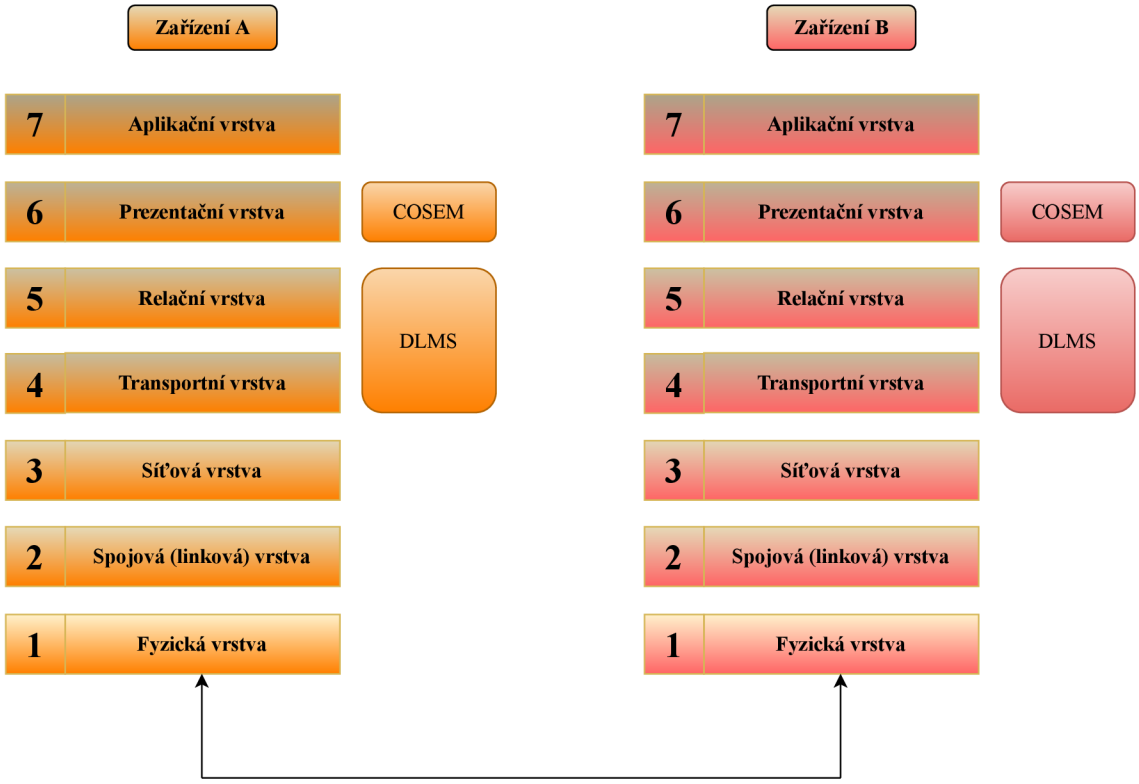
DLMS/COSEM má však i značné nevýhody [11]:

- problém úplnosti a „čistoty“ implementace normy. V praxi je dotazování měřiče s deklarovanou podporou DLMS jednoho výrobce buď omezeno programem dotazování od jiného výrobce základními parametry, nebo je jednoduše nemožné;
- velká složitost protokolu;
- extrémní neoblíbenost mezi domácími výrobci měřicích zařízení.

1.3.1 Model ISO/OSI

Síťový model OSI (Open Systems Interconnection) vypracovala organizace ISO (International Organization for Standardization). Prostřednictvím tohoto modelu mohou různá síťová zařízení navzájem komunikovat. Model definuje sedm základních vrstev. Každá vrstva plní v této interakci specifické funkce.

Model OSI byl vyvinut na konci 70. let za účelem podpory různých technik počítačových sítí, které v této době soutěžily o použití v hlavních národních sítích ve Francii, Velké Británii a Spojených státech. Tento model nedokázal v raných dobách internetu plně popsat síť a nezískal podporu architektů, což se později projevilo v méně normativním protokolu TCP/IP (Transmission Control Protocol/Internet Protocol), převážně pod vedením pracovní skupiny IETF (Internet Engineering Task Force) [12].



Obr. 1.4: Referenční model ISO/OSI

Referenční model ISO/OSI (Obr. 1.4) funguje jako abstraktní popis komunikace mezi zařízeními při přenosu dat. Protokol DLMS (má za úkol dialog mezi zařízeními a přenos dat) funguje na relační a transportní vrstvě, COSEM (formátování přenášených dat) na prezentační vrstvě [13].

1.3.2 Výměna informací v DLMS/COSEM

Výměna dat mezi měřicími zařízeními a systémy sběru dat pomocí objektového modelu rozhraní COSEM je založena na bázi klient-server (request/response), ve kterém systémy sběru dat hrají roli klienta a měřicí zařízení hrají roli serveru. Klient serveru odesílá požadavky na služby, a ten následně odesílá odpovědi na tyto požadavky. Server navíc může iniciovat nevyžádané žádosti o službu, aby informoval klienta o nastalých událostech.

Aplikace pro sběr dat a aplikace pro měření jsou modelovány jako jeden nebo více aplikačních procesů (APs). Komunikace proto v tomto prostředí probíhá vždy mezi klientem a serverovým AP (klientský AP požaduje služby a serverový AP je poskytuje).

Obecně platí, že APs na straně klienta a serveru jsou umístěny v samostatných zařízeních. Proto výměna zpráv probíhá prostřednictvím zásobníku protokolu.

Klíčové vlastnosti výměny dat pomocí DLMS/COSEM jsou následující [14]:

- přístup k měřicím zařízením může být umožněn různým stranám (klienti a třetí strany)
- mechanismy pro řízení přístupu ke zdrojům měřicího zařízení;
- bezpečnost a soukromí jsou zajištěny použitím kryptografické ochrany zpráv;
- klientský přístupový bod je schopen si vyměňovat data s jedním nebo více přístupovými body serveru současně;
- serverový přístupový bod je schopen si vyměňovat data s jedním nebo více klientskými přístupovými body současně. Objekty COSEM poskytují přehled o funkčnosti měřicích zařízení prostřednictvím jejich komunikačních rozhraní;
- výměna informací probíhá vzdáleně nebo lokálně. V závislosti na schopnostech měřicích zařízení, lze lokální a vzdálenou výměnu informací provádět současně bez vzájemného rušení;
- podle typu sítě (LAN – Local Area Network, WAN – Wide Area Network) mohou být použita různá komunikační média.

Klíčovým prvkem, který zajistí splnění výše uvedených požadavků, je AA (Application Association) – určení kontextů výměny dat.

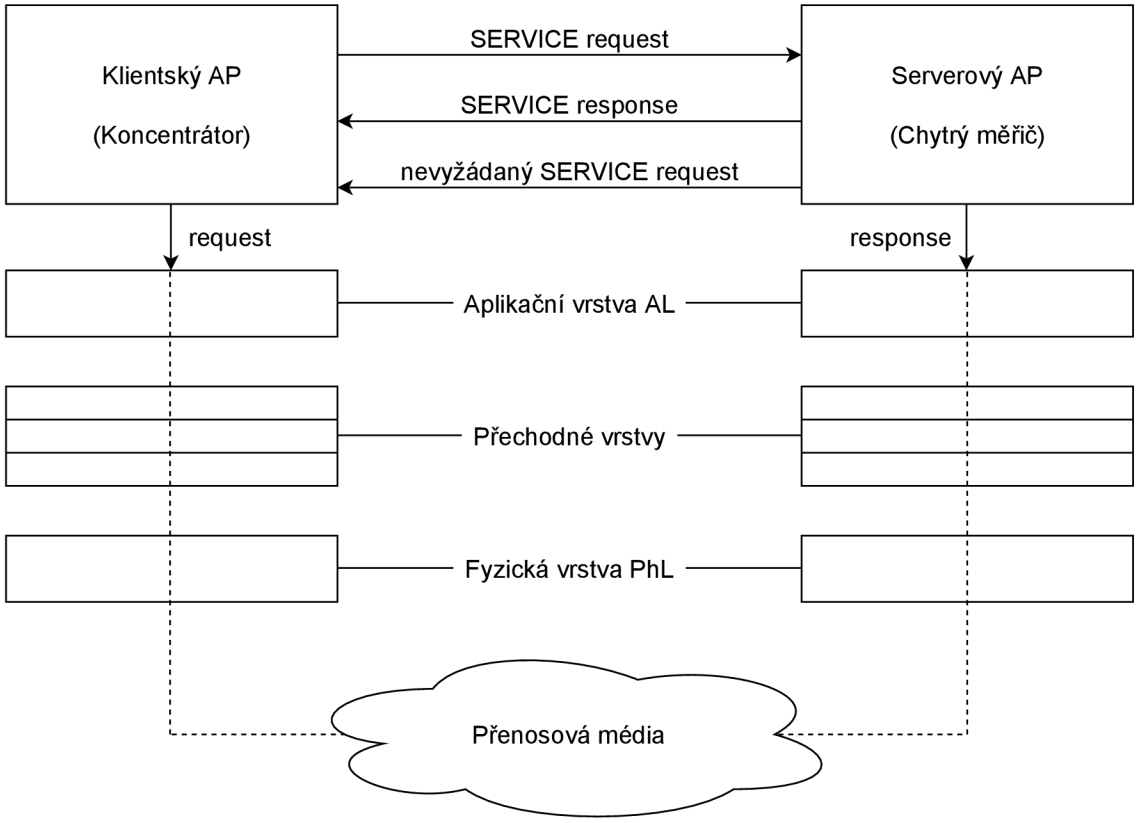
DLMS/COSEM využívá koncepty modelu OSI k uskutečnění výměny informací mezi měřiči a systémy sběru dat.

1.3.3 Komunikační model

Komunikace mezi APs (Application Processes) je tvořena komunikací mezi AEs (Application Entities). AE představuje komunikační funkce AP. Může existovat více sad komunikačních funkcí OSI v AP => jeden AP může být reprezentován více AEs. Na druhou stranu, každý AE reprezentuje pouze jediný AP.

Komunikační profily specifikují způsob modelování AP pomocí datového modelu DLMS/COSEM AL (Application Layer) a modelu COSEM.

Komunikační profily obsahují několik protokolových vrstev. Každá vrstva má svoji funkci, poskytuje služby své horní vrstvě a zároveň využívá služeb svých spodních vrstev. Klient a server COSEM APs využívají služeb DLMS/COSEM AL (nejvyšší vrstvy protokolu). Počet spodních vrstev a jejich typ závisí na použitém komunikačním médiu (každý profil charakterizují užité vrstvy protokolu a jejich charakteristiky).



Obr. 1.5: Komunikační model DLMS/COSEM

Obecný komunikační profil obsahuje [14]:

- nástroje pro kontrolu spojení (connection managers);
- fyzické vrstvy a MAC vrstvy specifické pro média;
- konvergenční vrstvy (díky kterým se váže vrstva MAC na DLMS/COSEM AL);
- DLMS/COSEM TL (Transport Layer);
- DLMS/COSEM AL (Application Layer);
- objektový model COSEM vytvářející AP.

Jakékoliv fyzické zařízení může podporovat více komunikačních profilů pro umožnění výměny dat užitím různých komunikačních médií (v takových případech je na klientském AP rozhodnout, jaké komunikační profily budou užity).

1.3.4 Logická jména a logické adresy

Logická jména a logické adresy jsou důležitými aspekty v komunikačních systémech DLMS/COSEM. Jméno identifikuje komunikující entity. Adresa identifikuje, kde tyto entity najít. Jména jsou mapována na adresy. Platí následující [5][14]:

- všechny entity DLMS/COSEM, včetně klientů, serverů a systémů třetích stran, musí být jedinečně pojmenovány podle jejich systémových názvů (systém title). Systémové názvy musí být trvale přiděleny;
- fyzická zařízení serveru mohou hostovat jedno nebo více LDs (logical devices). LDs musejí být jednoznačně identifikovány podle jejich LDN (Logical Device Name). LDs hostované stejným fyzickým zařízením sdílejí systémový název;
- každé fyzické zařízení musí mít příslušnou adresu. Záleží na komunikačním profilu a může se jednat o telefonní číslo, MAC adresu, IP adresu nebo o jejich kombinaci;
- fyzické adresy zařízení mohou být přednastaveny nebo mohou být přiřazeny během procesu registrace, což také zahrnuje vazbu mezi adresami a systémovými názvy;
- každý klient a server DLMS/COSEM (logické zařízení COSEM) je vázán na SAP (Server Access Point). SAPs jsou umístěny v podpůrné vrstvě DLMS/COSEM AL (Application Layer);
- systémový název musí jednoznačně identifikovat každou DLMS/COSEM entitu, která může být serverem, klientem nebo třetí stranou, a která má přístup k serverům prostřednictvím klientů. Systémový název musí mít délku 8 oktetů a musí být jedinečný;
- počáteční oktety (tj. počáteční 3 oktety zleva) by měly obsahovat třípísmenné ID výrobce. To stejné platí u prvních tří oktetů logického jména zařízení. Zbývajících 5 oktetů zajistí jedinečnost;
- před užitím kryptografických bezpečnostních algoritmů je nutný šifrovaný aplikační kontext – peerové si musí vyměňovat systémové názvy. V případě broadcastové komunikace odešle systémový název na server pouze klient;
- mechanismus identifikace uživatele klienta umožňuje serveru rozlišovat mezi různými uživateli na klientské straně a zaznamenávat jejich aktivity při přístupu k měřícímu zařízení.

1.3.5 Orientace spojení

DLMS/COSEM AL je orientovaná na připojení. Pro účely velmi jednoduchých zařízení, jednosměrně komunikujících zařízení a pro multicastové a broadcastové vysílání jsou předem zavedené AAs také povoleny. Úplná komunikační relace může např. zahrnovat pouze fázi výměny zpráv.

Komunikační relace se skládá ze tří fází [15]:

- nejprve je na aplikační úrovni vytvořeno spojení mezi aplikačními entitami AE (Application Entity), tzn. mezi klientem a serverem je vytvořeno AA (Application Association). Před založením AA musí být propojeny fyzické vrstvy PhL (Physical Layer) na straně klienta a serveru. Přechodné vrstvy mohou být propojeny, ale také nemusí. Každá vrstva může podporovat jedno či více připojení současně;
- jakmile je AA založena, může dojít k přenosu;
- po ukončení přenosu je AA uvolněna.

1.3.6 Application associations

Aby bylo možné přistupovat k COSEM objektům na serveru, musí být nejprve zřízena AA s klientem. Komunikace se odehrává pomocí tzv. APDU (Application Protocol Data Unit), které mohou být, pro zvýšení úrovně zabezpečení, zašifrovány. AAs jsou logická spojení mezi klientem a serverem, která mohou být založena buď na žádost klienta využívajícího služby ACSE (Association Control Service Element) na AL, anebo mohou být tato logická spojení předem zřízena. Může dojít buď k potvrzené nebo nepotvrzené AA [14].

COSEM LD (Logical Device) může podporovat jednu nebo více AAs, každou s jiným klientem. Každá AA určuje kontexty, ve kterých dochází k výměně informací.

Spojení, které bylo navržené klientem a přijaté serverem, bude znamenat potvrzenou AA za předpokladu, že [14][16]:

- uživatel klienta je známý serverem;
- navržený aplikační kontext je pro server přijatelný;
- navržený autentizační mechanismus je pro server přijatelný => proces autentizace je úspěšný;
- prvky xDLMS kontextu mezi klientem a serverem mohou být úspěšně vyjednány.

Může dojít i k nepotvrzené AA. Nepotvrzená AA je opět navržena klientem a požaduje se její přijetí ze strany serveru. Nedochází ovšem k žádnému vyjednávání. Nepotvrzené AAs se využívají zejména pro odesílání broadcastových zpráv z klientů na servery.

SN,LN (Short Name, Logical Name – objekty modelující AAs) určují jak název aplikačního kontextu, autentizačního mechanismu, xDLMS kontext, tak i sadu specifických přístupových práv k atributům COSEM objektů. Přístupová práva a bezpečnostní kontext se mohou v každém AA lišit.

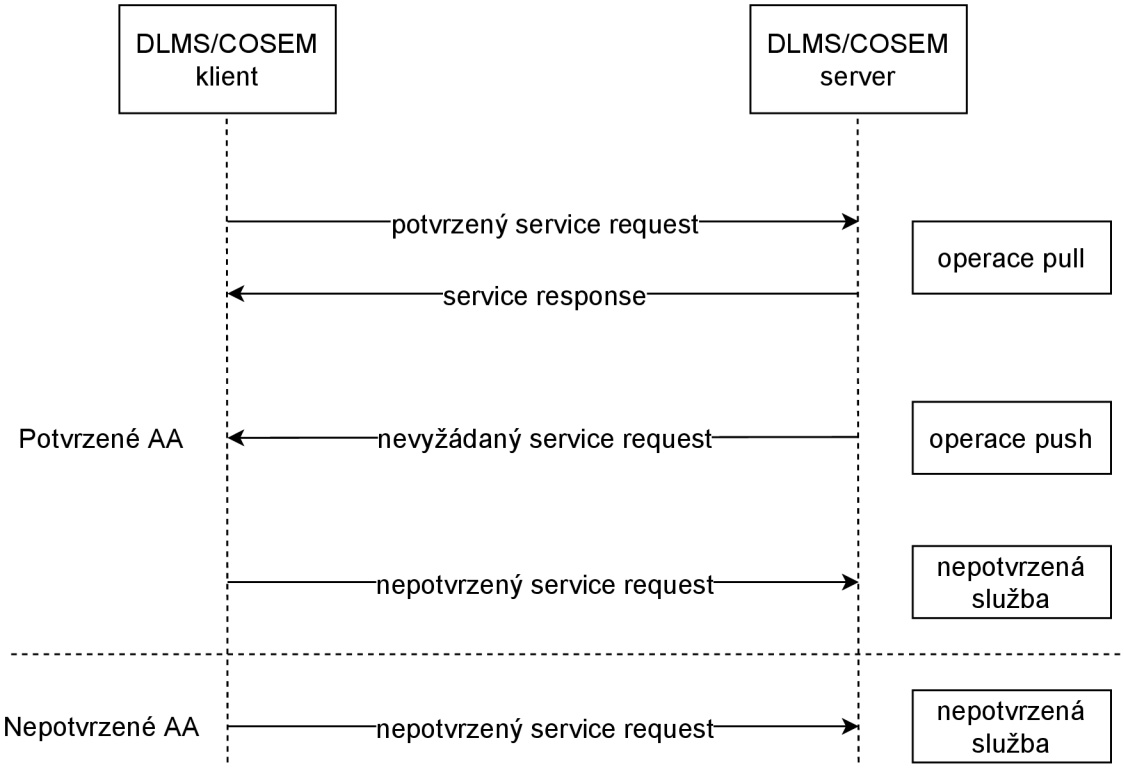
Služby a schopnosti xDLMS, které lze využít v určitém AA určuje xDLMS kontext.

V potvrzených AAs může klient odeslat potvrzené i nepotvrzené požadavky na služby, server ovšem pouze odesílá nevyžádané požadavky na služby klientovi. V nepotvrzených AAs může započít požadavky na služby pouze klient, a to jen nepotvrzené (server nemůže ani odpovídat, ani iniciovat požadavky na služby).

1.3.7 Zasílání zpráv

Zprávy, které si klient se serverem vyměňují v rámci DLMS/COSEM mohou být požadavky na služby (service request) a odpovědi na tyto požadavky (service response).

Vzory zpráv lze dělit podle možných situací, které mohou nastat v potvrzených AA a v nepotvrzených AA:



Obr. 1.6: Příklady zpráv DLMS/COSEM

- Zprávy v **potvrzeném** AA:

  - klient může serveru odeslat *service request* a server odpoví – operace *pull*;
  - klient může serveru odeslat *service request* a server neodpoví;
  - server může klientovi odeslat nevyžádaný *service request* – operace *push*.
- Zprávy v **nepotvrzeném** AA:
  - požadavky *service request* může posílat pouze klient a musejí být nepotvrzené. Server nemůže na tyto požadavky odpovídat a zároveň ani nemůže požadavky posílat klientovi.

1.3.8 Zabezpečení DLMS/COSEM

Měření energií pomocí chytrých měřičů sice představuje výhody, jako jsou efektivnější využití energií a jejich distribuce. Je to ovšem nová výzva z pohledu bezpečnosti, protože vše je spojeno skrz chytré sítě. Možné problémy:

- únik dat o měření energií zákazníků může usnadnit některé nezákonné činnosti (např. vloupání na základě zjištění, že v určité domácnosti je využití energií na minimální hodnotě nebo na nule, což by samozřejmě znamenalo, že se v dané domácnosti v tu chvíli nikdo nenachází);
- kvůli větší centralizaci bude snadnější ochromit vnitrostátní služby;
- nedostačující bezpečnost starších zařízení připojených do chytré sítě.

Aby se těmto problémům předešlo a riziko se minimalizovalo, DLMS/COSEM užívá bezpečnostní služby, jako je AA, výměna klíčů, zabezpečení obsahu zpráv, Anti-replay (sub-protokol IPsec, který znemožňuje provádět změny v paketech na cestě od zdroje k cíli) a RBAC (Role-based access control – omezení přístupu/práv méně kvalifikovaných osob pomocí rolí). Tyto bezpečnostní služby jsou umožněny díky kryptografickým algoritmům a protokolům:

- **AES-GCM** – důvěrnost dat
- **GMAC** – autentizace uživatelů a dat, integrita
- **ECDSA** – digitální podpis
- **SHA-1 a SHA-256** na podporu ECDSA při digitálním podepisování
- **AES Key Wrap** – aktualizace klíčů
- **ECDH** – dohoda na klíči

DLMS poskytuje tři sady zabezpečení (security suites), které splňují různé požadavky (viz Tab. 1.1).

DLMS/COSEM také obsahuje tzv. bezpečnostní úrovně (security levels):

- **No security (Lowest Level Security)** autentizace – užitečnost této úrovně spočívá v dovolení klientovi obdržet nějaké základní informace od serveru. Tento mechanismus nevyžaduje žádnou autentizaci => klient má přístup ke všem atributům COSEM objektů;

Tab. 1.1: Bezpečnostní sady DLMS

Security Suite	Šifrování	Dohoda na klíči	Digitální podpis	Hash	Přenos klíčů	Komprese
0	AES-GCM (128 bit)	-	-	-	AES-Wrap (128 bit)	-
1	AES-GCM (128 bit)	ECDSA (P-256)	ECDH (P-256)	SHA-256	AES-Wrap (128 bit)	V.44
2	AES-GCM (256 bit)	ECDSA (P-384)	ECDH (P-384)	SHA-384	AES-Wrap (256 bit)	V.44

- **LLS (Low Level Security) autentizace** – uživatelské jméno a heslo v plain text. Pro navázání AA je nutné, aby klient zadal a odeslal heslo, které je známé serverem. Pokud je heslo chybné nebo ho klient vůbec neodešle, nedojde k navázání AA;
- **HLS (High Level Security) autentizace** – tato metoda autentizace probíhá ve čtyřech krocích (viz Obr. 1.7). Pokud obě strany zjistí, že f(StoC) a *f(CtoS)* se shodují s očekávanou hodnotou, je autentizace úspěšná (f(nonce) může být vypočtena pomocí MD5 (nonce || secret)/SHA-1 (nonce || secret)/GMAC (secret || nonce)). Každý nonce je zabezpečený pomocí hesla (secret). Nonce zde má význam *number only used once* –> číslo lze použít pouze jednou, čehož se v kryptografii využívá pro zvýšení bezpečnosti.

Tab. 1.2: Autentizašní mechanismy podle ID

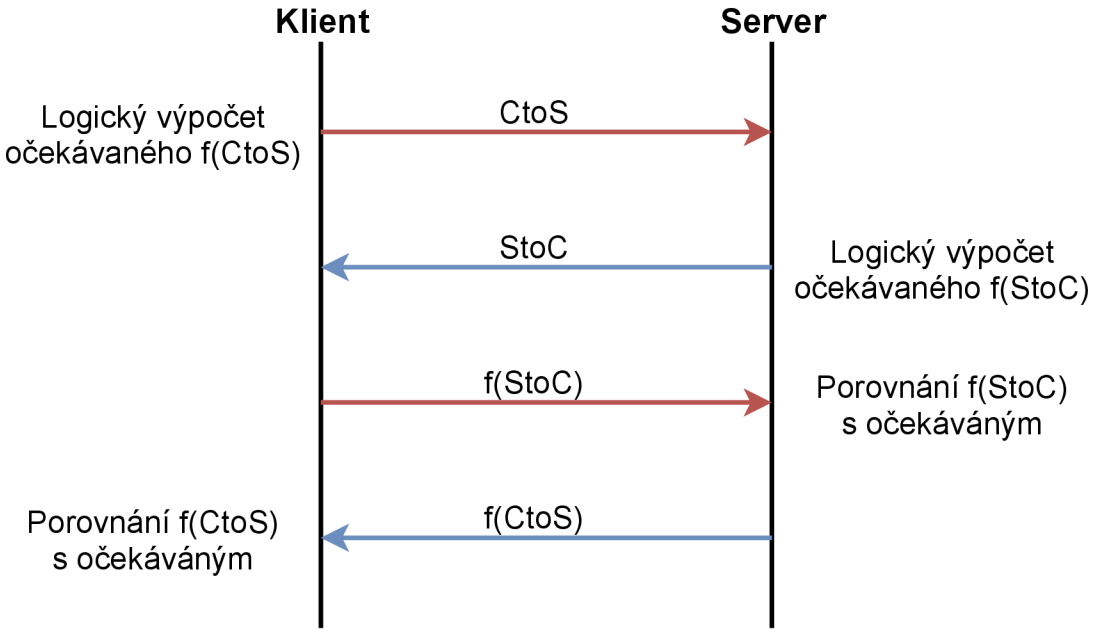
Druh autentizace	**ID mechanismu**
Bez zabezpečení	0
LLS	1
HLS	2
HLS s využitím MD5	3
HLS s využitím SHA-1	4
HLS s využitím GMAC	5
HLS s využitím SHA-256	6
HLS s využitím EC-DSA	7

Ovšem navzdory těmto bezpečnostním mechanismům má protokol i nějaké slabiny [29]:

- **Downgrade zabezpečení** – autentizace je volitelná a je indikována bitem v záhlaví zprávy, které je v plain textu –> útočník tedy může autentizaci

vypnout a upravit obsah zprávy (protiopatřením pro tento typ zranitelnosti je blokování všech zpráv s vypnutým bitem pro autentizaci);

- **Únik informací** – u šifrovaných APDU umožňuje protokol specifikovat typ služby v záhlaví (např. požadavek Get request), přičemž každá služba má pevně danou a známou strukturu, čehož může útočník může využít k narušení bezpečnosti (opatření – blokovat všechny APDU s označením explicitního typu služby);
- **Podvržený HLS server** – tato metoda spočívá v logickém výpočtu odpovědi na výzvu klienta útočníkem (podvržený server odchytí *CtoS* zprávu od klienta a vytvoří kopii, kterou mu pošle zpět jako *StoC*) . Podvržený server může klientovi odpovědět *CtoS* a *f(CtoS)* a donutit ho tak myslet si, že zná privátní klíč. Tento útok vyžaduje, aby byly APDU vyměňovány v plan textu (opatření – pokud se *StoC* rovná *CtoS*, tak klient takovou komunikaci odmítne);
- **Slovníkový útok na HLS** – pokud je provedena HLS autentizace pomocí MD5 nebo SHA-1, je možný slovníkový útok. Pokud útočník získá *CtoS* nonce (např. pomocí odchytávání provozu nebo podvrženého serveru), může se pokusit zjistit sdílený secret. Útočník tedy zná nonce a $h = f\ (nonce \mid\mid secret)$ – útočník tedy secret hádá pomocí slovníku, dokud nezíská h (protiopatření – neužívat MD5, SHA-1 a používat secret, který je náhodně generovaný a je tedy odolný vůči slovníkovému útoku);
- **Odpověď není vázána na žádost** – při MITM (Man In The Middle) útoku je útočník schopen narušit komunikaci klienta se serverem, odchytit a modifikovat žádost, kterou klient odesílá serveru, a následně ji poslat na server (server modifikovanou zprávu přijme pouze za předpokladu, že není vyžadována autentizace). Klient v tomto případě dostane od serveru odpověď a nemá možnost zjistit, že žádost byla upravena a server tedy vykonal něco jiného, než si klient přál (protiopatření – autentizace);



Obr. 1.7: HLS autentizace

1.3.9 Autentizační mechanismus

Autentizace subjektu je v komunikačních systémech zásadně důležitou bezpečnostní službou. Jedním z nejdůležitějších faktorů je ověření subjektu (zda žadatel o určitou identitu je ve skutečnosti tím, za koho se vydává). K dosažení tohoto cíle by již měl existovat vztah, který propojuje subjekt s identitou. Platí následující [15]:

- autentizace probíhá během založení AA;
- v potvrzené AA může klient autentizovat sebe sama, podobně mohou učinit i klient a server, kteří se dokážou autentizovat vzájemně;
- v nepotvrzené AA může sám sebe autentizovat pouze klient;
- v předem stanovených AAs není autentizace subjektů k dispozici;
- jakmile je AA založena, k atributům a metodám objektů COSEM lze přistupovat pomocí xDLMS služby, která se podřizuje bezpečnostnímu kontextu a přístupovým právům v dané AA;
- aplikační kontext určuje sadu ASE (Application Service Elements) přítomných v AL, syntaxi přenosu a zda je použito šifrování, či ne;
- bezpečnostní kontext je relevantní pouze tehdy, pokud aplikační kontext stanoví šifrování. Bezpečnostní kontext je spravován objekty „security setup“ a sestává z bezpečnostní politiky, bezpečnostního klíče atd.
- přístupová práva (odvíjejí se od role klienta a jsou přednastavena na serveru) určují možnosti klienta – zda může přistupovat k atributům a metodám

objektů COSEM v rámci AA.

1.3.10 Model měřicího systému DLMS/COSEM

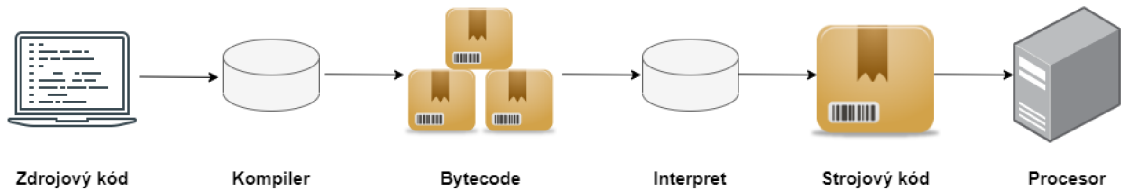
Měřicí zařízení se dá chápat jako fyzické zařízení obsahující jedno nebo více logických zařízení. Každé logické zařízení obsahuje řadu objektů COSEM, které zaručují funkčnost logického zařízení. Každé logické zařízení je jednoznačně identifikováno podle LDN (Logical Device Name). Server COSEM je strukturován do tří hierarchických úrovní: fyzické zařízení, logické zařízení a přístupné objekty COSEM [14].

Systémy sběru dat jsou modelovány jako sada klientských AP, které mohou být hostovány jedním nebo několika fyzickými zařízeními. Každý klient AP může mít různé role a různá přístupová práva udělená měřícím zařízením.

1.4 Programovací jazyk JAVA

JAVA je jedním z nejpoužívanějších programovacích jazyků na světě. Jedná se o interpretovaný, objektově orientovaný programovací jazyk inspirovaný jazyky C, C++, C#, Smalltalk a dalšími. Do tzv. mezikódu (kterému se také říká „bytecode“) je nejdříve přeložen zdrojový kód. V podstatě se jedná o binární kód, který má o hodně jednodušší sadu a přímo podporuje objektově orientované programování [17]. Kompilace kódu Java je znázorněna na Obrázku 1.8.

JVM (Java Virtual Machine) je abstraktní virtuální stroj s vlastní instrukční sadou a paměťovým prostorem. Pomocí JVM je zaručena tzv. „multiplatformnost“ => JVM je poskytnutý pro odlišné počítačové platformy, ať už se jedná o Windows, Mac OS, Linux, Android nebo různé typy procesorů.



Obr. 1.8: Kompilace kódu JAVA

Výhody interpreta a kompileru [17]:

- přenositelnost – na každém hardware, na kterém se nachází virtuální stroj poběží hotový program (zdrojový kód programu bude možné spustit);

- odhalení chyb ve zdrojovém kódu – jednoduše odhalíme chyby ve zdrojovém kódu pomocí kompilace do bytecode (pokud se ve zdrojovém kódu vyskytuje chyba, proces kompilace neproběhne korektně a je přerušen s popisem chyby, tím pádem může programátor pracovat efektivněji);
- málo zranitelný kód – není pro člověka jednoduché přečíst bytecode, ve kterém se aplikace šíří => zvýšená bezpečnost;
- jednoduchý vývoj – na rozdíl od C/C++ (kde se musí paměť uvolňovat „ručně“) zde provádí uvolňování alokované paměti provádí garbage collector;
- rychlost – virtuální stroj je schopen „cachovat“ výsledky běhu určitého programu a není proto nutné je zahazovat;
- stabilita – interpret je schopen (tím, že rozumí kódu) upozornit uživatele na vykonání rizikové operace a také může upozornit na chyby.

Nevýhody interpreta a kompileru:

- závislost na platformě – nejde přenést již zkompilovaný program z jedné platformy na druhou (je nutné jej na nové platformě opět zkompilovat);
- nemožnost úprav – po zkompilování do strojového kódu již nelze provádět úpravy (to jde vyřešit opětovnou kompilací);
- rychlost interpreta – interpret v nějakých případech není schopen plně využít výpočetního výkonu počítače a tím pádem může být interpretace velice pomalá;
- zranitelnost – z důvodu šíření programu ve formě zdrojového kódu je možné do něj zasahovat (upravovat jej nebo krást jeho části).

Java je distribuována ve třech edicích:

- Java SE – Standardní Edice;
- Java EE – Enterprise Edice;
- Java ME – Mikro Edice.

1.5 Raspberry Pi

Raspberry Pi je v podstatě malý počítač (základová deska), který byl vytvořen jako rozpočtové zařízení pro studium informatiky. Konečná cena produktu hrála při vývoji počítače hlavní roli, protože společnost při představení prototypu slíbila, že náklady na hotový produkt by neměly překročit 35 USD.

Tab. 1.3: Specifikace modelů Raspberry Pi.

Model	Rok vydání	CPU	GHz	Počet jader	RAM (MiB)	USB	Eth.	Wi-Fi	BT	Cena (USD)
A	2012	ARM117 6JZ-F	0,7	1	256	1	Ne	Ne	Ne	∼20
B	2012	ARM117 6JZ-F	0,7	1	512	2	Ano	Ne	Ne	∼35
B+	2014	ARM117 6JZ-F	0,7	1	512	4	Ano	Ne	Ne	∼30
A+	2014	ARM117 6JZ-F	0,7	1	256	1	Ne	Ne	Ne	∼25
2B	2015	ARM Cortex-A7	0,9	4	1024	4	Ano	Ne	Ne	∼35
Zero	2015	ARM117 6JZ-F	1	1	512	1	Ne	Ne	Ne	∼5
3B	2016	ARM Cortex-A53 x64	1,2	4	1024	4	Ano	Ano	4.1	∼40
Zero W	2017	ARM117 6JZ-F	1	1	512	1	Ne	Ano	4	∼10
3B+	2018	ARM Cortex-A53 x64	1,4	4	1024	4	Ano	Ano	4.2	∼35
4B	2019	ARM Cortex-A72 x64	1,5	4	1024 2048 6144 8192	4	Ano	Ano	5.0	35–75

Historie Raspberry Pi sahá až do května 2011, kdy David Braben, spoluzakladatel Raspberry Pi Foundation, představil neobvyklé zařízení o velikosti flash disku. První testovací verze počítače nebyly k dispozici masovému trhu a byly vyráběny pouze v omezeném množství.

V následujícím roce byl vydán masový model zařízení – tzv. „verze B“. Od té doby byla každý rok vydána nová aktualizovaná verze Raspberry Pi [18].

1.5.1 Specifikace a operační systémy

První modely Raspberry Pi byly založeny na jedno jádrovém procesoru s frekvencí 700 MHz. Později vývojáři vydali několik úprav [19].

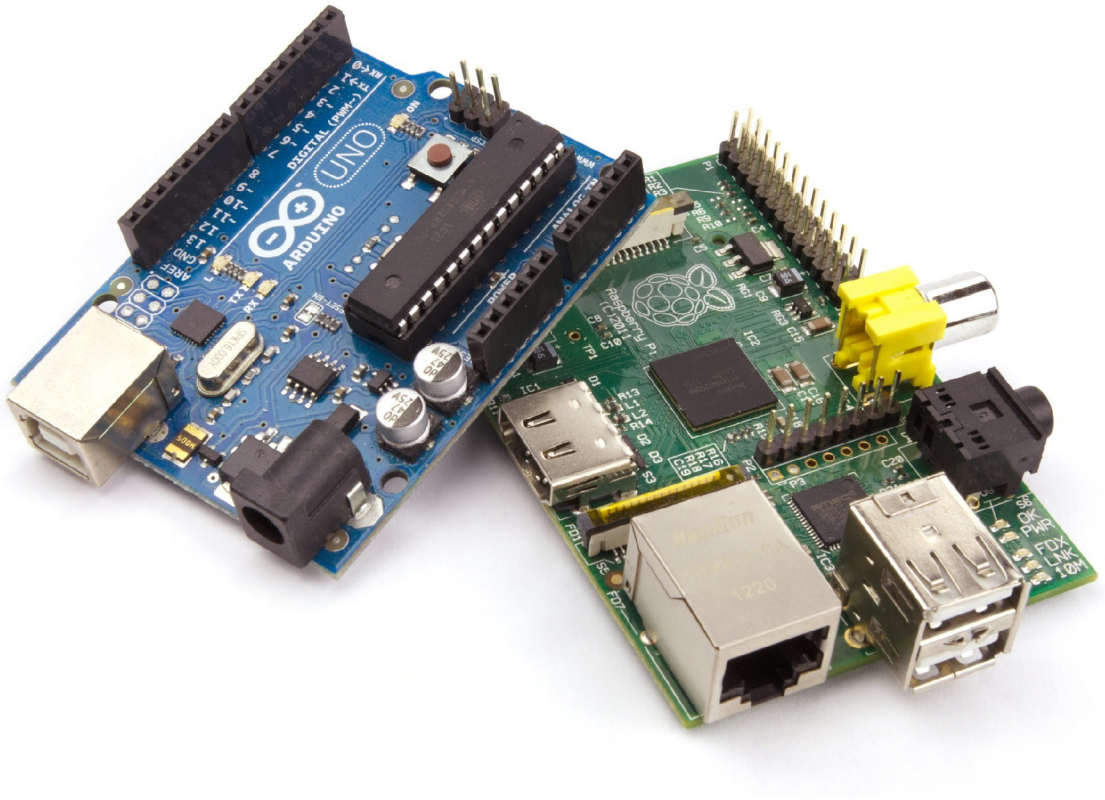
K ukládání informací se používají běžné paměťové karty SD nebo microSD. V závislosti na modelu se zařízení lišila nejen procesorem, ale také počtem USB portů, podporou Wi-Fi/Bluetooth a úpravou výstupu GPIO (General Purpose Input/Output).

Do desky Raspberry Pi je možné nainstalovat různé operační systémy. Novým uživatelům se nabízí platforma Raspbian, ale i mnohé další operační systémy. Nejnovější modely zařízení se čtyřjádrovým procesorem umožňují nainstalovat i Android a Windows 10.

Zatímco Raspberry Pi vypadá jako Arduino (viz Obr. 1.9 [30]), stále používá výrazně odlišný způsob provozu. Hlavním rozdílem je, že Arduino je miniřadič => dokáže mít v určitý moment spuštěn pouze jeden program, přičemž Raspberry Pi by se dalo z hlediska funkčnosti přirovnat spíše ke klasickému PC. Liší se také v možnosti připojení k internetu, kde Raspberry Pi lze jednoduše připojit k internetu skrze Ethernet nebo Wi-Fi pomocí integrované síťové karty, či USB. Arduino lze k internetu připojit pouze pomocí externího hardware a je nutná další konfigurace (není primárně určen k internetovému připojení).

Raspberry Pi, stejně jako běžné PC, běží pod jedním ze specializovaných operačních systémů. V závislosti na oblasti použití nebo osobních preferencích si může každý vybrat ten nejvhodnější. Níže je uveden seznam nejpopulárnějších operačních systémů pro Raspberry Pi se stručným popisem [18][20]:

- Raspbian – tento operační systém byl představen v roce 2015 jako hlavní pro Raspberry Pi. Je optimalizován pro procesory s architekturou AWP a nadále se docela aktivně vyvíjí. Operační systém je založen na Debianu GNU/Linux. Desktopové prostředí se skládá z LXDE (prostředí pro UNIX a další systémy kompatibilní s POSIX jako Linux a BSD) a správce oken Openbox (bezplatný správce pro X Window System). Distribuční sada obsahuje: program počítačové algebry Mathematica; upravenou verzi Minecraft PI; ořezanou verzi Chromu;
- Debian – open-source operační systém. Debian přichází s více než 59 000 předkompilovanými softwarovými balíčky a používá jádro Linux nebo FreeBSD. Standardní distribuce zahrnuje: desktopové prostředí GNOME se sadou nejpopulárnějších programů, jako je Firefox, LibreOffice, Evolution a další sady pro práci s multimédii. Je také možné instalovat obrázky s použitými desktopovými prostředími KDE, Xfce, LXDE, MATE a Cinnamon;
- Ubuntu – systém založený na Debianu GNU/Linux. Ubuntu je nejpopulárnější linuxovou distribucí pro webové servery. Distribuční sada obsahuje program pro prohlížení internetu; kancelářský balík, komunikační software atd.;



Obr. 1.9: Arduino (nalevo) a Raspberry Pi (napravo)

- Fedora – tento operační systém je založen na linuxové distribuci od slavné společnosti Red Hat. Distribuce zahrnuje LibreOffice, Mozilla Firefox a další software, který lze dodatečně nainstalovat prostřednictvím Centra aplikací GNOME;
- Arch Linux – bezplatná distribuce GNU/Linux. Vlastností tohoto systému je absence grafického instalátoru, pomocí čehož lze dobře trénovat dovednosti specialistů na Linux;
- Gentoo Linux – jedna z populárních distribucí GNU/Linux s flexibilní technologií správy balíčků. Systém poskytuje možnost maximální optimalizace pro konkrétní hardwarové řešení. Algoritmus správy balíčků usnadňuje implementaci pracovní stanice i serveru;
- RISC OS – operační systém speciálně vyvinutý pro procesory s architekturou ARM (Advanced RISC Machines). Vlastnosti jádra RISC OS umožňují systému provádět rychlejší spouštění ukládáním dat do paměti ROM (Read-Only Memory). Tento přístup také pomáhá chránit data v případě různých typů poruch a vlivu malwaru;
- další – OpenELEC, OSMC.

Na internetu lze kromě uvedených operačních systémů ještě najít mnoho dalších

úprav pro různé účely. Raspbian je ovšem hlavním prostředím pro Raspberry Pi.

1.5.2 Vlastnosti

Raspberry Pi není obyčejný počítač, je to malá deska s plošnými spoji, na které jsou nainstalovány komponenty. Deska Raspberry Pi se snadno vejde do dlaně dospělého člověka. Aby bylo zařízení kompletní, lze desku umístit do speciálního pouzdra. Existuje mnoho variant miniaturní systémové jednotky, prodávající se samostatně v různých online obchodech [18].

1.5.3 Rozšíření ve světě

Vývojáři Raspberry Pi oznámili prodej 12,5 milionů svých zařízení. Kdysi velmi odvážný a neznámý projekt se nyní řadí na třetí místo v popularitě v kategorii „univerzálních počítačů“ hned za Apple a Microsoft.

Většina prodávaných zařízení jsou nejvýkonnější modely 2B a 3B založené na více-jádrovém procesoru [19].

Cenová dostupnost, kompaktnost a flexibilita – to jsou vlastnosti, díky nimž si Raspberry Pi našel své místo na trhu. Počítače Raspberry Pi sice nejsou rychlé, ale pro mnoho lidí jsou klíčem k digitálnímu světu. Kromě toho je malá deska součástí komponentů pro domácí přehrávače médií, kancelářské stroje, nové startupy a mnoho dalších projektů.

2 Praktická část

V této části bakalářské práce bude popsán způsob implementace simulátoru chytrého měřiče a jeho dosavadní funkčnost. Pro implementaci DLMS serveru bylo využito vývojové prostředí Eclipse (programovací jazyk Java) s pomocí knihovny Gurux.DLMS [21], která umožňuje čtení chytrých měřičů kompatibilních s DLMS/COSEM. Pro možnost čtení hodnot ze simulátoru chytrého měřiče (klient) byl využit Gurux DLMS Director [22].

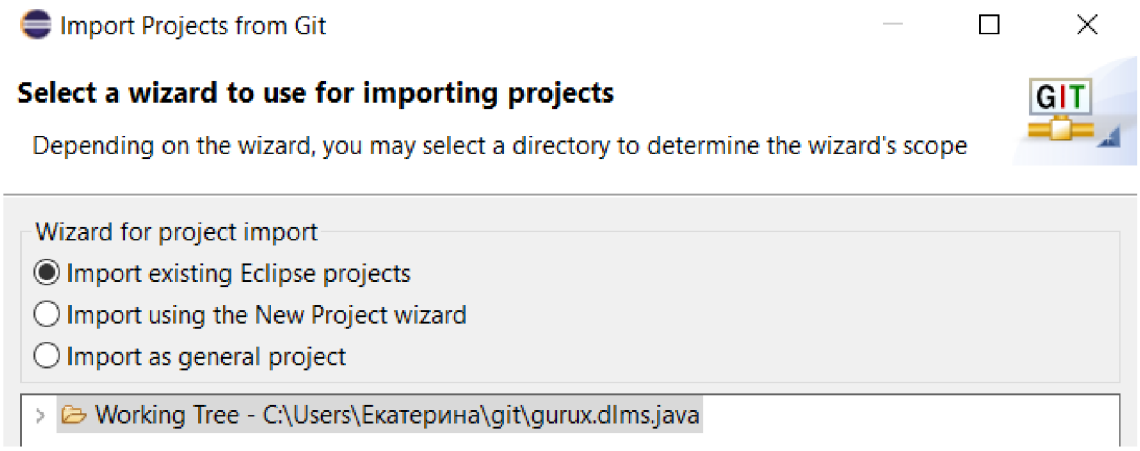
V první řadě byl importován projekt *gurux.dlms.server.example2.java* a bylo provedeno otestování funkčnosti DLMS serveru v rámci jednoho zařízení (laptopu), kde jako klient figuruje Gurux DLMS Director a jako server figuruje importovaný projekt v rámci vývojového prostředí Eclipse.

Po mírné úpravě importovaného projektu bylo možné provést testovací scénář a po tomto úspěšném testu mohl započít přesun projektu *gurux.dlms.server.example2.java*, exportovaného ve spustitelném JAR formátu – *smeter.jar*, do hardwarově omezeného prostředí Raspberry Pi. Projekt byl před přesunutím navíc upraven, aby simulátor chytrého měřiče vypisoval předem definované hodnoty a navíc byl přidán nový OBIS kód pro definici napětí baterie.

2.1 Vývoj aplikace

Pro úspěšné spuštění aplikace Gurux DLMS Server je nejprve nutné vytvořit prostředí. Pro zjednodušení práce s projektem je vhodné používat git. Výhoda využívání Git proti lokálnímu importu je zejména v jedoduchosti a jistotě, že projekt obsahuje všechny soubory, které jsou k běhu nutné. S využitím Git tedy není možné importovat neúplný projekt.

Importování repozitáře git do Eclipse: File –> Import –> Git –> Projects from Git –> Existing local repository –> gurux.dlms.java –> *gurux.dlms.server.example2.java*. Po výběru repozitáře (existing local repository) je nutné zvolit možnost “Import existing Eclipse projects” a zvolit potřebný projekt, který je třeba importovat, tedy *gurux.dlms.server.example2*.



Obr. 2.1: Git import

Gurux knihovna vyžaduje využívání Maven Dependencies. Maven je správce knihoven a zajišťuje stahování a vzdájemnou kompatibilitu knihoven mezi sebou. Jeho konfigurace je uskutečněna pomocí souboru *pom.xml*, který je nutné vložit do projektu. Tento soubor jednoznačně definuje, které knihovny a jaké verze má Maven v rámci projektu stáhnout a importovat.

Práce s *pom.xml* je jednoduchá. Maven neustále kontroluje změny v souboru *pom.xml*, v případě změny aktualizuje knihovny, případně vytvoří nové nebo je odstraní. Je tedy možné mít zálohovaný pouze soubor *pom.xml*, nikoli celé knihovny (výhodné při využívání Git).

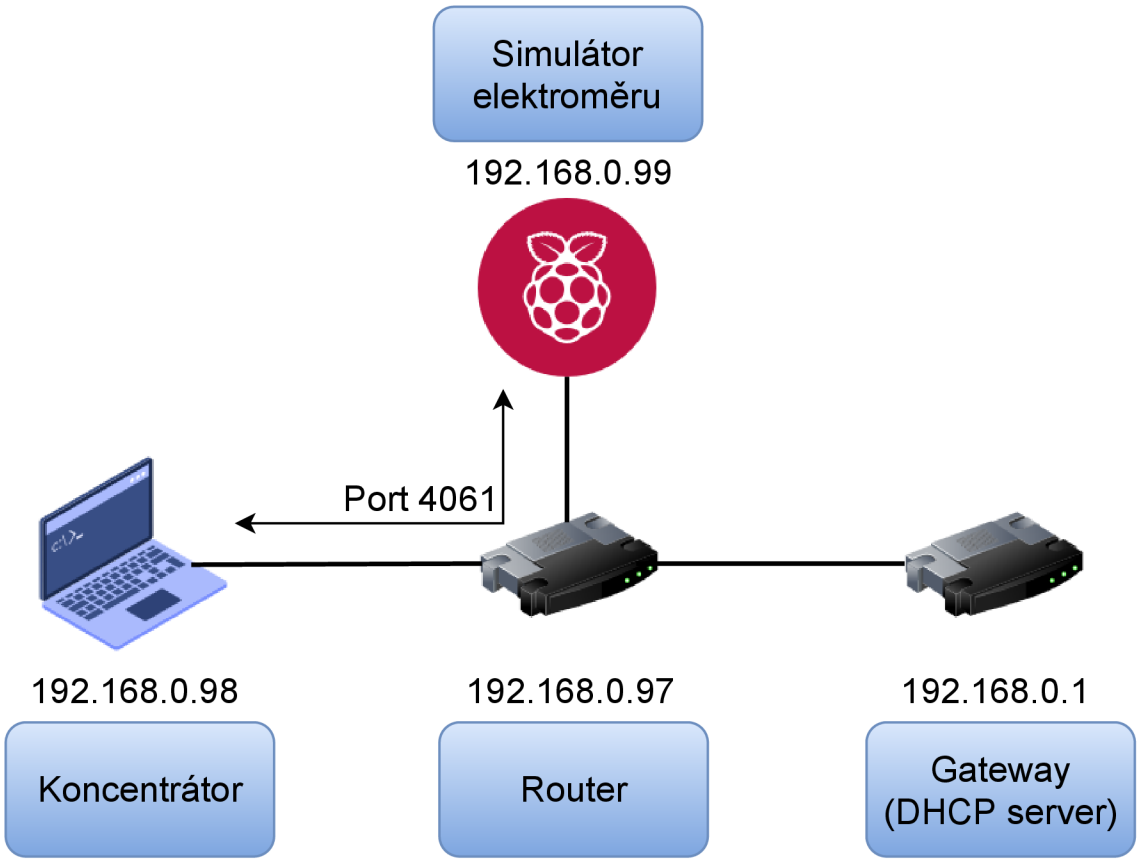
```
<dependencies>
  <dependency>
    <groupId>org.gurux</groupId>
    <artifactId>gurux.net</artifactId>
    <version>1.0.21</version>
  </dependency>
  <dependency>
    <groupId>org.gurux</groupId>
    <artifactId>gurux.dlms</artifactId>
    <version>4.0.24</version>
  </dependency>
</dependencies>
```

Obr. 2.2: Výstup pom.xml dependencies

2.2 Testovací prostředí

Celý testovací scénář probíhal na reálných zařízeních. Před realizací samotného testovacího scénáře bylo nutné provést ověření správné funkčnosti (tj. propojení Gurux DLMS directoru s Gurux DLMS serverem). Vlastní implementace simulátoru chytrého měřiče je uskutečněna ve vývojové platformě Eclipse, aplikace Gurux DLMS Director je spuštěna nativně. Tyto dvě aplikace komunikují pomocí TCP/IP dle Tab. 2.1.

Topologie sítě byla pro umožnění testovacího scénáře zapojena dle Obr. 2.3



Obr. 2.3: Schéma zapojení

Zařízení pro uskutečnění scénáře byly nakonfigurovány na přijetí IP adresy od DHCP serveru, bylo tedy nutné zjistit jejich IP adresy. To lze uskutečnit hned několika způsoby. V tomto případu byla zvolena metoda skenování místní sítě pomocí open-source skeneru sítě *Angry IP Scanner* [23].

Nyní je již možné vypsat IP adresy jednotlivých zařízení v místní síti pomocí *Angry IP Scanner* (viz Tab. 2.1).

Tab. 2.1: IP adresy jednotlivých zařízení

Zařízení	IP adresa
Gateway	192.168.0.1
Router	192.168.0.97
Laptop	192.168.0.98
Raspberry Pi	192.168.0.99

2.3 Vlastní implementace simulátoru chytrého měřiče

V této části bude probráno vše potřebné k implementaci a zprovoznění simulátoru chytrého měřiče. K provedení tohoto byl využit programovací jazyk Java z důvodu jeho multiplatformnosti (platformou může být myšlen jak operační systém – např. Windows, Mac OS atd. – tak i hardwarová platforma – např. x86, ARM atd.). K jednoduché správě tohoto simulátoru elektroměru byl použit Gurux DLMS Director z důvodu jeho efektivnosti a zároveň rychlé a jednoduché kofiguraci.

Pro implementaci bylo nejprve nutné z knihovny Gurux.DLMS [21] importovat do vývojové platformy Eclipse projekt *gurux.dlms.server.example2*, který představuje DLMS server.

Po spuštění metody *main*, která je vytvořena v rámci třídy *GuruxDlmsServerExample.java* (viz Obr. 2.5), dojde k vytvoření komponent SN a LN DLMS serveru na portech 4060, 4061, 4062 a 4063. Nyní je server připraven na propojení s klientem.

```
Short Name DLMS Server in port 4060
Logical Name DLMS Server in port 4061
Short Name DLMS Server with IEC 62056-47 in port 4062
Logical Name DLMS Server with IEC 62056-47 in port 4063
Press Enter to close.
```

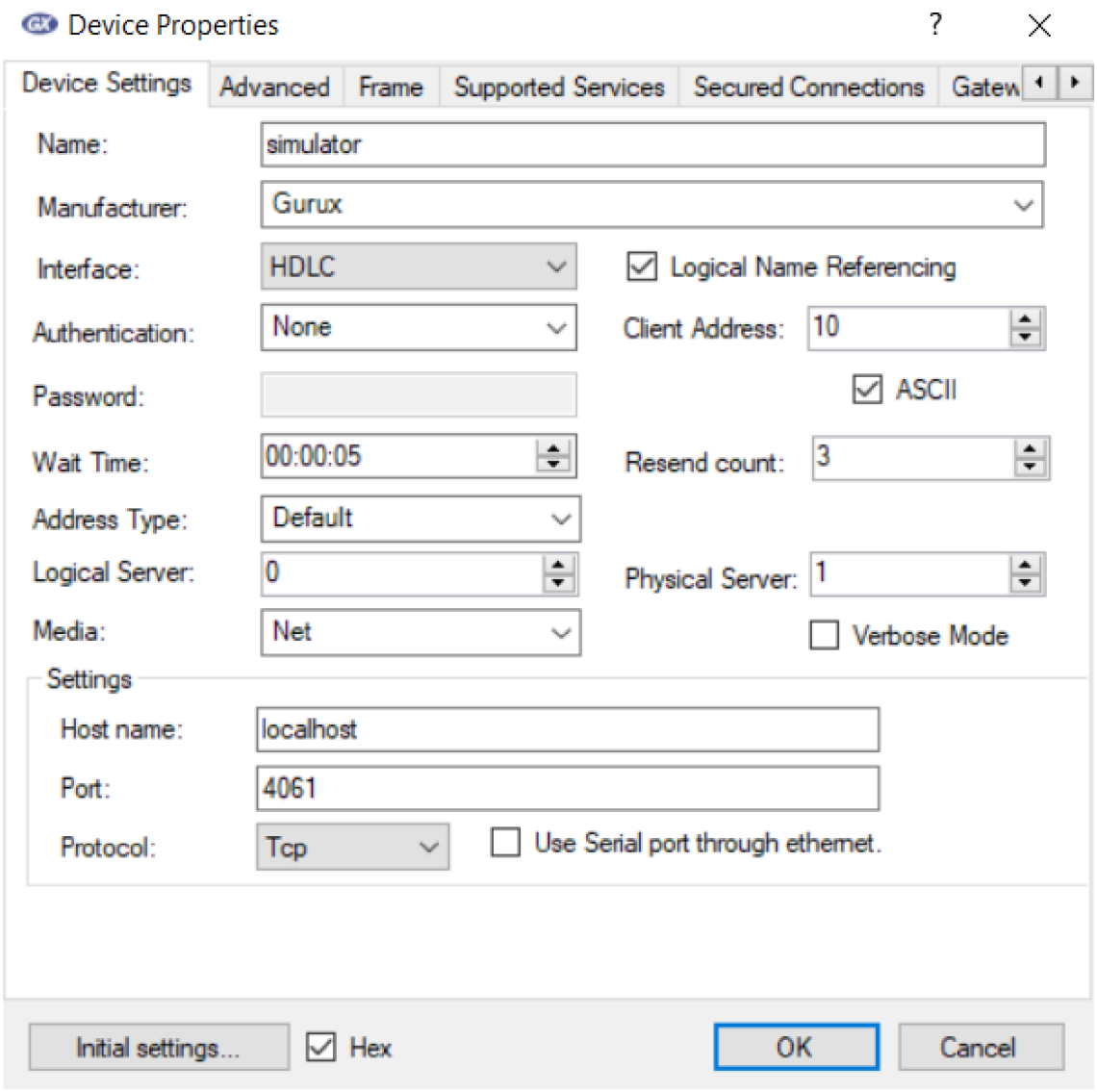
Obr. 2.4: Výpis konzole po spuštění metody main

```
public static void main(String[] args) throws Exception {
    // Create servers and start listen events.
    try {
        // Create Gurux DLMS server component for Short Name
        GXDLMSServerSN SNServer = new GXDLMSServerSN();
        SNServer.initialize(4060);
        System.out.println("Short Name DLMS Server in port 4060");
        // Create Gurux DLMS server component for Logical Name
        GXDLMSServerLN LNServer = new GXDLMSServerLN();
        LNServer.initialize(4061);
        System.out.println("Logical Name DLMS Server in port 4061");
        // Create Gurux DLMS server component for Short Name
        GXDLMSServerSN_47 SN_47Server = new GXDLMSServerSN_47();
        SN_47Server.initialize(4062);
        System.out.println(
                "Short Name DLMS Server with IEC 62056-47 in port 4062");
        // Create Gurux DLMS server component for Logical Name
        GXDLMSServerLN_47 LN_47Server = new GXDLMSServerLN_47();
        LN_47Server.initialize(4063);
        System.out.println(
                "Logical Name DLMS Server with IEC 62056-47 in port 4063");

        System.out.println("Press Enter to close.");
        while (System.in.read() != 13) {
            System.out.println("Press Enter to close.");
        }
        /// Close servers.
        System.out.println("Closing servers.");
        SNServer.close();
        LNServer.close();
        SN_47Server.close();
        LN_47Server.close();
        System.out.println("Servers closed.");
    } catch (RuntimeException ex) {
        System.out.println(ex.getMessage());
    }
}
```

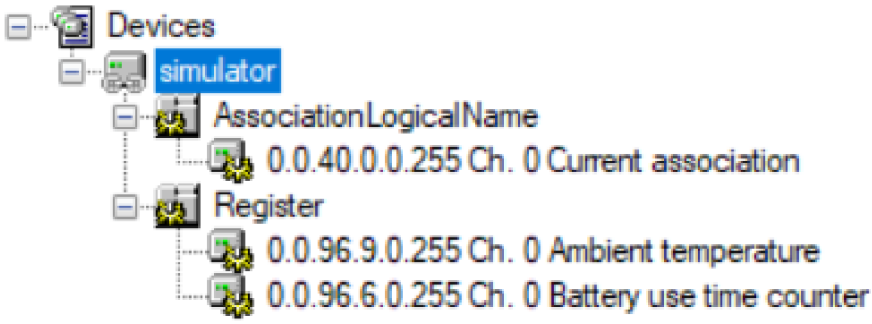
Obr. 2.5: Hlavní metoda main – vlastní implementace simulátoru chytrého měřiče

Jakmile je DLMS server spuštěný, lze ho propojit s klientem, tedy s Gurux DLMS Directorem, ve kterém je možné vytvořit koncentrátor, který bude komunikovat se simulátorem chytrého měřiče přes *port 4061*, pomocí „File –> Add Device“ (viz Obr. 2.6).



Obr. 2.6: Nastavení Gurux DLMS Director

Následně už jen zbývá propojit server s klientem, čehož se dá docílit kliknutím pravým tlačítkem myši na nově vytvořený objekt *simulator* v Gurux DLMS directoru a zvolením možnosti „Connect“. Po propojení se objeví dotazovací okénko, které vyžaduje potvrzení pro inicializaci objektů z chytrého měřiče. Na Obr. 2.7 jdou vidět již načtené objekty.



Obr. 2.7: Výpis Gurux DLMS Directoru po připojení

Nyní již klient komunikuje se serverem. Tímto tedy byla ověřena základní funkčnost spojení klient-server v prostředí notebooku (DLMS server spuštěn přes Eclipse a DLMS klient/koncentrátor přes Gurux DLMS Director).

```
Client Connected.
Client Read value from 0.0.40.0.0.255 attribute: 3.
Client Read value from 0.0.40.0.0.255 attribute: 4.
Client Read value from 0.0.40.0.0.255 attribute: 5.
Client Read value from 0.0.40.0.0.255 attribute: 6.
Client Read value from 0.0.40.0.0.255 attribute: 7.
Client Read value from 0.0.40.0.0.255 attribute: 8.
Client Read value from 0.0.40.0.0.255 attribute: 9.
Client Read value from 0.0.40.0.0.255 attribute: 10.
Client Read value from 0.0.40.0.0.255 attribute: 11.
Client Read value from 0.0.96.9.0.255 attribute: 2.
Client Read value from 0.0.96.6.0.255 attribute: 3.
Client Read value from 0.0.96.6.0.255 attribute: 2.
Client Disconnected.

Closing servers.
Servers closed.
```

Obr. 2.8: Čtení hodnot z chytrého měřiče

Na Obr. 2.8 jsou vidět atributy OBIS kódů, na které které server dotazuje podle definice jednotlivých objektů specifikovaných v *gurux.dlms.server.example2.java* (Obr. 2.9) – tedy OBIS kód pro měření teploty (*0.0.96.9.0.255*), počítadlo cyklů baterie chytrého měřiče (*0.0.96.6.0.255*) a k tomu ještě atributy objektu *Association Logical Name* (*0.0.40.0.0.255*), o kterých jsou podrobnější informace v kap. 2.5.

```
// Add objects of the meter.
temperature = (GXDLMSRegister) getItems().findByLN(ObjectType.REGISTER,
    "0.0.96.9.0.255");
if (temperature == null) {
  // CPU temperature.
  temperature = new GXDLMSRegister("0.0.96.9.0.255");
  temperature.setScaler(1);
  temperature.setUnit(Unit.TEMPERATURE);
  temperature.setValue(100);
  // Temperature is send using signed byte.
  temperature.setDataType(2, DataType.INT8);
  getItems().add(temperature);
}
// Battery use time counter
GXDLMSRegister r = (GXDLMSRegister) getItems().findByLN(ObjectType.REGISTER,
    "0.0.96.6.0.255");
if (r == null) {
  r = new GXDLMSRegister("0.0.96.6.0.255");
  // Battery use time counter is send using UInt16.
  r.setDataType(2, DataType.UINT16);
  getItems().add(r);
}
batteryUseTimeCounter = new GXBatteryUseTimeCounter(r);
batteryUseTimeCounter.start();
////////////////////////////////////////////////////////////////////////
// Server must initialize after all objects are added.
super.initialize();
```

Obr. 2.9: Definice OBIS kódů

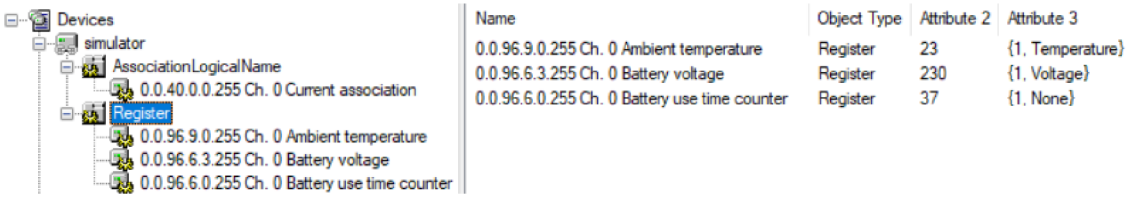
Nyní je třeba upravit kód, aby DLMS server směrem ke koncentrátoru posílal určité, ručně naprogramované, hodnoty (teplota 23 a nově přidaná hodnota napětí baterie na 230). Všechny změny kódu se odehrávají ve třídě *GXDLMSBase.java*. Tím pádem je nutné přidat nový OBIS kód pro definici napětí baterie – *0.0.96.6.3.255* [24]. Ve výchozím nastavení je DLMS server naprogramován na generování náhodných hodnot pro teplotu, které při každém dotazu na hodnotu teploty odesílá náhodně vygenerované. Za tohle odpovídá funkce *UpdateTemperature()*, ve které je implementovaný generátor náhodných čísel – ten je tedy nutné odebrat a změnit hodnotu metody *temperature.setValue()* na 23. Tímto je vyřešeno náhodné generování teploty a nyní už lze přejít k definici nového OBIS kódu – viz Obr. 2.10, na kterém lze sledovat změnu u definice teploty (tato změna spočívá především ve změně *DataType* teploty z INT8 na UINT16, protože datový typ INT8 může držet pouze hodnoty o velikosti 8 bitů se znaménkem => hodnoty -128 až 127, což by nestačilo pro nově přidané napětí) a přídání OBIS kódu pro napětí baterie.

```
// Add objects of the meter.
temperature = (GXDLMSRegister) getItems().findByLN(ObjectType.REGISTER,
    "0.0.96.9.0.255");
if (temperature == null) {
  // CPU temperature.
  temperature = new GXDLMSRegister("0.0.96.9.0.255");
  temperature.setScaler(1);
  temperature.setUnit(Unit.TEMPERATURE);
  temperature.setValue(23);
  temperature.setDataType(2, DataType.UINT16);
  getItems().add(temperature);
}

voltage = (GXDLMSRegister) getItems().findByLN(ObjectType.REGISTER,
        "0.0.96.6.3.255");
    if (voltage == null) {
      // Battery voltage.
        voltage = new GXDLMSRegister("0.0.96.6.3.255");
        voltage.setScaler(1);
        voltage.setUnit(Unit.VOLTAGE);
        voltage.setValue(230);
        voltage.setDataType(2, DataType.UINT16);
      getItems().add(voltage);
    }
```

Obr. 2.10: Definice OBIS kódů po přídání napětí baterie

Všechny potřebné úpravy kódu jsou tímto dokončeny a je možné započít novou simulaci. Předtím je ovšem nutné v Gurux DLMS Directoru vymazat položky *AssociationLogicalName* a *Register* (kliknutím pravého tlačítka myši –> Delete), aby se při dalším propojení mohl načíst do registru i nový OBIS kód. Když teď dojde k opětovnému propojení koncentrátoru se simulátorem chytrého měřiče, objeví se dotazovací okénko, které vyžaduje potvrzení pro inicializaci objektů z chytrého měřiče. Po potvrzení se načtou všechny OBIS kódy včetně nově definovaného napětí baterie a po přečtení hodnot z chytrého měřiče (*Read Values*) lze sledovat všechny nadefinované hodnoty (teplotu *23 °C*, napětí baterie *230 V*) v čase *37 s* od spuštění chytrého měřiče (Battery use time counter –> 37), viz Obr. 2.11.



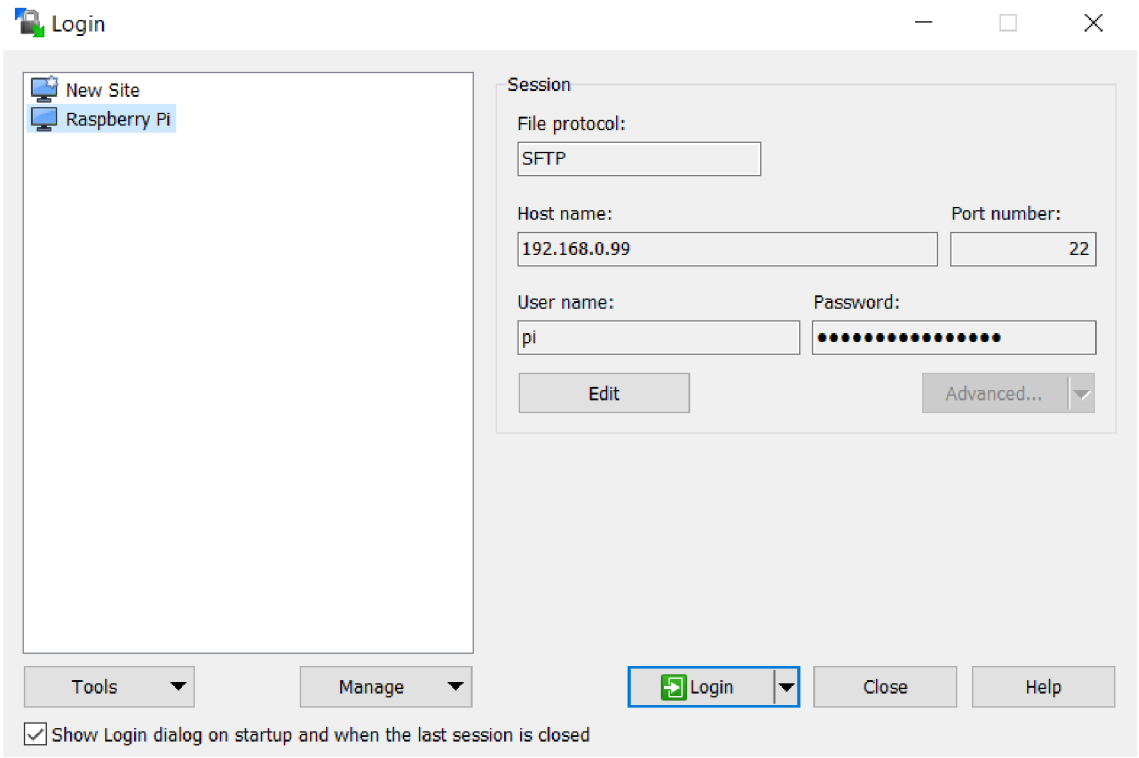
Obr. 2.11: Čtení hodnot ze simulátoru chytrého měřiče

2.4 Realizace

Vlastní implementace simulátoru chytrého měřiče zahrnuje komunikaci serveru s klientem na portu 4061 v rámci jednoho zařízení. Klient je schopen ze serveru číst jednotlivé, ručně nakonfigurované, atributy. Byly definovány tři OBIS kódy, kde každý je spojen s jednou hodnotou na simulátoru chytrého měřiče. Gurux DLMS director podlě těchto OBIS kódů správně vypisuje přečtené veličiny.

Díky ověřené funkčnosti komunikace klient-server je možné přejít k dalšímu bodu, a to k přenesení upraveného Eclipse projektu *gurux.dlms.server.example2* do hardwarově omezeného prostředí – Raspberry Pi. To lze provést jednoduše – prvním krokem bude exportování projektu ve vývojovém prostředí Eclipse kliknutím na složku projektu v *Package Explorer* pravým tlačítkem myši –> Export –> Java/Runnable JAR file –> v *Launch Configuration* vybrat exportovaný projekt, vybrat, kde projekt bude uložen a nakonec zvolit možnost *Extract required libraries into generated JAR* –> Finish.

Tímto je projekt exportován do spustitelného souboru *smeter.jar*. Přenesení na Raspberry Pi lze uskutečnit hned několika způsoby (WinSCP, PSCP atd.). Pro účel bakalářské práce byla použita metoda bezpečného přenosu pomocí protokolu SFTP (Secure File Transfer Protocol). Tento protokol je k dispozici v open-source FTP klientu WinSCP (viz Obr. 2.12) [25]. Ve WinSCP lze uložit šablonu pro příští možnou potřebu. Pro Raspberry Pi jsou ve výchozím nastavení přihlašovací údaje následující: User Name – *pi*, Password – *raspberry*. Následně už jen stačí potvrdit přihlášení k Raspberry Pi pomocí tlačítka *Login*.



Obr. 2.12: WinSCP

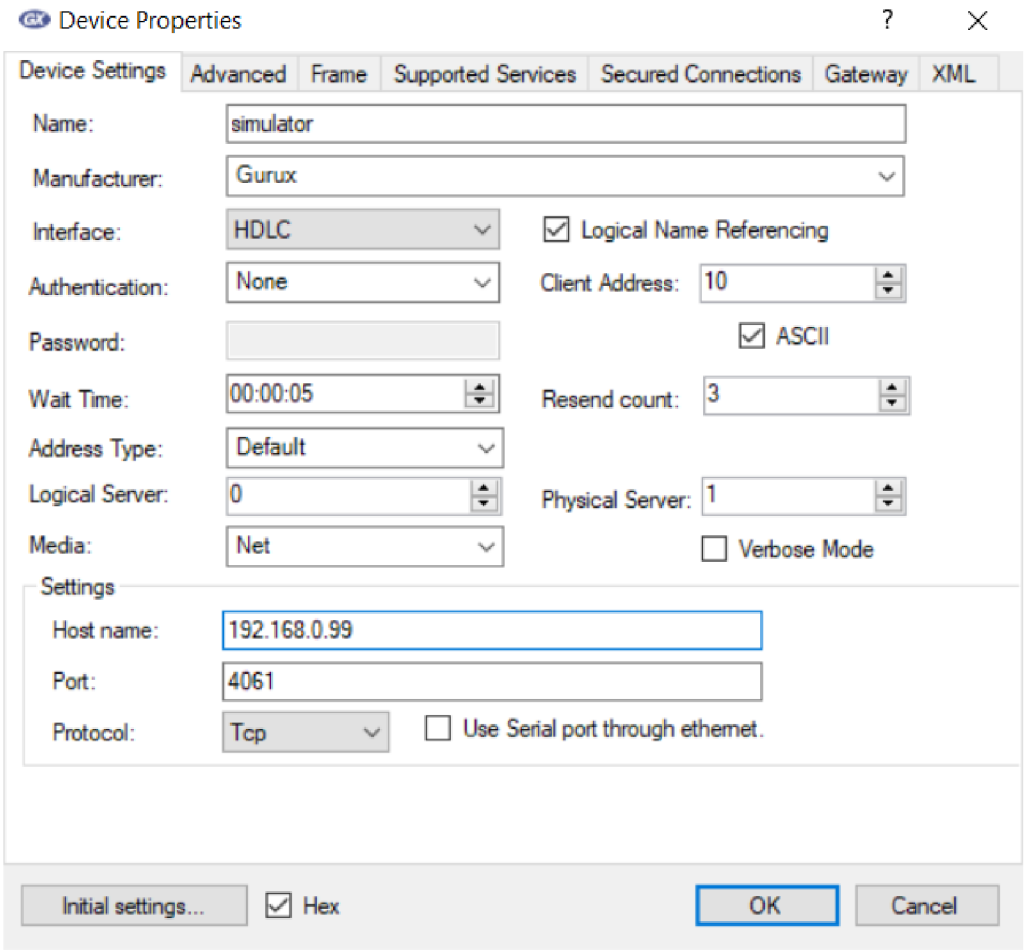
Před možností přenosu exportovaného Eclipse projektu je nutné vytvořit složku, do které bude projekt přenesen. Skrze WinSCP tohle přímo bohužel není možné, protože *root* složka Raspberry Pi nedovoluje vytvoření nové složky kvůli omezeným právům.

Pomocí ikony *Open session in PuTTy* nebo pomocí klávesové zkratky *Ctrl+P* lze otevřít SSH relaci přes PuTTy.

Po otevření SSH relace, zadáním následujících příkazů, bude vytvořena složka *simulator* a budou upravena práva této složky na *777* (to znamená, že složku *simulator* bude moci upravovat kdokoliv, což sice představuje bezpečnostní hrozbu, ale v tomto konkrétním případě to lze akceptovat, jelikož se jedná o testovací prostředí).

```
pi@raspberry:~ $ cd /
pi@raspberry:/ $ sudo su
pi@raspberry:/# mkdir simulator
pi@raspberry:/# chmod 777 simulator
```

Do složky *simulator* nyní lze pomocí WinSCP přenést Eclipse projekt *smeter.jar*. Následně je nutné přenastavit vlastnosti zařízení *simulator* v Gurux DLMS Directoru (projekt *simulator.gxc*), kde je třeba změnit *Host name* z *localhost* na IP adresu Raspberry Pi – *192.168.0.99* (viz Obr. 2.13).



Obr. 2.13: Přenastavení Gurux DLMS Directoru pro komunikaci s Raspberry Pi

Teď jen zbývá otestovat, zda vše funguje. Před samotným spuštěním projektu je třeba instalace Java (OpenJDK) na Raspberry Pi [26]. Standardní Raspbian repozitáře zahrnují dva různé Java balíčky – JRE (Java Runtime Environment) a JDK (Java Development Kit). Instalace nejnovější verze se provádí následujícími příkazy:

```
pi@raspberry:/# apt update
pi@raspberry:/# apt install default-jdk
```

Verze nainstalovaného *OpenJDK* lze ověřit pomocí příkazu:

```
pi@raspberry:/# java -version
```

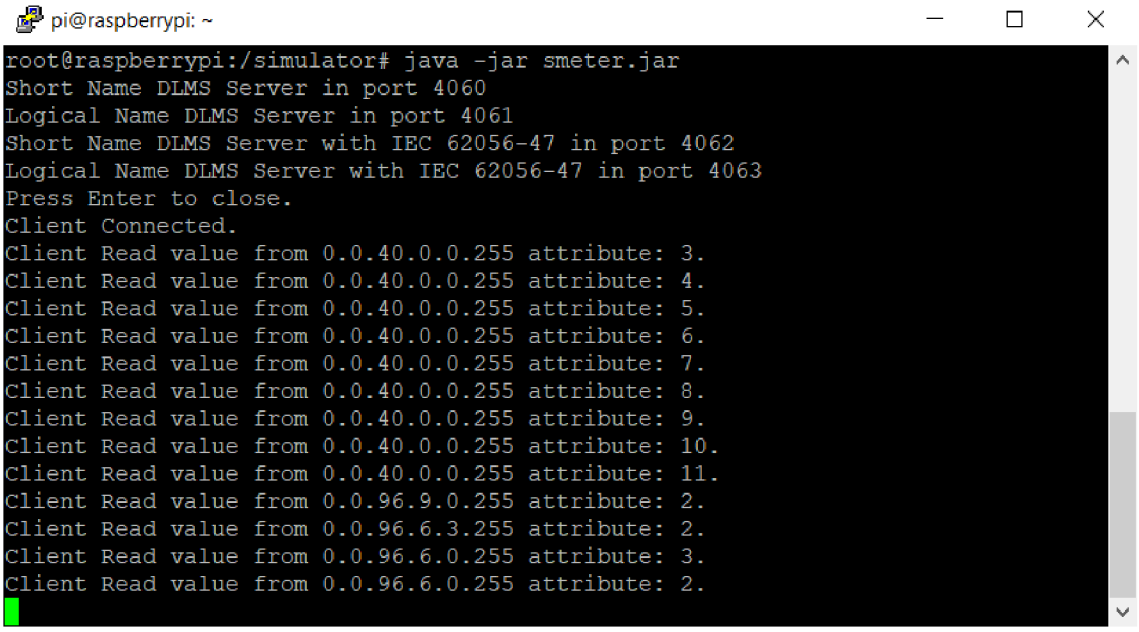
Výstup by měl vypadat následovně:

```
openjdk version "11.0.9.1"2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Raspbian-1deb10u2)
OpenJDK Server VM (build 11.0.9.1+1-post-Raspbian-1deb10u2, mixed
mode)
```

A nakonec pro spuštění projektu *smeter.jar* je třeba použít následující příkaz:

```
pi@raspberry:/# cd simulator
pi@raspberry:/simulator# java -jar smeter.jar
```

Tímto je spuštěn simulátor chytrého měřiče v hardwarově omezeném prostředí – tedy na Raspberry Pi. Pro ověření, zda vše funguje, bude provedeno čtení hodnot ze simulátoru chytrého měřiče (*simulator –> Read*), což funguje, jak má, viz Obr. 2.14.

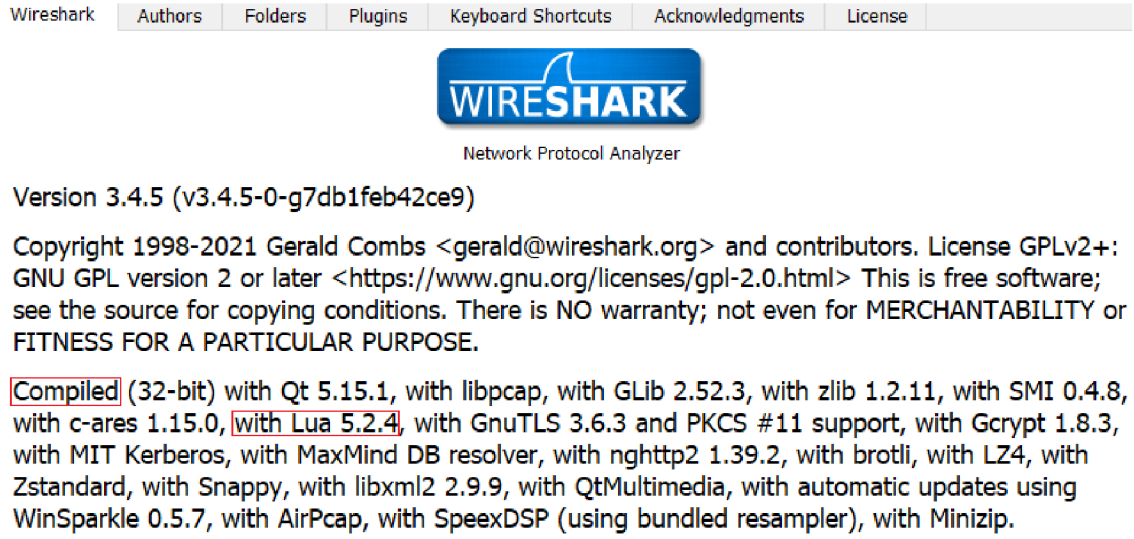


Obr. 2.14: Spuštění DLMS serveru na Raspberry Pi

2.5 Wireshark analýza

Pomocí programu Wireshark lze provést analýzu komunikace, která je realizována protokolem DLMS. Wireshark ovšem sám o sobě nedokáže protokol DLMS zachytit. Pro účely této analýzy byl implementován tzv. *Dissector protokolu DLSM pro Wireshark* [27], což je modul pro analýzu PCAP souborů s DLMS komunikací.

Pro správné fungování je nutné ve Wiresharku zkontrolovat, zda je konkrétní verze Wiresharku kompilována pomocí Lua interpreteru. To lze zjistit, po spuštění Wiresharku, kliknutím na *Help –> About Wireshark* v záložce *Wireshark* (viz Obr. 2.15).



Obr. 2.15: About Wireshark – Compiled with Lua

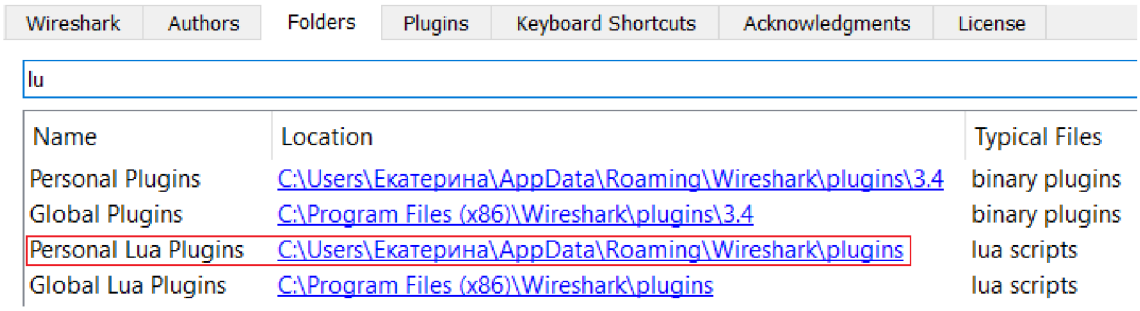
Následně je nutné přejít do záložky *Folders* a kliknutím a potvrzením dotazovacího okénka vytvořit složku *plugins*, pokud ještě v minulosti nebyla vytvořena. Do složky *plugins* je třeba přesunout soubory *dlms.lua*, *hdlc.lua* a *wrapper-dlms.lua* ze staženého *DLMS Dissectoru* (Obr. 2.16).

Tímto je *DLMS Dissector* implementován do programu Wireshark a je možné zachytit DLMS komunikaci. Na Obr. 2.17 lze vidět tato zachycená komunikace. Komunikace začíná odesláním *DLMS AARQ (AA Request)* klientskou částí, tedy koncentrátorem. Klient zde žádá cílovou stanici, tedy simulátor měřiče, o navázání spojení. Po obdržení pozitivní odpovědi *DLMS AARE (AA Response)*, odeslané serverem, může začít komunikace. Tato odpověď se nachází v druhém paketu z Obr. 2.17 a měla by obsahovat následující zprávu: *DLMS AARE Association Response: accepted*. Kvůli chybě v *DLMS Dissectoru* se ovšem zpráva nezobrazuje korektně.

1. Logical Name	LN objektu
2. Object list	Seznam objektů, které tato asociace může nabídnout a přístupová práva pro všechny metody a objekty
3. Associated partners ID	SAPs klienta a serveru (adresa)
4. Application context name	Název aplikačního kontextu
5. xDLMS context info	Podporovaná služba a maximální velikost PDU
6. Authentication mechanism name	Název použitého autentizačního mechanismu (level)
7. Secret	Heslo
8. Association status.	Informace o tom, zda je tento objekt použit
9. Security setup reference	LN objektu použitého pro nastavení zabezpečení
10. User list	Seznam uživatelů
11. Current user	Identifikátor aktuálního uživatele

Tab. 2.2: Atributy Association Logical name

Analýza komunikace probíhala po spuštění simulátoru měřiče spojením přes *Gurux DLMS Director* (kliknutím pravého tlačítka na *simulator* –> *Connect*) a přečtením všech hodnot na měřiči (*Read*). Po této akci se začně koncentrátor postupně dotazovat měřič na vybrané atributy OBIS kódů, kde na každý dotaz měřič obratem odesílá adekvátní odpověď. Nejprve jsou odeslány dotazy na atributy 3 až 11 OBIS kódu *0.0.40.0.0.255*. Jedná se o objekt *Association Logical Name*, který slouží k určení, jaký druh funkčnosti (objekty a služby) může měřič nabídnout. Používá se také ke sdělování informací o autentizaci (úroveň a heslo). Pro každou úroveň ověřování existuje vlastní *Association Logical name* objekt. Seznam jednotlivých atributů je znázorněn v Tab. 2.2.



Obr. 2.16: About Wireshark – Personal Lua Plugins

No.	Time	Source	Destination	Protocol	Info
1	0.00…	192.168.0.98	192.168.0.99	DLMS	DLMS AARQ Association Request
2	0.01…	192.168.0.99	192.168.0.98	DLMS	4061 → 59841 [PSH, ACK] Seq=1 Ack=46 Win=502 Len=60
3	2.58…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=3
4	2.72…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, structure (6 bytes)
5	2.72…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=4
6	2.73…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, structure (16 bytes)
7	2.73…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=5
8	2.73…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, structure (18 bytes)
9	2.73…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=6
10	2.73…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, structure (16 bytes)
11	2.74…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=7
12	2.74…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, octet-string (5 bytes)
13	2.74…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=8
14	2.75…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, enum (1 bytes)
15	2.75…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=9
16	2.75…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, octet-string (6 bytes)
17	2.75…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=10
18	2.76…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, array (1 bytes)
19	2.76…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=15, OBIS=0.0.40.0.0.255, attr=11
20	2.76…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, structure (5 bytes)
21	2.76…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=3, OBIS=0.0.96.9.0.255, attr=2
22	2.76…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, long-unsigned (2 bytes)
23	2.77…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=3, OBIS=0.0.96.6.3.255, attr=2
24	2.77…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, long-unsigned (2 bytes)
25	2.77…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=3, OBIS=0.0.96.6.0.255, attr=3
26	2.77…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, structure (5 bytes)
27	2.78…	192.168.0.98	192.168.0.99	DLMS	GetRequestNormal, class=3, OBIS=0.0.96.6.0.255, attr=2
28	2.78…	192.168.0.99	192.168.0.98	DLMS	GetResponseNormal, long-unsigned (2 bytes)

Obr. 2.17: Komunikace koncentrátoru se simulátorem měřiče

2.6 Budoucí rozšíření

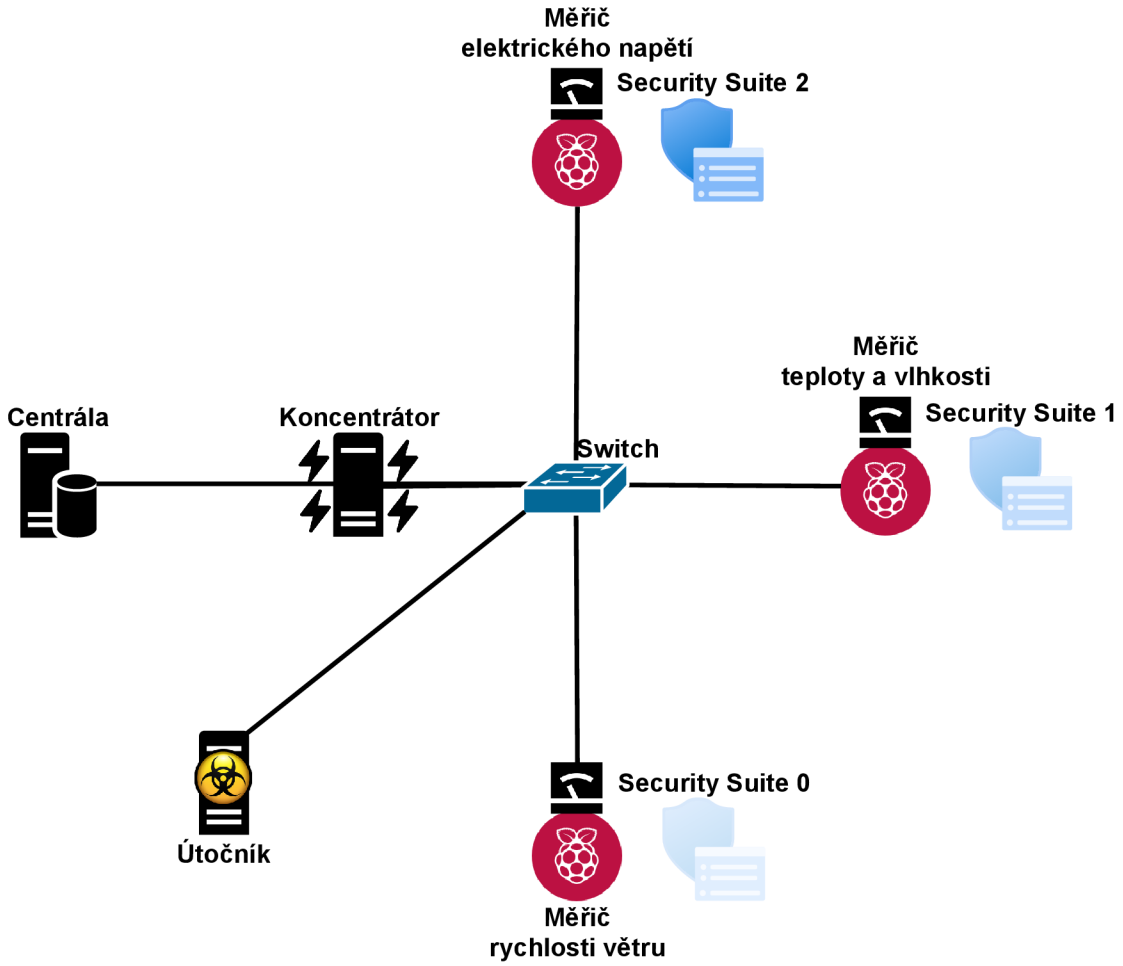
Tato kapitola se bude zabývat možným rozšířením do diplomové práce.

Použití chytrých měřičů a protokolu DLMS má obrovské možnosti dalšího rozšíření. Dalo by se například uvažovat o implementaci měřiče elektrického proudu, který bude připojen k Raspberry Pi a tím dosáhnout vytvoření simulátoru chytrého měřiče, který bude zaznamenávat opravdové hodnoty v reálném čase. Dále mohou být podobným způsobem implementovány simulátory chytrých měřičů na měření např. teploty, vlhkosti vzduchu, spotřeby vody, rychlosti větru, atd. Naměřené hodnoty těmito simulátory chytrých měřičů poté budou posílány na koncentrátor, který bude připojen, spolu s měřiči, k centrále a tím vznikne menší smart grid v rámci VUT laboratoře.

Bude také nutné zajistit zabezpečení, aby se pak dala simulovat různá narušení (např. dostupnosti) měřičů, odposlech komunikace pomocí MITM útoku, odchycení a modifikace zpráv odesílaných klientem na server (také MITM útok), podvržení DLMS/COSEM serveru, slovníkový útok na HLS atp.

V tomto vytvořeném smart gridu tedy bude několik simulátorů chytrých měřičů a každý z nich může komunikovat pomocí jiného security suite (např. měřič

elektrického napětí pomocí security suite 2, měřič teploty a vlhkosti pomocí security suite 1 a měřič rychlosti větru pomocí security suite 0). Tímto bude zajištěna různorodost zabezpečení a autentizace a tím pádem bude možné testovat bezpečnost jednotlivých měřičů pomocí simulovaných útoků a vyhodnocovat tyto bezpečnostní incidenty.



Obr. 2.18: Příklad schéma zapojení pro budoucí rozšíření

Tím, že programovací jazyk Java je multiplatformní, dalo by se případně simulátor chytrého elektroměru virtualizovat a implementovat jej v rámci Cyber Range. Zároveň by tento virtualizovaný elektroměr mohl sloužit jako nástroj na testování zabezpečení nějakého již vytvořeného smart gridu pro zjištění, zda se bude v této síti koncentrátor dotazovat připojeného virtuálního elektroměru na odečet hodnot.

Závěr

Bakalářská práce je zaměřena na vytvoření simulátoru chytrého měřiče (smart metru) komunikujícího pomocí protokolu DLMS. Koncept simulátoru chytrého měřiče byl implementován pomocí programovacího jazyku Java v hardwarově omezeném prostředí Raspberry Pi. Tento simulátor chytrého měřiče po spuštění předával předem určená data koncentrátoru, jehož funkce byla uskutečněna pomocí Gurux DLMS Directoru.

V teoretické části práce byly rozebrány elektrické sítě a důležitost chytrých měřičů v dnešní, čím dál tím rychleji technologicky rozvíjející se, době. Největší pozornost je věnována protokolu DLMS/COSEM, jeho základním funkcím, tedy jak funguje spojení, zabezpečení, autentizace a další důležité věci. Dále jsou zde popsány základní informace o programovacím jazyku Java a o platformě Raspberry Pi.

V praktické části bakalářské práce byla popsána vlastní implementace simulátoru chytrého měřiče, která byla provedena v několika krocích. Nejprve bylo nutné vytvořit testovací prostředí pro zkoušku komunikace simulátoru chytrého měřiče (*gurux.dlms.server.example2.java*) s koncentrátorem (*Gurux DLMS Director*). Po otestování funkčnosti mohla započít realizace zadání bakalářské práce, tedy vytvoření simulátoru chytrého měřiče komunikujícího pomocí protokolu DLMS a napsaného v programovacím jazyku Java. Tato první část byla splněna v testovacím prostředí. Projekt byl upraven pro lepší porozumění jeho kódu a snadnější rozpoznání správné funkčnosti. Simulátor chytrého měřiče do té doby generoval náhodné hodnoty, což bylo úpravou kódu změněno na generování předem určených hodnot. Další výzvou bylo přenést Java projekt do hardwarově omezeného prostředí, tedy na Raspberry Pi. Po úspěšném přenosu projektu a po zjištění korektní funkčnosti byla nakonec provedena analýza komunikace simulátoru chytrého měřiče s koncentrátorem pomocí programu Wireshark pro lepší pochopení funkčnosti.

Zadání bakalářské práce bylo tedy splněno. Navíc byl proveden návrh na možné budoucí rozšíření do diplomové práce, který pojednává o obrovském potenciálu probíraného tématu díky velkému množství možností na další analýzu (např. testování bezpečnosti protokolu DLMS v nově vytvořeném smart gridu).

Literatura

[1] DLMS/COSEM for smart metering. *Smart Energy.* [online]. [cit. 2020-11-26]. Dostupné z URL: <https://www.smart-energy.com/top-stories/dlms-cosem-for-smart-metering/>.

[2] IEC 62056-5-3:2017. *IEC Webstore.* [online]. [cit. 2020-11-26]. Dostupné z URL: <https://webstore.iec.ch/publication/27065>.

[3] Na světě je stále 840 milionů lidí bez přístupu k elektřině. *O Energetice.* [online]. [cit. 2020-11-26]. Dostupné z URL: <https://oenergetice.cz/elektrina/svete-stale-840-milionu-lidi-bez-pristupu-k-elektrine>.

[4] Spotřeba elektřiny byla v roce 2018 nejvyšší za celé sledované období. *TZB Info.* [online]. 15. 5. 2019 [cit. 2020-11-22]. Dostupné z URL: <https://energetika.tzb-info.cz/elektroenergetika/19020-spotreba-elektriny-byla-v-roce-2018-nejvyssi-za-cele-sledovane-obdobi>.

[5] KOHOUT, David. *Zátěžový generátor zpráv DLMS/COSEM.* Brno, 2019, 59 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Tomáš Lieskovan.

[6] Advantages and disadvantages of smart meters. *Look After My Bills.* [online], [cit. 2020-11-20]. Dostupné z URL: <https://lookaftermybills.com/blog/advantages-disadvantages-of-smart-meters/>.

[7] Smart metering. *Pro Elektrotechniky.* [online]. [cit. 2020-11-28]. Dostupné z URL: <http://www.proelektrotechniky.cz/smart-metering.php>.

[8] Smart vs AMR meters. *Yuenergy.* [online]. [cit. 2020-11-27]. Dostupné z URL: <https://www.yuenergy.co.uk/news/smart-vs-amr-meters-what-is-the-difference>.

[9] DLMS Modelling - Messaging - Transport. *DLMS.* [online]. DLMS User Association [cit. 2020-11-20]. Dostupné z URL: <https://www.dlms.com/dlms-cosem/overview?fbclid=IwAR2TikdV9rvOXKNHrtaLO3h6Ccn8bq6rNOugfftgLDIWPZIJlxosodRae6c>.

[10] JIRKA, Matěj. *Framework DLMS/COSEM pro sběr dat v AMM systémech.* Praha, 2016, 70 s. Dilomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, Katedra řídící techniky. Vedoucí práce Jiří Novák.

[11] Smart Energy International. *Smart Energy.* [online]. [cit. 2020-11-28]. Dostupné z URL: <https://www.smart-energy.com/wp-content/uploads/Edward_Beroset.pdf>.

[12] The OSI model explained. *Networkworld.* [online]. [cit. 2020-11-26]. Dostupné z URL: <https://www.networkworld.com/article/3239677/the-osi-model-explained-and-how-to-easily-remember-its-7-layers.html>.

[13] BUŠ, Ondřej. *Simulátor DLMS koncentrátoru.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Tomáš Lieskovan

[14] Green Book: Architecture and Protocols.*DLMS.* [online]. Zug, Switzerland: DLMS User Association, 2017 [cit. 2020-10-30]. Dostupné z URL: <https://www.dlms.com/files/Green-Book-Ed-83-Excerpt.pd>.

[15] DÁVIDÍK, Roland. *Automatizovaný tester bezpečnosti chytrých zařízení v energetice.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Tomáš Lieskovan.

[16] MATOUŠEK, Petr. *Analysis of DLMS Protocol.* Brno, 2017, 38 s. Technical Report. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z URL: <https://www.fit.vut.cz/research/publication-file/11616/TR-DLMS.pdf>.

[17] Úvod do jazyka Java. *IT Network.* [online]. [cit. 2020-11-11]. Dostupné z URL: <https://www.itnetwork.cz/java/zaklady/java-tutorial-uvod-do-jazyka-java?fbclid=IwAR0Pztl0lLlxQ-F3kfEm0x4vtXZwnJbfw8H-0qat3h_KGjt50JMjXQn1w3Q>.

[18] Raspberry Pi. *Raspberry Pi.* [online]. [cit. 2020-11-20]. Dostupné z URL: <https://www.raspberrypi.org/>.

[19] Raspberry Pi Models. *ThePiHut.* [online]. [cit. 2020-11-29] Dostupné z URL: <https://thepihut.com/blogs/raspberry-pi-roundup/raspberry-pi-comparison-table>.

[20] 20 Best Operating Systems You Can Run on Raspberry Pi in 2020. *Fossmint.* [online]. [cit. 2020-11-29]. Dostupné z URL: <https://www.fossmint.com/operating-systems-for-raspberry-pi/>.

[21] Gurux DLMS Java. *GitHub.*[online]. GitHub [cit. 2019-11-20]. Dostupné z URL: <https://github.com/Gurux/gurux.dlms.java>.

[22] Gurux DLMS Director. *Gurux Fi.* [online]. [cit. 2020-12-06]. Dostupné z URL: <https://www.gurux.fi/Download>.

[23] Angry IP Scanner. *Angry IP.* [online]. [cit. 2021-05-05]. Dostupné z URL: <https://angryip.org/>.

[24] Data Type Output. *Gurux Fi.* [online]. [cit. 2021-05-09]. Dostupné z URL: <https://www.gurux.fi/node/4325>.

[25] WinSCP 5.17 Download. *WinSCP.* [online]. [cit. 2021-05-11]. Dostupné z URL: <https://winscp.net/eng/download.php>.

[26] Installing Java on Raspberry Pi. *Linuxize.* [online]. [cit. 2021-05-13]. Dostupné z URL: <https://linuxize.com/post/install-java-on-raspberry-pi/>.

[27] DLMS Dissector. *GitHub.* [online]. [cit. 2021-05-15]. Dostupné z URL: <https://github.com/matousp/dlms-analysis>.

[28] Description of OBIS code for IEC 62056 standard protocol. *Promotic.* [online]. [cit. 2021-05-20]. Dostupné z URL: <https://www.promotic.eu/en/pmdoc/Subsystems/Comm/PmDrivers/IEC62056_OBIS.htm>.

[29] DLMS/COSEM Protocol Security Analysis. *Datawok.* [online]. [cit. 2021-05-20]. Dostupné z URL: <https://datawok.net/notes/2020/dlms-sec>.

[30] Arduino and Raspberry Pi. *Pixabay.* [online]. [cit. 2021-05-25]. Dostupné z URL: <https://pixabay.com/photos/arduino-raspberry-pi-electronic-4753005/>.

[31] DLMS/COSEM Single Phase Smart Meter. *Taiwan Excellence.* [online]. [cit. 2021-05-25]. Dostupné z URL: <https://www.taiwanexcellence.org/en/award/product/32158>.

Seznam symbolů, veličin a zkratek

AA	Application Association
AAs	Application Associations
ACSE	Association Control Service Element
AE	Application Entity
AEs	Application Entities
AL	Application Layer
AMI	Advanced Metering Infrastructure
AMR	Automatic Meter Reading
AP	Application Process
APDU	Application Protocol Data Unit
APs	Application Processes
ARM	Advanced RISC Machines
ASE	Application Service Elements
BSD	Berkeley Software Distribution
COSEM	Companion Specification for Energy Metering
DLMS	Device Language Message Specification
EE	Enterprise Edition
GNU	GNU's Not Unix!
GPIO	General Purpose Input/Output
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IoT	Internet of Things
ISO	International Organization for Standardization
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LAN	Local Area Network
LD	Logical Device
LDN	Logical Device Name
LDs	Logical Devices
LN	Logical Name
LXDE	Lightweight X11 Desktop Environment
MAC	Media Access Control
ME	Micro Edition
MITM	Man In The Middle
OBIS	Object Identification System
OS	Operation System

OSI	Open Systems Interconnection
OSMC	Open Source Media Center
PhL	Physical Layer
PLC	Programmable Logic Controller
POSIX	Portable Operating System Interface
PPP	Point-to-Point Protocol
RBAC	Role-Based Access Control
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RS	Recommended Standard
SAP	Server Access Point
SAPs	Server Access Points
SD	Storage Device
SE	Standard Edition
SN	Short Name
TCP/IP	Transmission Control Protocol/Internet Protocol
TL	Transport Layer
UA	User Association
WAN	Wide Area Network

Seznam příloh

A Obsah přiloženého CD

Obsah přiloženého CD byl také nahrán na vzdálené úložiště[1].

- `/` ..... kořenový adresář přiloženého CD
  - `DLMS Dissector` ..... Wireshark plugin nutný pro analýzu komunikace
    - `dlms.lua`
    - `hdlc.lua`
    - `wrapper-dlms.lua`
  - `Ecplise projekt` ..... výchozí stav simulátoru chytrého měřiče
    - `gurux.dlms.server.example2.java.zip`
  - `Gurux DLMS Director` ..... objekt pro čtení hodnot ze simulátoru měřiče
    - `simulator.gxc`
  - `Spustitelný JAR soubor` ..... JAR soubor určený ke spuštění na Raspberry Pi
    - `smeter.jar`
  - `Wireshark analýza` ..... DLMS komunikace
    - `dlmscom.pcapng`
  - `Simulátor_elektroměru_s_DLMS_protokolem.pdf` ..... elektronická verze práce

[1] `https://owncloud.cesnet.cz/index.php/s/N4cmPIViTvxba3n`