VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Martin Štůsek

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

UNIVERZÁLNÍ PLATFORMA PRO VZDÁLENOU SPRÁVU IOT ZAŘÍZENÍ A VIZUALIZACI M2M DAT

UNIVERSAL PLATFORM FOR REMOTE MANAGEMENT OF IOT DEVICES AND VISUALIZATION OF M2M DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

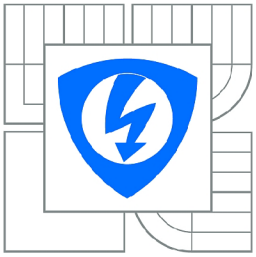
AUTOR PRÁCE
AUTHOR

Bc. Martin Štůsek

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Pavel Mašek

BRNO 2016



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Martin Štůsek	***ID:***	146975
Ročník:	2	***Akademický rok:***	2015/2016

NÁZEV TÉMATU:

Univerzální platforma pro vzdálenou správu IoT zařízení a vizualizaci M2M dat

POKYNY PRO VYPRACOVÁNÍ:

V důsledku masivního rozšíření nízko-výkonových zařízení (embedded devices) v roli chytrých měřičů/senzorů vzniká, kromě otázky přenosu těchto dat s využitím mobilních sítí, také potřeba vizualizace získaných dat pro koncové uživatele. V současné době se lze setkat s řešeními, kdy jednotlivé společnosti nabízejí své vizualizační platformy, které jsou ovšem omezeny pouze pro konkrétní typ přístroje. V diplomové práci bude proto pozornost soustředěna na využití protokolu TR-069 pro vzdálenou konfiguraci zařízení v IoT infrastruktuře. Dále bude provedena implementace komplexní platformy pro vizualizaci získaných dat od senzorů (M2M zařízení). Důraz bude kladen především na vytvoření univerzální databáze pro ukládání dat z různých M2M zařízení a dále pak jejich agregaci a zobrazení.

DOPORUČENÁ LITERATURA:

[1] BOSWARTHICK, David, Omar ELLOUMI a Olivier HERSENT. 2012. M2M communications: a systems approach. Hoboken, N.J.: Wiley, xxiii, 308 p. ISBN 978-1-119-99475-6.
[2] ALVES, Alexandre de Castro. OSGi in Depth. Manning Publications; 1st ed, 2012, 394 s. ISBN 978-1935182177.

Termín zadání:	1.2.2016	***Termín odevzdání:***	25.5.2016

Vedoucí práce: Ing. Pavel Mašek
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem této diplomové práce je vytvoření univerzální aplikace umožňující vizualizaci M2M dat s podporou vzdálené konfigurace protokolem TR-069. Teoretická část práce obsahuje podrobný popis M2M komunikace, protokolu TR-069, standardu OSGi a SQLite databáze. Dále jsou popsány použité technologie pro konfiguraci protokolem TR-069, zejména klient modus TR-069 a server genieacs. Praktická část obsahuje souhrn všech vytvořených balíčků a komunikačních rozhraní. Je popsána také implementovaná databáze a uživatelské rozraní, umožňující přehlednou vizualizaci získaných dat.

KLÍČOVÁ SLOVA

ACS, automatická konfigurace, balíček, CPE, databáze, genieacs, GUI, H2H, inteligentní senzory, Knopflerfish, M2M, modus TR-069, OSGi, SQLite, TR-069, uživatelské rozhraní

ABSTRACT

The aim of this diploma thesis is to create universal application able to visualize M2M data and allows remote management of smart sensors using TR-069 protocol. First part of this thesis contains comprehensive evaluation of TR-069 standard and OSGi platform. Next, extensive analysis of embedded databases with detailed description of SQLite platform is provided. Auto-configuration server genieacs and modus TR-069 client, two parts needed for the proper run of remote configuration, are described in more detail in following section. Practical part of this thesis contains description of all created OSGi bundles together with communication interfaces. Moreover, the description of designed database and developed user interface is given.

KEYWORDS

ACS, auto-configuration, bundles, CPE, database, genieacs, GUI, H2H, Knopflerfish, M2M, modus TR-069, OSGi, smart sensors, SQLite, TR-069, user interface

ŠTŮSEK, Martin *Vývoj univerzální platformy pro vzdálenou správu IoT zařízení a vizualizaci M2M dat*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 80 s. Vedoucí práce byl Ing. Pavel Mašek,

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Vývoj univerzální platformy pro vzdálenou správu IoT zařízení a vizualizaci M2M dat“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové 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 autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) 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 autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Pavlu Maškovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Technicka 12, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .
podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Z důvodu velkého rozšíření chytrých M2M (Machine-to-Machine) meřících zařízení, vzniká v posledních letech potřeba vizualizace získaných dat pro koncového uživatele. Společnosti zabývající se touto problematikou již mají uživatelská rozhraní pro koncové uživatele. Tyto aplikace ale vetšinou spolupracují pouze se zařízeními od konkrétního výrobce. Cílem této diplomové práce je proto vytvořit otevřenou platformu umožňující vzdálenou správu zařízení a vizualizaci získaných dat v uživatelském rozhraní.

Z důvodu zajištění co nejvyšší úrovně zabezpečení je nutné pravidleně aktualizovat software, který je na zařízení spuštěn. K tomuto účelu je vyvíjen protokol TR-069, který umožňuje vzdálenou správu zařízení bez zásahů ze strany uživatele. To přináší zvýšení uživatelské přívetivosti těchto systémů. Implementace protokolu TR-069 je důležitá také z pohledu telekomunikačních operátorů, kteří tento protokol velmi často využívají pro vzdálenou správu svých zařízení. V práci je protokol využit především pro přenos počáteční konfigurace měřících zařízení.

Díky nárůstu výpočetního výkonu domácích směrovačů, které ve většině případů využívají procesory založené na architektuře ARM (Acorn RISC Machine) či MIPS (Microprocessor without Interlocked Pipeline Stages), je možno na těchto směrovačích spustit prostředí Java. Podpora této technologie je důležitá pro spuštění OSGi frameworku, jež je preferován HGi (Home Gateway Initiative) aliancí. Tyto zařízení také obsahují vestavěnou flash paměť na kterou mohou být uložena přijatá data od senzorů/měřících zařízení.

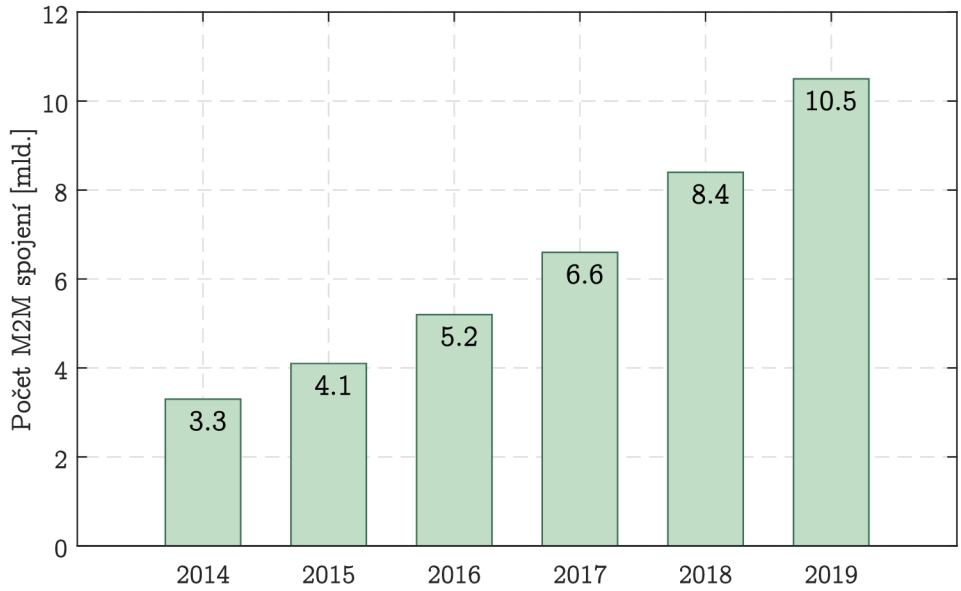
Pro sytémy chytrých zařízení se využívá technologie OSGi (Open Services Gateway initiative), která byla vyvinuta s cílem poskytnout modulární platformu nezávislou na konkrétním zařízení tzv. embedded zařízení. Umožňuje také dynamicky aktualizovat běžící programy bez nutnosti restartování zařízení. Z dostupných otevřených implemntací standardu OSGi byl vybrán framework Knopflerfish.

Pro uložení dat po delší časové období je využito databáze SQLite, která je určena pro systémy s nižším výpočetním výkonem. Z uživatelského hlediska je nejdůležitějším bodem uživatelské rozhraní, které obstarává veškerou interakci mezi uživatelem a zařízením. Toto rozhraní využívá moderní technologie jako JavaScript, HTML5 a WebSocket. Dle moderních trendů je rozhraní plně responzivní.

1 KOMUNIKACE TYPU M2M

Objem komunikace M2M (Machine-to-Machine) vykazuje v posledních letech obrovský nárůst, dle předpovědí firmy Cisco bude tento trend v nejbližší budoucnosti pokračovat [33].

Jak lze vidět na Obr. 1.1 v roce 2019 bude počet M2M spojení 10,5 miliard, to je téměř čtyřikrát více než v roce 2015. Z tohoto důvodu je nutné počítat s komunikací typu M2M jako s dominantní technologií blízké budoucnosti.



Obr. 1.1: Růst počtu M2M spojení dle Cisco.

1.1 Rozdíly mezi M2M a H2H komunikací

Jak již vyplývá z názvu H2H (Human-to-Human), jedná se o komunikaci člověka s člověkem. I když se na první podled může zdát, že při telefonování se jedná o komunikaci typu M2M, není tomu tak. Při této činnosti je stále nutná interakce zařízení s člověkem, podobná situace nastává i při prohlížení internetových stránek, kdy je uživatel zdrojem dotazů na server.

Komunikace typu M2M úplně vynechává člověka z komunikačního řetězce, pro uživatele se potom takovéto zařízení jeví jako neviditelné.

Rozdíl mezi M2M a H2H je také ve struktuře přenášených dat. Zařízení typu M2M jsou většinou malá, bateriemi napájená, zařízení s nízkým výpočetním výkonem (často označovaná jako „embedded devices“). Proto se pro komunikaci využívají jednoduché protokoly, které obsahují minimum nadbytečných informací. Výsledné

pakety mají malou velikost např. pro protokol WM-BUS (Wireless M-BUS) se jejich velikost pohybuje do 50 B. Pakety M2M zařízení jsou většinou odesílány v pevně stanovené časové intervaly.

Většina datového provozu generovaného M2M zařízeními je ve směru od zařízení k serveru tzv. upload. Tento provoz má nízkou přenosovou rychlost a využívá malou šířku přenosového pásma. Síťový provoz generovaný H2H zařízeními má opačný charakter tj. ze serveru k uživateli tzv. download. Tento provoz také využívá mnohem větší šířku pásma.

Některá M2M zařízení jsou využívána v systémech reálného času, a proto je u nich kladen důraz na nízké zpoždění a chybovost při přenosu. U zařízení H2H nejsou tyto parametry tak klíčové, např. při telefonování může být několik paketů ztraceno a kvalita hovoru je stále dostatečná [17].

Přehled výše uvedených parametrů je uveden v Tab. 1.1.

Tab. 1.1: Rozdíly mezi M2M a H2H komunikací [17].

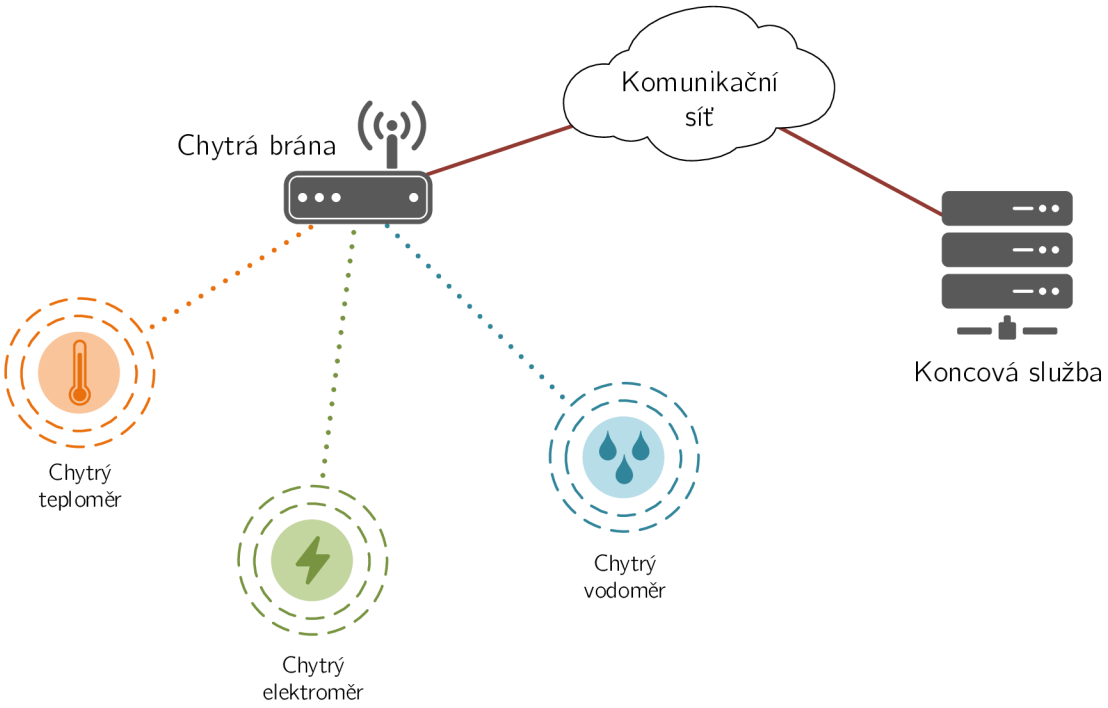
Parametr	H2H	M2M
Počet zařízení	Počet zařízení je vysoký, růst H2H zařízení je však pomalejší a nemá potenciál dosáhnout počtu M2M zařízení.	Počet zařízení je velmi vysoký a stále prudce roste. Množství zařízení přináší také problém s adresací těchto zařízení – jeden z důvodů zavedení IPv6 (Internet Protocol version 6).
Objem dat	Vetšina datového toku je download využívající velkou šířku pásma.	Vetšinu provozu tvoří upload. Jsou přenášeny malé datové jednotky, při přenosu je využita malá šířka pásma.
Napájení	Baterie může být vyměněna, nebo znovu nabita.	Baterie vetšinou nelze vyjmout a proto musí mít dlouhou životnost.
Zpoždění	Existuje částečná tolerance ke zpoždění, která nenarušuje kvalitu služeb (hovoru).	Některé M2M aplikace pracují v reálném čase, proto je tolerance ke zpoždění velmi nízká.
Nároky na síť	Dnešní sítě jsou vetšinou navrženy pro H2H komunikaci, s velkým datovým tokem.	M2M komunikace má jiné nároky na přenosovou síť. Komunikuje více uživatelů s menším datovým tokem, který je generován v pravidelných intervalech. Je nutné kontrolovat zda pravidelné vysílání nezahltí síť.

1.2 Obecný popis M2M

Základní úlohou M2M komunikace je ve většině případů navázání obousměrného spojení pro výměnu dat s koncovou službou přes komunikační síť (Internet).

S koncovou službou na vzdáleném serveru může komunikovat několik zařízení patřících do jedné skupiny. Z důvodu konečného počtu spojení, které je koncová služba schopna uskutečnit, nejsou M2M zařízení přímo spojena se službou na vzdáleném serveru, ale do komunikační cesty je vložen koncentrační prvek označovaný jako MTCG (Machine-type Communication Gateway). Tento prvek se nazývá brána a slouží jako prostředník pro komunikaci mezi M2M zařízeními a koncovou službou jak lze vidět na Obr. 1.2.

Tento způsob komunikace mezi zařízeními a vzdálenou službou se velmi často využívá v systémech inteligentních měřících přístrojů v domácnostech (tzv. smart homes) [3].



Obr. 1.2: Komunikace M2M s koncentračním prvkem.

1.3 Požadavky na M2M zařízení

S nárůstem M2M komunikace začalo také vznikat množství nových zařízení s velmi specifickými vlastnostmi. Tyto zařízení přinesly specifické požadavky na koncové aplikace a komunikační síť.

1.3.1 Množství zařízení

Jednou z největších změn, kterou M2M zařízení přinesla, je jejich počet, který rychle stoupá. Je jasné, že počet M2M zařízení překoná v blízké době počet zařízení, která s uživatelem přímo komunikují (počítače, mobilní telefony, atd.).

Tento růst klade vyšší požadavky zejména na propustnost komunikačních sítí. Dnes budované sítě jsou spíše navrhovány pro větší datový tok než pro velkou koncentraci komunikujících zařízení, která je specifická pro M2M komunikaci [3].

1.3.2 Rozmanitost zařízení

Existuje mnoho návrhů na použití M2M komunikace v reálném světě. Tyto návrhy počítají se zařízeními s různým výpočetním výkonem, přenosovou rychlostí atd. Tato rozmanitost zařízení a požadavků na ně může být největší překážkou ve vytvoření obecného modelu pro M2M komunikaci [3].

1.3.3 Viditelnost zařízení

Jak již bylo zmíněno dříve, M2M komunikace nevyžaduje žádnou spolupráci s uživatelem. Tento požadavek (vlastnost) je velmi důležitý, protože vylučuje možnost zanesení chyby do komunikace uživatelem.

Z tohoto důvodu je správa zařízení klíčovou částí správy sítě a musí být korektně implementována v zařízení [3].

1.3.4 Aplikace v kritických případech

Některá zařízení využívající M2M komunikaci mohou sloužit k záchraně životů, nebo řídit životně důležitou infrastrukturu. Tyto zařízení kladou důraz zejména na odezvu komunikační sítě [3].

1.3.5 Konstrukce zařízení

K výše uvedeným požadavků na M2M zařízení lze přidat požadavky na konstrukci zařízení, které mají zásadní vliv na způsob komunikace v síti.

Omezení ve funkčnosti

Výpočetní výkon dnešních M2M zařízení je několikanásobně nižší než výkon chytrých telefonů nebo přenosných počítačů. Některá zařízení nenabízejí ani vzdálenou aktualizaci programového vybavení. Důvodem těchto omezení je cena, protože konkurence tlačí cenu těchto zařízení co nejníže.

Omezení funkčnosti vychází také z logického rozhodnutí, které počítá s tím že chytrý senzor není komplexní zařízení ale jen jednoduchá výpočetní jednotka. Ve vetšině případů není třeba, aby senzor prováděl složité komplexní operace, stačí pouze aby odeslal data koncové aplikaci, kde se data zpracují [3].

Nízká spotřeba

Některá M2M zařízení nejsou používána uvnitř budov a proto nemohou být trvale připojena ke zdroji napájení. Z tohoto důvodu je vetšina M2M zařízení napájena zabudovanými bateriemi. Životnost baterie WM-BUS zařízení se pohybuje v rozmezí 10 až 15 let, což je doba, která převyšuje morální životnost instalovaného zařízení.

Tato skutečnost má za následek snížení komunikace zařízení po síti jen na nejnutnější případy, kdy zařízení odesílají data o měření jen v pěvně stanovených časových intervalech [3].

Vestavěná zařízení

Mnoho M2M zařízení je umístěno v systémech se speciálními bezpečnostními podmínkami. To činí výměnu bez zásahu do celého systému velmi obtížnou v některých případech až nemožnou [3].

Doba života

Oblasti použití M2M zařízení jsou i mimo komunikační technologie, kde probíhá vývoj komponent velmi rychle. Požadavky na dobu života M2M zařízení tedy budou velmi odlišné např. v oblasti elekrické sítě, kde je výměna velmi obtížná, bude doba života M2M zařízení mnohem delší [3].

1.4 Komunikační protokoly M2M

Oblast, kterou pokrývá M2M komunikace, je velmi různorodá. Z tohoto důvodu je využíváno mnoho rozličných protokolů a technologií s odlišnými vlastnostmi. Některé M2M aplikace kladou větší nároky na přenosové rychlosti použitých technologií (video stream), jiné nepotřebují vysokou přenosovou rychlost ale zaměřují se na nízkou spotřebu energie (chytré senzory). Porovnání nejčastěji používaných bezdrátových technologií s krátkým dosahem je uvedeno v Tab. 1.3. V následujících kapitolách budou blíže popsány dvě technologie: ZigBee 1.4.1 a Wireless M-Bus 1.4.2.

1.4.1 ZigBee

Technologie ZigBee vychází z bezdrátového energeticky úsporného standardu IEEE 802.15.4, ktrerý definuje fyzickou a linkovou vrstvu [21]. ZigBee tento standard rozšiřuje o síťovou/zabezpečovací vrstvu a také o vrstvu služeb pro aplikační vrstvu. Předchozí dvě vrstvy (síťová, vrstva služeb) definuje ZigBee aliance, nad těmito vrsvami je aplikační vrstva, kterou definuje zákazník.

Fyzická vrstva ZigBee definuje způsob bezdrátové komunikace a tři frekvenční pásma, ve kterých může probíhat komunikace:

- 2,4 GHz, 16 kanálů, 250 kb/s, definováno celosvětově,
- 915 MHz, 10 kanálů, 40 kb/s, definováno pro americký kontinent,
- 868 MHz, 1 kanál, 20 kb/s, definováno pro Evropu.

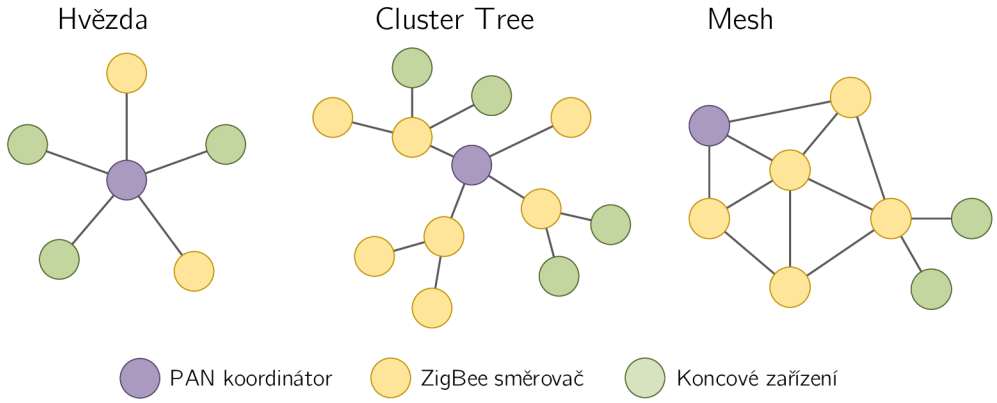
Linková vrstva definuje komunikaci prostřednictvím rámců. Vrstva definuje celkem čtyři typy rámců, které přenášejí užitečná data nebo jsou určeny pro správu, řízení či sestavení spojení:

- **Data Frame** – rámec přenášející užitečná data,
- **Acknowledgement Frame** – rámec přenášející potvrzovací informace pro linkovou vrstvu,
- **MAC Command Frame** – rámec k centralizovanému řízení a nastavení klientů v síti,
- **Beacon Frame** – rámec pro synchronizaci zařízení v síti, umožňuje přepnout zařízení do režimu nízké spotřeby.

Pro adresaci může být využito adres dlouhých 64 bitů nebo adres zkrácených na 16 bitů. Zkrácená adresa umožňuje adresovat 65 535 zařízení, každá síť je dále identifikována pomocí PAN ID (Personal Area Network ID), který je určen pro identifikaci překrývajících se sítí. Z hlediska topologie jsou definovány tři typy sítí jak lze vidět na obrázku Obr. 1.3. Nejjednoduší topologií je hvězda, zde je řízením pověřeno jedno zařízení tzv. koordinátor a ostatní pracují jako koncová zařízení. Úpravou topologie lze získat topologii Cluster Tree, kde je využito směrovačů pro komunikaci koncových zařízení s koordinátorem. Kombinací předchozích vznikne mesh síť [22].

Další vrstvy jsou definovány standardem ZigBee. Síťová vrstva se stará o připojení/odpojení zařízení od sítě, zabezpečení a směrování. Pro zabezpečení je využito šifry AES (Advanced Encryption Standard). V případě koordinátora sítě se vrstva stará o přiřazování adres novým zařízením.

Aplikační vrstva ZigBee umožňuje párování zařízení podle poskytovaných služeb. Pomocná vrsva objektů ZigBee definuje roli zařízení (koordinátor, směrovač, koncové zařízení) [22].



Obr. 1.3: Možné ZigBee topologie.

1.4.2 Wireless M-Bus

Standard Wireless M-Bus definuje bezdrátový přenos dat zejména mezi vodoměry, elektroměry a koncentrátory. WM-Bus specifikuje měřící zařízení klient (slave) a další zařízení nejčastěni server (master) který funguje jako centrální prvek [30].

Standard definuje také tři různé režimy označované S, T a R. Jednotlivé režimy se liší přenosovou rychlostí a typem komunikace 1 (jednosměrná), 2 (obousměrná). Přehled komunikačních režimů lze nalézt v Tab. 1.2.

Bezdrátová komunikace probíhá ve 12 kanálech v pásmu kolem 868 MHz (kanály 863,30 MHz a 868,95 MHz jsou využívány v režimu S a T, v režimu R je 10 uživatelsky volitelných kanálů s frekvencí 868,03 + n × 0,06 MHz), každý z komunikačních režimů má odlišné požadavky např. specifikovaný kanál, přesnost frekvence a tloeranci přenosové frekvence. Dosah v případě pužití antény o délce 1/4 vlny (8,2 cm) je při přímé vidielnosti 500 až 600 m [30].

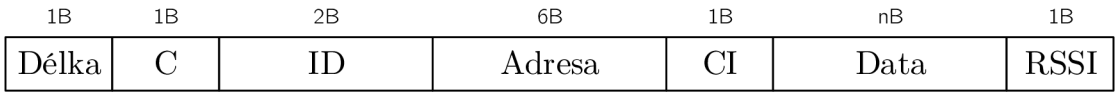
Topologie sítě má hvězdicovou strukturu, která ve svém středu obsahuje centrální jednotku, ke které snímače přenášejí svá data. Centrální prvek nikdy nezahajuje komunikaci, pracuje tedy jako server, čeká na navázání spojení měřící jednotkou, která pracuje jako klient. V případě obousměrné komunikace přechází klient do přijímacího režimu pouze na krátký čas jím vytvořené komunikace. Jen v tomto časovém úseku může koncentrátor vysílat řídící data měřící jednotce [30].

Adresování protokolu Wireless M-bus je pouze na základě sériového čísla, které je uloženo v paměti měřiče. Koncentrátor by měl obsahovat tabulku sériových čísel, se kterými může komunikovat. Pokud takovou tabulku neobsahuje, přijímá data od všech senzorů.

Wireless M-bus pakety jsou velmi jednoduché a nezaručují žádné zabezpečení přenosu, data mohou být zabezpečena pouze na aplikační vrstvě. Využívá se 128 bitová AES šifra. Formát Wireless M-bus paketu je uveden na Obr. 1.4 [30].

Tab. 1.2: Režimi komunikace protokolu Wireless M-bus [30].

Režim	Rychlost	Komunikace	Popis
S1	32,768 kb/s	Jednosměrná	V tomto režimu odesílá zařízení data několikrát denně. Koncentrátor může v tomto režimu šetřit energii tak, že je ve výchozím stavu v úsporném režimu. Při komunikací musí měřící zařízení před odesláním dat poslat „wake up“ signál a až poté data.
S1-m	32,768 kb/s	Jednosměrná	Stejný jako S1, ale koncentrátor nesmí přejít do režimu spánku.
S2	32,768 kb/s	Obousměrná	Obousměrná verze režimu S1.
T1	100 kb/s	Jednosměrná	Označuje vysílací režim, v tomto módu měřící zařízení periodicky zasílá data koncentrátoru. Interval zasílání je nastavitelný (sekundy, minuty).
T2	100 kb/s	Obousměrná	Obousměrná varianta režimu T1. Koncentrátor si navíc může vyžádat data mimo stanovený čas.
R2	4,8 kb/s	Obousměrná	Označuje přijímací režim, který dovoluje příjmat data od více měřících zařízení s využitím multiplexu.



Obr. 1.4: Wireless M-bus paket.

- **Délka** - délku paketu,
- **C** - pole obsahující řídící data,
- **ID** - unikátní identifikátor výrobce,
- **Adresa** - sériové číslo jednotky které slouží pro adresaci,
- **Data** - data aplikační vrstvy,
- **RSSI** (Received Signal Strength Indicator) - indikátor síly signálu, volitelný.

Tab. 1.3: Bezdrátové technologie krátkého dosahu [6].

Technologie	Typy podsítí	Rychlost	Úsporné	Použití	Typ dat	Koncová zařízení
ZigBee	Inteligentní smíšené podsítě	Nízká	Ano	Senzory, monitorování	Senzory, monitorování	Inteligentní měřící zařízení
Bluetooth	Domácí a kancelářské podsítě	Nízká	Ano	Sdílení hudby	Hlas, data s nízkou četností, hudba	Chytré telefony
UWB (Ultra-Wideband)	Domácí a kancelářské podsítě	Vysoká	Ne	Sdílení hudba a videa	Video, data s vysokou četností	Videokamery, video projektory
IEEE 802.15.6	Podsítě v oblasti lidského těla	Nízká	Ano	Zdravotnictví	Biomedicínská data	Zdravotní pomůcky
Wi-Fi	Domácí a kancelářské podsítě	Vysoká	Ne	Domácí brány, připojení k internetu	VoIP, data, video	Počítače, chytré telefony
Wireless M-BUS	Podsítě pro senzory s hvězdicovou topologií	Nízká	Ano	Senzory, monitorování	Senzory, monitorování	Inteligentní měřící zařízení
Femtocell	Buňkové stanice s nízkým výkonem	Vysoká	Ne	Buňkové telefony	VoIP, data, video	Domácí Node B
HomeRF (Home Radio Frequency)	Systémy domácí zábavy	Vysoká	Ne	Hlasové služby	Hlas, data	Videotelefony

2 AGREGACE DAT M2M KOMUNIKACE

S prudkým nárůstem objemu M2M komunikace začíná otázka agragace dat nabývat na důležitosti. Začínají proto vznikat pracovní skupiny zabývající se touto problematikou, každá ze skupin však přichází s proprietálním řešením. Jelikož je vývoj těchto mechanismů na počátku, nebyl v této oblasti doposud definován žádný globální standard.

Oblast výzkumu se zaměřuje zejména na problematiku nalezení nejvýhodnějších cest ke koncovým uzlům (senzorům) a snížení datového toku při sestavování směrovacích tabulek [9]. Pracuje se také na zabezpečovacích agregačních mechanismech, jejichž úkolem je potlačit datový tok škodlivých koncových zařízení [32]. Taková zařízení, se snaží zahltit koncentrační prvek vysokým datový tokem tak, aby již nemohl přijímat data z jiných uzlů. Z pohledu této práce jsou však nejdůležitější mechanismy zajišťující přímou agregaci dat a prioritizaci důležitých datových toků.

Mezi zástupce poslední zmíněné skupiny patří například mechanismus CSDA (Complex Sensor Data Aggregation) [23], který je podrobněji popsán v kapitole 2.1.

2.1 Mechanismus CSDA

Většina současných řešení agragace dat na MTCG zařízeních je v dneší době napsána v jazyce `C` a je neoddělitelnou součástí firmware. Vývoj agregačních mechanismů je proto pro poskytovatele velmi obtížný jelikož musí znát přesné specifikace MTCG - je však také omezen výkonostními parametry zařízení. Modifikace agragačního mechanismu tak vyžaduje aktualizaci celého firmware zařízení, která je velmi riskantní.

CSDA proto nevyužívá jazyka `C`, ale agregační mechanismy jsou implementovány skrze konfigurační XML soubory [23].

2.1.1 Problémy agregačních mechanismů ve firmware

Jak bylo uvedeno v předchozím textu, vetšina současných agregačních mechanismů je pevnou součástí firmware zařízení. To přináší dva hlavní problémy: vývoj a aktualizace programů ve firmware [23].

Vývoj firmware

Zde se vyskytují dva základní problémy. Prvním jsou omezené systémové prostředky zařízení. Z důvodu vysokého počtu těchto zařízení je kladen důraz zejména na co nejnižší cenu. To pak úzce souvisý s nízkými hardwarovými parametry těchto zařízení. Jelikož je obtížné využít na těchto zařízeních vyšší programovací jazyky, je

většinou využito jazyka `C`. Avšak programování v jazyce `C` je obtížné a také správa paměti je problémová.

Druhý problém zahrnuje vývojová prostředí MTCG, která se liší v závislosti na použitém zařízení. Vývoj se typicky skládá z fází: vývoj agragační logiky na počítači, křížová-kompilace pro bránu, tvorba obrazu firmware a nahrání obrazu do MTCG. Všechny tyto procesy jsou rozdílné pro každou bránu, jelikož systém a uživatelské aplikce nejsou standardizované [23].

Aktualizace firmware

V případě potřeby aktualizovat firmware MTCG po nasazení do provozu mohou vzniknout v aktualizačním procesu komplikace při provádění následujících kroků.

- **Modifikace programu** – je třeba provést změny v kódu z důvodu změny agregační logiky. Zde nastávají problémy uvedené v předchozím textu. Navíc program vyžaduje opakované testování z důvodu předcházením regresím.
- **Instalace firmware** – nový firmware musí být nainstaován na MTCG. Existují dva způsoby. Prvním je přímá instalace na zařízení, to však při dnešním množství zařízení není téměř možné. Druhý způsob je využití konfoguračních protokolů jako TR-069. V tomto případě musí být celý firmware odeslán skrze senzorovou síť, která může být při přenosu takto velkého množství dat zahlcena.
- **Zotavení po chybě** – při aktualizaci nového firmware ze sítě může nastat neočekávaný problém s napájením, které vyústí v poškození obrazu (firmware). Proto musí MTCG obsahovat záložní paměťovou oblast, která má nejméně velikost firmware [23].

2.1.2 Obecné mechanismy agregace

Proces agregace probíhá na MTCG ve třech krocích. Jako vstupy procesu slouží adresy koncových uzlů (senzorů) a jejich hodnoty. Agregace může být rozdělena do tří kategorií: statistické výpočty pro redukci velikosti dat, filtrování pro snížení frekvence dat a zřetězení pro snížení přenosu hlavičkových souborů [23].

Statistické výpočty

Velikost dat je snížena výpočtem statistických hodnot, jako jsou průměr a histogram hodnot v definovaném časovém rámci. Statistické výpočty mohou být provedeny několikrát, může být zkombinováno také více hodnot od různých senzorů. Například při vyhodnocení odchylky teplot, jsou rozdíly spočítány z hodnot několika senzorů a na závěr sečteny [23].

Filtrování

Datový tok z MTCG může být snížen filtrováním dat s nízkou prioritou. Filtrování je rozděleno do dvou podkategorií: vstupní (využívá adresu senzorů) a prahové.

- **Filtrování adres** – pro získání hodnot z důležitých senzorů, MTCG příjmá data pouze ze senzorů se specifickou adresou.
- **Prahové filtrování** – účelem tohoto filtrování je získání dat která naznačují problém. Např. pokud teplota překročí určitý práh, získaná hodnota je odeslána na server. V opačném případě nejsou data odeslána [23].

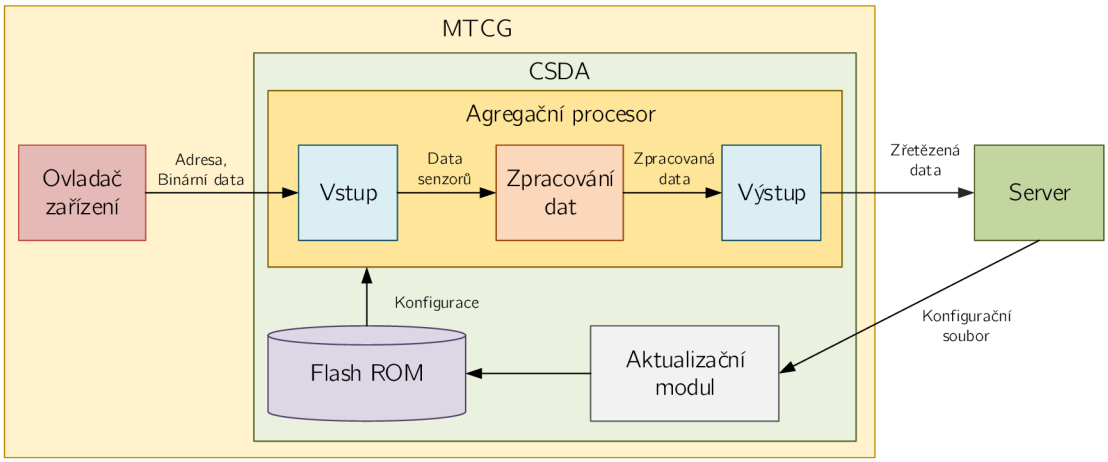
Zřetězení

Při odesílání dat na server, je přidána hlavička transportního protokolu. Z důvodu snížení datového toku, je proto spojeno několik hodnot z různých senzorů do jednoho bloku a odesláno najednou [23].

2.1.3 Architektura CSDA

Jak bylo zmíněno v úvodu této kapitoly, je obtížné a riskantní aktualizovat agregční mechanismy které jsou součástí firmware MTCG. CSDA proto přichází s řešením tohoto problému, jednotlivé moduly CSDA mechanismu jsou zobrazeny na obrázku Obr. 2.1.

Agregační procesor, jehož chování lze měnit úpravou konfiguračního souboru, odstraňuje nutnost programování. Aktualizační modul [23] umožňuje změnu logiky agregace skrze komunikační síť. Návrh agregátoru a konfigurace je popsán v následujících sekcích.



Obr. 2.1: Architektura CSDA.

Agregační procesor

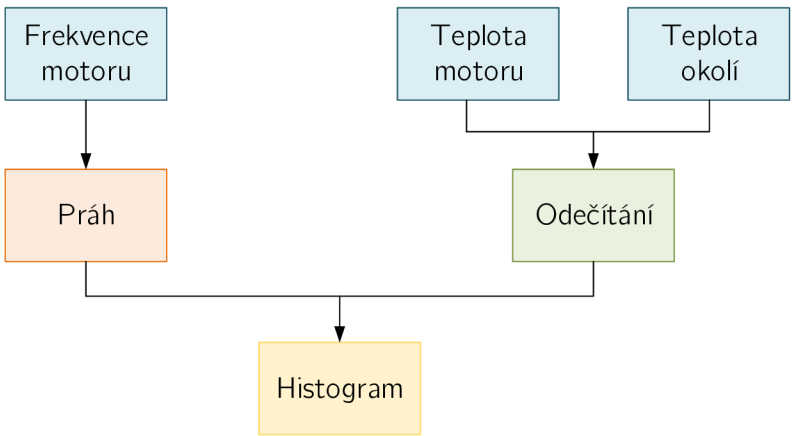
Implementace agregačních mechanismů, popsaných v sekci 2.1.2, je realizována v agregačním procesoru ve třech krocích: vstup, zpracování dat a výstup. Nezbytná logika pro zpracování každého kroku je předem definována v agregačním procesoru. Logika zpracování je definována v konfiguračním souboru, jehož struktura je definována jazykem XML.

- **Vstup** – ze zařízení jsou přijata data včetně adres koncových uzlů a binárních dat (včetně hodnot). Dále jsou vyfiltrovány adresy jednotlivých zařízení a z binárnách dat jsou extrahovány hodnoty senzorů.
- **Zpracování dat** – na hodnotách získaných ze senzorů jsou provedeny statistické výpočty jako průměr, minimum, maximum a histogram. Je provedeno také prahové filtrování, které může být kombinováno se statistickými výpočty.
- **Výstup** – v procesu zpracování dat je vytvořeno několik sekvencí výstupních dat. Tyto data jsou zřetězena a periodicky odesílána na server, nebo odeslána okamžitě pokud jsou urgentní [23].

Konfigurační jazyk

Agregační procesor generuje jeden výstup z několika vstupních toků. Obr. 2.2 ilustruje případ, kdy jsou teploty motoru shromažďovány pouze při vysokém zatížení. Ve vstupním modulu jsou extrahovány tři hodnoty senzorů. Pokud hodnota frekvence překročí práh, teplota prostředí je odečtena od teploty motoru pro potlačení vlivu prostředí. Nakonec je vytvořen histogram tohoto rozdílu.

Struktura tohoto modelu, kdy je vytvořen jeden výstup z několika vstupů odpovídá struktuře stromu. Proto lze adaptovat strukturu jazyka XML, který má také stromovou strukturu [23].



Obr. 2.2: Příklad agregačního procesu.

Aktualizační modul

Tento modul se stará o stažení a zapsání nové konfigurace na MTCG. Je obsažena také technologie pro zotavení po neúspěšné aktualizaci konfigurace. To je zajištěno oddělením agregačního procesoru a aktualizačního modulu do samostatných procesů. Aktualizace probíhá ve třech krocích:

1. Aktualizační modul stáhne konfigurační soubor do operační paměti a nakopíruje starou konfiguraci do flash paměti. Ta slouží jako záloha aktuální konfigurace.
2. Nová konfigurace je zapsána do flash paměti.
3. Aktualizační modul restartuje agregační procesor.
4. Agregační procesor načte novou konfiguraci z flash paměti.

I když se nepodaří uložení nové konfigurace do flash paměti, z důvodu výpadku napájení, konfigurace může být obnovena ze zálohy. Bez ohledu na to, zda se stahování nové konfigurace nepodařilo. Velikost flash paměti, vyžadované pro zálohu konfigurace, je přinejmenším rovna velikosti konfiguračního souboru [23].

2.1.4 Implementace na MTCG

Jelikož MTCG zařízení nemusí být výkonné zařízení poskytuje CSDA metody pro snížení využití operační paměti a procesoru. Mezi tyto metody patří buffering dat a binární konfigurace [23].

Buffering dat

Všechny přijaté hodnoty se musejí porovnat s konfiguračním souborem. Zde může nastat problém s vyčerpáním systémových prostředků při vysokém počtu současně přicházejících dat, každá hodnota musí být porovnána s konfiguračním souborem.

Proto CSDA implementuje buffering dat, kdy se vždy zpracováva několik přijatých dat najednou. Díky tomu je možné aby konfiguračním souborem prošlo několik hodnot najednou. Tímto postupem se ztrácí real-time zpracování dat, avšak zpoždění v řádu jednotek milisekund není důležité jelikož časová prodleva přenosu dat na server je vyšší [23].

Binární konfigurace

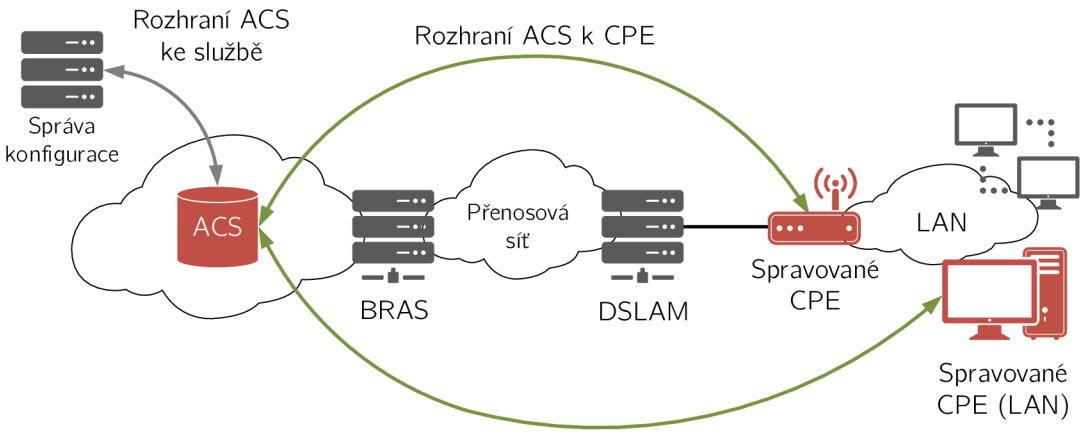
Parsování XML souborů na MTCG zařízení je obtížné, jelikož vyžaduje vysoké nároky na operační paměť. Kvůli paměťové úspoře je XML soubor předem převeden do binární podoby. Binární formát může být proveden v pořadí od shora dolů, což předchází výkonnostním ztrátám způsobeným náhodným přístupem do paměti [23].

3 VZDÁLENÁ KONFIGURACE

S rostoucím počtem M2M zařízení je velmi náročné měnit nastavení každého zařízení individuálně. Nicméně je stále důležité udržovat na zařízeních aktuální nastavení a software z bezpečnostních důvodů, na které je v dnešní době kladen stále větší důraz [2]. Proto vznikly pracovní skupiny zabývající se vzdálenou konfigurací síťových prvků. Výsledkem tohoto výzkumu jsou automatické konfigurační protokoly z nichž nejpoužívanější je TR-069 [4].

3.1 Funkční komponenty TR-069

TR-069 definuje protokol pro zabezpečenou automatickou konfiguraci koncového zařízení CPE (Customer Premise Equipment) využívající auto-konfigurační server ACS (Auto-Configuration Server), umístění jednotlivých prvků v architektuře TR-069 je zobrazeno na Obr. 3.1. Protokol TR-069 podporuje řadu funkcí pro vzdálenou správu zařízení (automatická konfigurace, správa software, sledování stavu a výkonu).



Obr. 3.1: Umístění prvků v architektuře TR-069 [4].

3.1.1 Automatická konfigurace

Protokol umožňuje, aby server poskytoval informace o jednom nebo více koncových CPE, dle různých kritérií. Tento mechamismus umožňuje poskytování parametrů, ale také možnost přidání dalších parametrů dle specifikací výrobce.

Protokol umožňuje poskytnout informace o CPE v době navázání spojení ale také kdykoli v následující době. To obsahuje také definici pro asynchronní, serverem inicializované, znovu poskytnutí informací o CPE [4].

3.1.2 Správa software/firmware

TR-069 poskytuje nástroje pro řízení stahování nového software/firmware. Protokol umožňuje identifikaci verzí, spuštění stahování a také oznámení o průběhu stahování obrazu. Je zde také přítomna funkce digitálně podepsaných souborů, které mohou sloužit jako instalační instrukce pro koncová zařízení. Digitálně podepsané balíčky zajišťují také integritu stažených dat [4].

3.1.3 Sledování stavu a výkonu

Je definována sada parametrů, které mohou sloužit pro kontrolu stavu a výkonu koncového zařízení. TR-069 také poskytuje možnost sledovat parametry, které nejsou součástí protokolu. Pro tyto účely je definován syntax pro definici přídavných parametrů, které mohou být monitorovány [4].

3.1.4 Diagnostika

Protokol TR-069 definuje sadu parametrů, které jsou dostupné pro diagnostiku spojení nebo problému se službami. Je zde také aplikován mechnismus pro definici výrobcem požadovaných parametrů [4].

3.2 Architektura TR-069

V architektuře protokolu TR-069 je zahrnuto několik součástí, které jsou unikátní pro tento protokol např. RPC (Remote Procedures Calling), ale využívá také sadu standardních protokolů SOAP (Simple Object Access Protocol), HTTP (Hypertext Transfer Protocol), SSL/TLS (Secure Sockets Layer/Transport Layer Security), TCP/IP (Transmission Control Protocol/Internet Protocol) [4]. Přehled využívaných protokolů lze nalézt v Tab. 3.1.

3.2.1 Mechanismy zabezpečení

Protokol TR-069 je navržen tak, aby zajišťoval co nejvyšší míru zabezpečení. Obsahuje metody zajišťující ochranu před manipulacemi s transakcemi na cestě mezi serverem a koncovým zařízením, poskytuje utajení těchto transakcí a umožňuje několik úrovní autentizace. Protokol poskytuje tyto metody zabezpečení [4].

- Podpora SSL/TLS pro komunikaci mezi ACS a CPE. Tento mechanismus zajišťuje důvěrnost transakcí, integritu dat a umožňuje autentizaci mezi CPE a ACS pomocí certifikátů.
- Vrstva HTTP podporuje autentizaci na základě sdílených klíčů.

Inicializační modely zabezpečení

Inicializace zabezpečovacích mechanismů je popsána z důvodu různých obchodních modelů pro distribuci koncových zařízení. Jsou uvažovány tři modely [4].

- Distribuce CPE poskytovatelem služeb s pevně definovaným ACS serverem.
- Maloobchodní distribuce zařízení, kde je provedena asociace zařízení s poskytovatelem služeb a zákazníkem v době prodeje.
- Maloobchodní distribuce, kde není provedena asociace zařízení.

Tab. 3.1: Souhrn protokolů využívaných v TR-069 [4].

Vrstva	Popis
CPE/ACS aplikace	Tato vrstva je definována individuálně na každém zařízení a není proto součástí standardu.
RPC metody	Vzdálené volání procedur, je definováno protokolem TR-069. Každé zařízení musí osahovat seznam parametrů, které jsou přístupné pro ACS. Seznam parametrů je definován v jednotlivých datových modelech.
SOAP	Standardní XML (Extensible Markup Language) zápis, použitý pro volání vzdálených procedur.
HTTP	Standardní HTTP.
TCP/IP	Standardní TCP/IP.

3.3 Komponenty architektury

Vzdálené volání procedur RPC definuje seznam parametrů a metod, které musí být obsaženy na koncovém zařízení. V následujících sekcích je uveden souhrn nejdůležitějších komponent [4].

3.3.1 Parametry

Specifikace RPC definuje mechanismus, jakým mají být vyčítány nebo ukládány parametry ke konfiguraci a monitorování stavu zařízení. Každý parametr se sestáva z páru `název-hodnota`, kde název označuje příslušný parametr. Parametry mají hierarchickou strukturu a jsou odděleny tečkou.

Parametr může být pouze pro čtení nebo může umožňovat i zápis. Parametry umožňující jen čtení jsou používány pro získání informací o CPE, jako stav zařízení nebo sběr statistik. Parametry s možností zápisu mohou upravovat aspekty

činnosti koncového zařízení. Všechny parametry s možností zápisu musí umožňovat také čtení.

Protokol TR-069 podporuje také metodu pro automatické nalezení parametrů podporovaných na CPE [4].

3.3.2 Přenos souborů

V protokolu TR-069 je implementována podpora stahování i nahrávání souborů, pro různé účely, nejčastěji aktualizace software.

Po úspěšné inicializaci spojení mezi CPE a ACS a úspěšném nalezení cesty k souboru může být využito různých protokolů pro přenos souboru. Nejčastěji je využito HTTP, ale může být použito HTTPS (Hypertext Transfer Protocol Secure), FTP (File Transfer Protocol) nebo TFTP (Trivial File Transfer Protocol) [4].

3.3.3 CPE - oznámení o navázání spojení

Protokol definuje mechanismus, který umožňuje informovat ACS o změnách provedených na CPE. Je také požadavek na co nejnižší frekvenci komunikace z důvodu využití přenosového pásma.

Tento mechanismus umožňuje počáteční nastavení koncového zařízení a také obsahuje logiku pro navázání periodické komunikace se serverem. Ta je důležitá z důvodu informování serveru o provedených změnách.

Při každém navázání spojení se CPE identifikuje serveru pomocí svého sériového čísla a kódu výrobce [4].

3.3.4 ACS - oznámení o navázání spojení

Velmi důležitým parametrem je možnost asynchronně informovat koncové zařízení o změnách provedených na serveru. To umožňuje použít tento auto-konfigurační protokol v systémech pracujích blízko režimu reálného času. Koncovému zařízení to umožňuje téměř okamžitě přistupovat ke službám na serveru bez čekání na další oznamovací interval [4].

3.4 Procedury a požadavky

Protokol TR-069 definuje povinné procedury, které jsou nutné např. pro nalezení ACS, navázání spojení nebo vytvoření relace.

3.4.1 Nalezení ACS

Pro získání adresy serveru může být použito několik metod, které jsou uvedeny níže.

1. Využití DNS (Domain Name System) pro získání IP adresy serveru, z URL (Uniform Resource Locator) adresy, která je uložena v CPE.
2. Lze využít část IP vrstvy pro automatickou konfuguraci. To znamená využití DHCP (Dynamic Host Configuration Protocol), jehož zprávy obsahují URL serveru a pomocí DNS získá CPE IP adresu serveru.
3. CPE může obsahovat výchozí adresu ACS, na kterou se odkazuje pokud není nastavena jiná URL adresa [4].

3.4.2 Navázání spojení

Po úspěšném získání adresy auto-konfiguračního serveru je nutné navázat spojení, to může navázat CPE nebo ACS. Avšak v některých případech lze navázat spojení pouze ze strany CPE z důvodu využití NAT (Network Address Translation).

Navázání spojení od CPE

Koncové zařízení může kdykoliv navázat spojení při znalosti adresy ACS. Povinně však musí navázat spojení v těchto případech.

- Při prvním připojení CPE do sítě a prvotní konfiguraci.
- Při spuštění zařízení.
- Při každém periodickém intervalu uvedeném v nastavení.
- Když je požadováno metodou.
- Při změně adresy ACS.
- Při každé změně parametrů.

Z důvodu předcházení zahlcení auto-konfiguračního serveru je na každém CPE uveden maximální limit četnosti informování o změnách parametrů [4].

Navázání spojení od ACS

Navázání spojení směrem od ACS k CPE může být provedeno pomocí mechanismu oznámení o požadavku na spojení. Na straně CPE je tento parametr povinný na ACS je doporučený.

Možnost navázat spojení ze strany serveru je realizovatelná pouze v případě, kdy je adresa CPE veřejně dostupná. V opačném případě může spojení navázat pouze klient [4].

3.4.3 Povinné zprávy vzdáleného volání

Souhrn všech metod, které jsou definovány vrstvou vzdáleného volání procedur je uveden v Tab. 3.2.

Tab. 3.2: Povinné zprávy vzdáleného volání procedur [4].

Název metody	Povinnost na CPE	Povinnost na ACS
CPE metody	Odpovědi	Volání
GetRPCMethods	Povinné	Volitelné
SetParameterValues	Povinné	Povinné
GetParameterValues	Povinné	Povinné
GetParameterNames	Povinné	Povinné
SetParameterAttributes	Povinné	Volitelné
GetParameterAttributes	Povinné	Volitelné
AddObject	Povinné	Volitelné
DeleteObject	Povinné	Volitelné
Reboot	Povinné	Volitelné
Download	Povinné	Povinné
Upload	Volitelné	Volitelné
FactoryReset	Volitelné	Volitelné
GetQueuedTransfers	Volitelné	Volitelné
ScheduleInform	Volitelné	Volitelné
SetVouchers	Volitelné	Volitelné
GetOptions	Volitelné	Volitelné
Metody serveru	Volání	Odpovědi
GetRPCMethods	Volitelné	Povinné
Inform	Povinné	Povinné
TransferComplete	Povinné	Povinné
RequestDownload	Volitelné	Volitelné
Kicked	Povinné	Volitelné

3.4.4 Správa relace

Všechny relace musí začínat informační zprávou od CPE obsahující inicializační HTTP zprávu. Ta slouží k nastavení komunikačních limitů klienta a jeho kódování.

Relace je ukončena pokud nemá server ani klient více požadavků k odeslání a nezbývá odpovědět na žádný z požadavků. Mezi každým ACS a CPE může v jeden čas existovat pouze jedna relace [4].

CPE - Iinicializace relace

Po navázání spojení zahajuje CPE relaci zasláním inicializačního požadavku na server. Tato zpráva indikuje, že je koncové zařízení připraveno přijímat požadavky ze serveru.

Při této úvodní komunikace je povoleno pouze zasílání SOAP obálky. V argumentech zprávy je obsažen parametr maximálního počtu obálek, které lze zpracovat v následném HTTP požadavku.

CPE může inicializovat spojení jen v případě, že má uzamknuty parametry přístupné skrze jeho rozhraní. Důvodem je ochrana před změnou parametrů jiným mechnismem. Tento zámek může přetrvávat až do konce relace [4].

CPE - Přichozí požadavky

Koncové zařízení odpovídá na dotazy v pořadí v jakém byly přijaty. Pro předcházení uváznutím CPE nečeká na potvrzení předchozího požadavku od serveru než odešle nový požadavek [4].

CPE - Odchozí požadavky

Pokud má CPE požadavky na server může jej odeslat v libovolné pořadí s ohledem na odpovědi odesílané z CPE na ACS.

Pokud klient přijme požadavek s `HoldRequest` s hodnotou `true` musí vyčkat s odesláním dalšího požadavku do doby, než se zmení pole `HoldRequest` na `false`, pokud obdrží CPE požadavek s prázdnou hlavičkou může vyhodnotit `HoldRequest` jako `false`.

Pokud jsou všechny požadavky ACS vyřešeny a CPE nemá další požadavky, musí odeslat alespoň jeden požadavek či odpověď na server [4].

CPE - Ukončení relace

Klient musí ukončit relaci pokud jsou splněny následující podmínky.

1. Server nemá žádné další požadavky.
2. CPE nemá žádné další požadavky.
3. Klient přijal všechny nevyřízene požadavky ze serveru.
4. Klient odeslal všechny požadavky na ACS.

CPE musí ukončit spojení pokud nepřijal žádnou odpověď od serveru za dobu větší než 30 sekund.

Pokud nejsou tyto požadavky splněny klient musí pokračovat v relaci.

V případě, že je relace neočekávaně ukončena musí být provedena inicializce relace od začátku [4].

ACS - Iinicializace relace

Po přijetí informačního požadavku od CPE musí server reagovat informační odpovědí.

Pokud dokáže CPE pracovat s více než jednou obálkou, server přidá do odpovědi maximální počet obálek, který může být v relaci zasílán [4].

ACS - Přichozí požadavky

Server odpovídá na požadavky CPE v pořadí jakém byly přijaty. Pro předcházení uváznutí nesmí ACS čekat s odpovědí na požadavek do doby než dojde potvrzení předchozího požadavku od CPE [4].

ACS - Odchozí požadavky

Pokud má server požadavky k odeslání, měl by je odeslat v pořadí respektujícím odpovědi odeslané serverem klientovi. Server může přidat jeden i více dotazů do posloupnosti a odeslat je klientovi. Není definován limit počtu požadavků, které může server odeslat před přijetím odpovědí od klienta.

Pokud má ACS jeden či více požadavků k odeslání, nebo zbývají odeslat odpovědi na předchozí dotazy, server musí odeslat alespoň jeden dotaz/odpověď na CPE. Prázdná HTTP odpvěď je možná pouze pokud nemá server více dotazů, nebo nezbývá odeslat žádné odpovědi [4].

ACS - Ukončení relace

Pokud je spojení vytvořeno klientem, jen ten je odpovědný za ukončení spojení. Server může zvážit ukončení spojení pokud jsou splněny následující požadavky.

1. Klient nemá další požadavky.
2. Server nemá další požadavky.
3. Klient odeslal všechny zbývající odpovědi na ACS.
4. Server přijal všechy zbývající požadavky od CPE.

Pokud nejsou splněny výše uvedené požadavky, ale server nepřijal žádnou odpověď od CPE déle než 30 sekund může zvážit ukončení spojení [4].

3.5 Datové modely v TR-069

Protokol TR-069 definuje několik typů zařízení popsaných datovým modelem, který obsahuje informace o vlastnostech a možnostech zařízení. Seznam datových modelů kompatibilních s TR-069 je uveden v Tab. 3.3.

Tab. 3.3: Datové modely využívane v TR-069 [5].

Datový model	Použití
TR-064 a TR-133	CPE zařízení v lokální síti.
TR-068 a TR-124	Internetové brány a modemy.
TR-098	Model internetové brány pro TR-069.
TR-104	Poskytování parametrů pro VoIP (Voice over IP) CPE.
TR-106	Šablona datového modelu pro TR-069.
TR-110 a TR-133	Model pro vzdálenou správu domácí sítě.
TR-122	Model pro ATA (Analog Telephone Adapter) zařízení.
TR-126	Triple-Play služby kvality zážitku.
TR-128 a TR-123	Testovací podpora pro TR-069.
TR-131	Požadavky na ACS rozhraní do vyšší vrstvy.
TR-135	Datový model pro set-top boxy.
TR-140	Datový model pro zařízení poskytující služby úložiště.
TR-142	Datový model pro zařízení pasivní optické sítě.
TR-143	Model pro testování výkonnosti a statistické monitorování sítě.
TR-157	Model pro UPnP (Universal Plug and Play) a DLNA (Digital Living Network Alliance) zařízení.
TR-181	Model datových zařízení pro TR-069.
TR-196	Datový model pro femto přístupové body.

3.6 Dostupné implementace ACS

Téměř každý výrobce koncových zařízení podporujících TR-069 pužívá vlastní implementaci automatického konfiguračního serveru. Tyto implementace jsou vetšinou proprietární a umožňují využití plného potenciálu TR-069 pouze s vlastním klientem, nebo nepodporují všechny RPC metody. Existuje i několik implementací ACS s otevřeným zdrojovým kódem umožňujícím upravit/přidat TR-069 metody tak, aby byla zaručena plná kompatibilita s klientem. Mezi takovéto ACS patří například genieacs 3.6.1 [10] nebo jCWMPServer 3.6.2 [16].

3.6.1 Genieacs

Tento ACS je postaven na moderních technologiích jako jsou Node.js, Redis a MongoDB. Správa TR-069 zařízní může být prováděna v přehledném grafickém rozhraní nebo v konzoli. Genieacs je navržen jako technologicky nezávislý ACS umožňující rozšíření funkčnosti pro potřeby jednotlivých poskytovatelů. V současné době genieacs neumožňuje autentizaci CPE pomocí uživatelského jména a hesla, ale přijme jakoukoliv příchozí HTTP/HTTPS komunikaci [10].

Architektura

Genieacs obsahuje tři různá rozhraní starající se o správu protokolem TR-069.

- `genieacs-cwmp` – slouží pro komunikaci mezi CPE a ACS, rozhraní se stará o vytvoření, řízení relace a o samotnou konfiguraci protokolem TR-069.
- `genieacs-nbi` – toto rozhraní slouží pro komunikaci s vnějšími aplikacemi a poskytuje například REST API (Representational State Transfer Application Programming Interface) pro uživatelské rozhraní. Pomocí tohoto rozhraní může být CPE spravováno přes konzoli.
- `genieacs-fsi` – rozhraní slouží jako server pro stahování souborů z ACS [10].

Grafické rozhraní

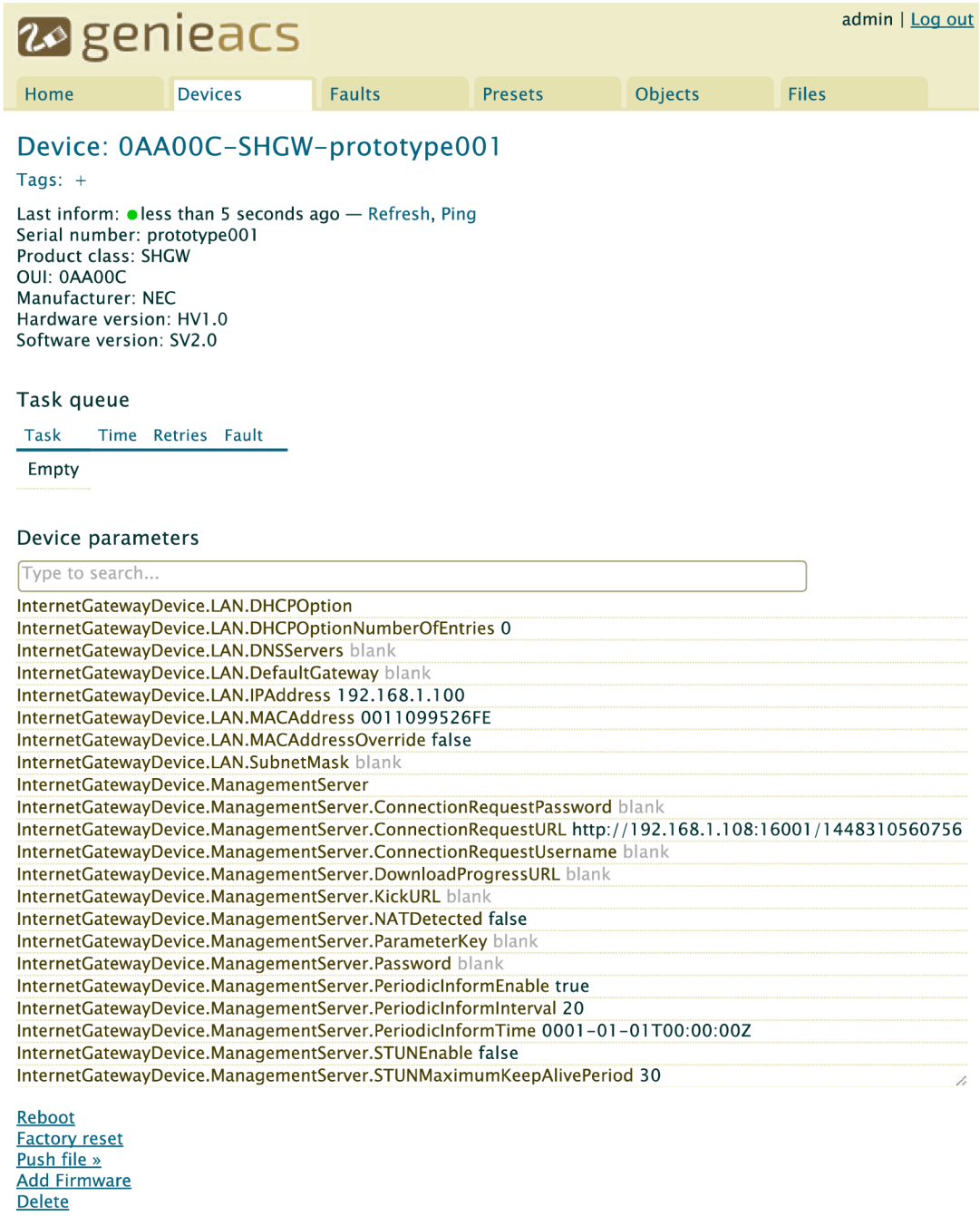
Uživatelské rozhraní poskytuje přehledné prostředí pro správu TR-069 zařízení, jak lze vidět na Obr. 3.2. Rozhraní je napsáno v jazyce Ruby s využitím frameworku Rails 4, je plně nezávislé na jádře ACS, se kterým komunikuje skrze REST API. Pro správnou funkci vyžaduje spuštění rozhraní `genieacs-nbi`, bez tohoto rozhraní není možné koncové zařízení ovládat [10].

Konfigurace

Pro konfigraci genieacs slouží tři konfigurační soubory ve formátu JSON a jeden ve formátu JavaScript.

- `config.json` – nejdůležitější konfigurační soubor, obsahující url adresu databáze a definici portů na kterých jsou spuštěna jednotlivá rozhraní.
- `parameters.json` – soubor slouží pro vytvoření zástupných názvů paramatrů a jejich datových typů, ve kterých lze poté hledat podle klíčových slov či datových typů.
- `custom_commands.json` – pomocí tohoto souboru lze definovat příkazy které umožní použití rozdílných technologií pro TR-069 konfiguraci. ACS může například komunikovat s TR-069 zařízením skrze protokol Telnet.

- `auth.js` – tento soubor slouží pro autentizaci ACS k CPE. Obsahuje uživatelské jméno a heslo, kterým se ACS autentizuje CPE. Autentizace CPE k ACS není v současné době implementována a ACS odpovídá na všechny příchozí dotazy. Autentizaci lze realizovat s využitím proxy serveru, který přesměrovává dotazy na ACS [10].



Obr. 3.2: Grafické rozhraní genieacs.

Struktura příkazů

Genieacs je postaveno nad databází MongoDB, která nevyužívá relačních tabulek ale používá formát podobný JSON. Příkazy mají tedy strukturu podobající se dotazům na MongoDB.

Zařízení může na dotazy odpovědět stavovým kódem 200, pokud byl příkaz správně vykonán, nebo 202 pokud byl zařazen do fronty a bude vykonán v dalším periodickém informačním intervalu.

Příkazy pro získání a nastavení parametrů lze vidět na List. 3.1 v List. 3.2. Pro získání či nastavení parametrů je použita metoda POST. Příkazy jsou z konzole odesílány pomocí programu cURL ale může být použita jakákoliv jiná knihovna např. PHP (Hypertext Preprocessor), JavaScript [10].

Listing 3.1: Příkaz pro získání parametrů.

```
curl −i 'http://adresa_serveru:port/devices/id_zarizeni\
    −X POST \
    −−data '{ "name": "getParameterValues", "parameterNames": \
    ["parametr1", "parametr2"] }'
```

Listing 3.2: Příkaz pro nastavení parametru.

```
curl −i 'http://adresa_serveru:port/devices/id_zarizeni\
    −X POST \
    −−data '{ "name": "setParameterValues", "parameterValues": \
    [["parametr", "hodnota"]]}'
```

3.6.2 JCWMPServer

Tento ACS s otevřeným zdrojovým kódem je celý napsán v jazyce Java, bez použití jakéhokoliv frameworku, jedinou další závislostí nutnou pro spuštění je připojení do MySQL databáze[16].

Architektura

JCWMPServer se skládá ze serveru a části pro zpracování zpráv, které jsou mezi sebou velmi úzce propojené [16].

- `Server` – tato část se stará o navázání a řízení spojení s klientem, umožňuje také autentizaci pomocí jména a heše hesla tzv. Digest access authentication. Serverová část se také stará o komunikaci s databází. Výchozí databází je

MySQL, ale je možné využít jakoukoliv databázi, která implementuje rozraní `I_Database`.

- `Messaging` – v této sekci jsou zpracovány všechny přijaté zprávy a dle obsahu vykonány v jedné z tříd. Pro rozšíření funkcionality, je nutné v této části vytvořit třídy starající se o zpracování potřebných metod.

Uživatelské rozhraní

JCWMPServer nabízí i velmi jednoduché uživatelské rozhraní, které pro svůj běh vyžaduje PHP a správně nastavené připojení k MySQL databázi. Propojení mezi serverem a grafickým rozhraním je omezeno na minimum. Obě aplikace běží nezávisle a jediným spojujícím prvkem je databáze, nad kterou se provádí všechny operace.

Konfigurace

Pro konfiguraci slouží pouze jeden XML soubor `serverSetting.xml`, ve kterém jsou obsaženy všechny potřebné údaje IP adresa, port, autentizační klíč a jméno.

3.7 Dostupné implementace TR-069 klientů

Pro vzdálenou konfiguraci CPE existuje velké množství TR-069 klientů. Mnoho z nich je proprietárních a není možné rozšířit jejich funkcionalitu přidáním nových TR-069 metod, to umožňují klienti s otevřeným zdrojovým kódem. V následujících sekcích budou popsány tři dostupné implementace TR-069 klientů s otevřeným zdrojovým kódem: Freecwmp 3.7.1, EasyCwmp 3.7.2 a Modus TR-069 3.7.3. V tabulce Tab. 3.4 jsou srovnány dostupné implemetace TR-069 klientů z hlediska implementace metod vyžadovaných protokolem TR-069.

3.7.1 Freecwmp

Tento projekt se zaměřuje na co nejvyšší integraci TR-069 klienta v systému OpenWrt a snadnou rozšířitelnost pro mnohostranné použití. Freecwmp jako jedinný TR-069 klient šířený pod licencí GPLv2 (General Public License version 2) pracuje s OpenWrt bez jakýchkoliv dodatečných úprav [26].

Architektura

Freecwmp je rozdělen do dvou hlavních částí – jádra a skriptů. Jádro je nápsáno v jazyce `C` stará se o navázání a řízení komunikace se serverem. Kolekce skriptů slouží

pro vykonávání akcí vyvolaných ACS např. nastavení nebo získání parametrů, stažení souboru, nahrání souboru atd. Jádro a skripty jsou na sobě plně nezávislé, to umožňuje přidání nových funkcí bez nutnosti opakované úpravy a kompilace jádra [26].

Vlastnosti

Všechny standardní i nestandardní metody požadované protokolem TR-069 mohou být spojeny s kolecí skriptů na CPE. Cílový ACS může být definován pouze v čase kompilace a to z důvodu snížení velikosti klienta. Freecwmp umožňuje stažení a spuštění libovolného skriptu, lze také akualizovat, instalovat a odstraňovat balíčky opkg (Open PacKaGe Management), které jsou využívány balíčkovým systémem v OpenWrt [26].

3.7.2 EasyCwmp

Tento klient vycházející z projektu freecwmp 3.7.1, je vyvýjen společností PIVA Software a je šířen pod licencí GPLv2. Cílem tohoto projektu je vytvořit implemetaci plně odpovídající standardu TR-069 [18].

Architektura

Klient se sestává ze dvou částí – jádra a datového modelu. Jádro, které je napsáno v jazyce `C`, se stará o komunikaci s ACS a zajišťuje řízení, navázání spojení a vykonávání vzdálených procedur. Datový model slouží k popsání CPE a podporuje datové modely TR-098, TR-181, TR-104, TR-106 a TR-111.

EasyCwmp umožňuje dvě různé implementace datového modelu. Bezplatné řešení implementuje datový model v prostředí shell, komerční varianta využívá implementace v jazyce `C`. Komerční řešení je vhodné pro velké datové modely, kde by zpracování pomocí shell vyžadovalo mnohonásobně více času. Obě implementace datového modelu umožňují rozšířit model o vlastní objekty popisující vlastnosti CPE [18].

Vlastnosti

Klient umožňuje spoluprácí s POSIX (Portable Operating System Interface) kompatibilními systémy jako jsou Android, OpenWrt a Linuxové distribuce. Aplikace pracuje v jednom vlákně a může být spuštěna na rozličných architekturách ARM, x86, MIPS atd. Je implementována také zabezpečená komunikace SSL, přenos souborů je možný s využitím protokolů HTTP, HTTPS a FTP. Klient podporuje také komunikaci s využitím IPv6 adres [18].

3.7.3 Modus TR-069

Tento klient, vyvinutý společností Orange Labs, je napsán v jazyce Java dle standardu OSGi. To mu umožňuje velmi jednoduchou rozšířitelnost funkčnosti a aktualizaci metod [8].

Architektura

Jak lze vidět na Obr. 3.3, je každá metoda služby `RPCMethodMngService` implementována jako samostatný balíček (v terminologii OSGi bundle). Tento návrh umožňuje přidání nových funkcí vytvořením libovolného balíčku implementujícího metody rozhraní služby `RPCMethodMngService`. Všechny balíčky, implementující metody standardu TR-069 registrované u služby `RPCMethodMngService`, jsou přístupné skrze rozhraní `TR69ClientApi Bundle`.

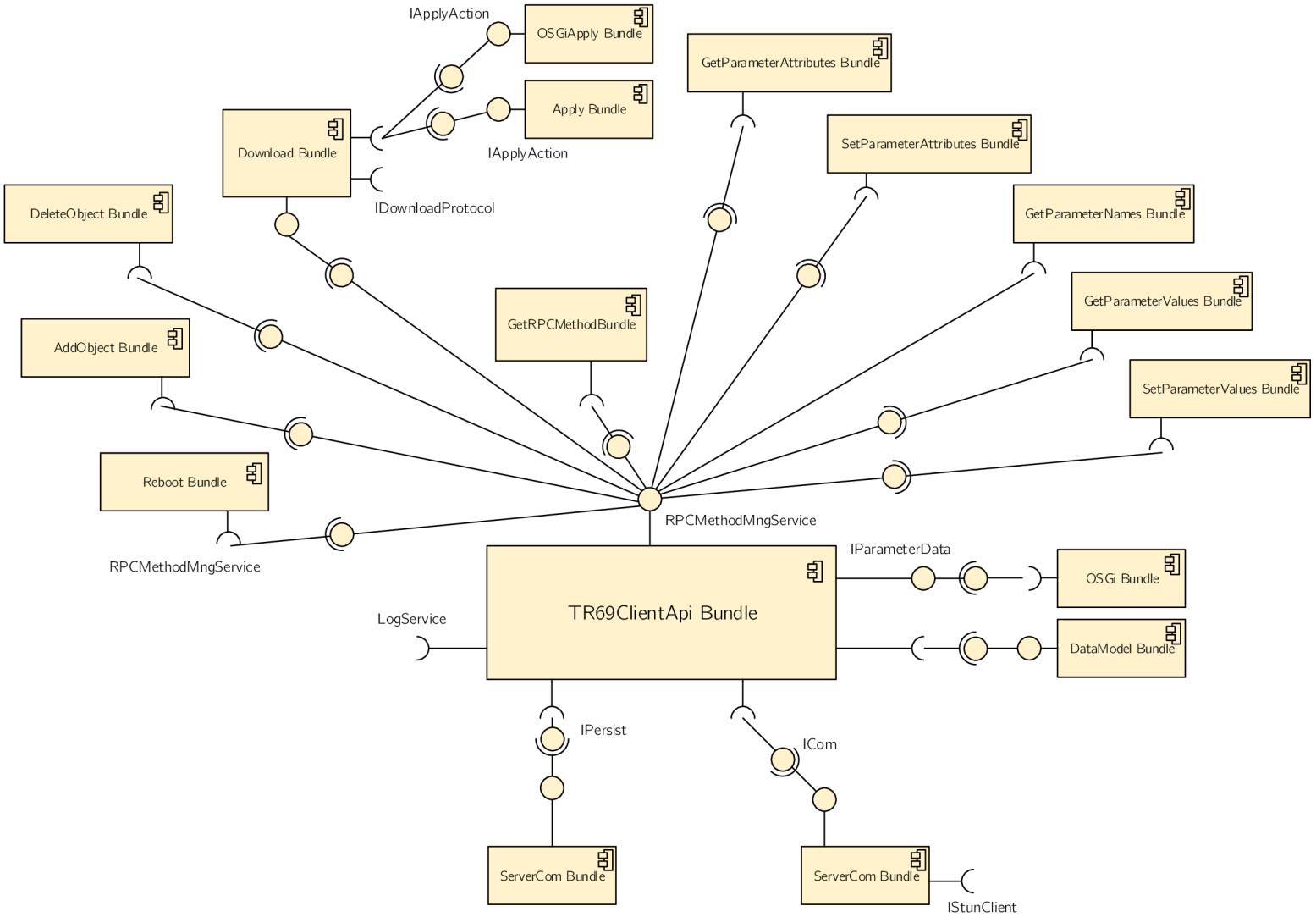
Balíček `TR69ClientApi Bundle`, dále poskytuje rozhraní pro moduly starající se o komunikacikaci se serverem `ICom`, ukládání dat `IPersist`, volbu datového modelu `IModel` a správů balíčků `IParameterData`.

- `OSGi Bundle` – tento balíček se stará o řízení (zastavení/spuštění, odstranění, instalace) balíčků poskytujících služby TR-069.
- `DataModel Bundle` – definuje použitý datový model, v této době je implementován model TR-106 definující objekty `Device` a `InternetGatewayDevice`.
- `FilePersist Bundle` – stará se o ukládání dat do paměti CPE.
- `ServerCom Bundle` – tento balíček funguje jako server starající se o sestavení a řízení komunikace s ACS [8].

Vlastnosti

Implementace klienta v jazyce Java umožňuje běh aplikace na všech architekturách podporujících běh virtuálního stroje JVM (Java Virtual Machine) a OSGi Frameworku (ARM, x86, částečně MIPS). Díky implementaci dle doporučení OSGi [25] obsahuje klient metody, které nejsou na jiných platformách realizovatelné např. aktualizace nebo instalace nových OSGi balíčků.

Velkou nevýhodou klienta je nemožnost použití šifrovaného spojení, které není v balíčku `ServerCom Bundle` implementováno [8].



Obr. 3.3: Architektura modus-tr69 [8].

Konfigurace

Pro konfifuraci klienta Modus TR-069 slouží tři soubory.

- `CSV.cfg` – obsahuje definici cest ke konfiguračním a logovacím souborům.
- `config.cfg` – v tomto souboru je uvedena definice kořenového zařízení a nastavení vhodného komunikačního rozhraní.
- `usine.txt` – obsahuje definice objektů dostupných na CPE dle zvoleného datového modelu, některé parametry jsou generované automaticky při spuštění a proto je nelze měnit [8].

Tab. 3.4: Porovnání dostupných TR-069 klientů [28].

	RPC CPE metody	EasyCwmp	netcwmp	freecwmp	cwmplclient	open-tr069	modus-tr069
Povinné	GetRPCMethods	■	■	■	■	■	■
	SetParameterValues	■	■	■	■	■	■
	GetParameterValues	■	■	■	■	■	■
	SetParameterAttributes	■	■	■	■	■	■
	GetParameterAttributes	■	■	■	■	■	■
	GetParameterNames	■	■	■	■	■	■
	AddObject	■	■	■	■	■	■
	DeleteObject	■	■	■	■	■	■
	Reboot	■	■	■	■	■	■
	Download	■	■	■	■	■	■
Volitelné	FactoryReset	■	■	■	■	■	■
	ScheduleInform	■	■	■	■	■	■
	Upload	■	■	■	■	■	■
	GetQueuedTransfers	■	■	■	■	■	■
	SetVouchers	■	■	■	■	■	■
	GetOptions	■	■	■	■	■	■

■	kompletní metoda
■	nekompletní metoda
■	metoda neexistuje

4 DATABÁZE PRO EMBEDDED ZAŘÍZENÍ

Pro sběr a ukládání M2M dat je vhodné v delším časovém intervalu použít místo klasických souborů databázové systémy, umožňující mnohem rychlejší práci s daty. V souvislosti s volbou vhodné databáze ovšem vyvstává problém výkonostních limitů ze strany embedded zařízení. Klasické databáze, známé z velkých serverů pracující na modelu klient-server (např. MySQL), mají vysoké nároky na systémové prostředky zařízení. Proto začaly vznikat embedded databáze (nevyžadují externí server), které jsou velmi často tvořeny jediným souborem [11].

To to řešení je mnohem méně náročné na systémové prostředky a je tedy vhodné pro použití v embedded zařízeních. Embedded databáze však nenabízejí tak pokročilé funkce jako databáze založené na modelu klient-server, to ale ve většině případů nebrání jejich nasazení. Mezi nejpoužívanější databáze patří H2 [12], Berkley DB [24], Raima Database Manager [27] a SQLite 4.1, která je použita v této práci.

4.1 Databáze SQLite

SQLite je dnes jednou z nejpoužívanějších databází a lze ji nalézt v systémech Android, iOS, Mac, Windows 10 a v aplikacích jako Skype, Chrome a Dropbox. Je široce podporována na všech dostupných architekturách x86, x64 a ARM i operačních sytémech Android, Linux, Windows, Mac OS. I když je SQLite napsána v jazyce C díky otevřenému zdrojovému kódu je její podpora implementována v mnoha jiných jazycích např. Java, Python, PHP [15].

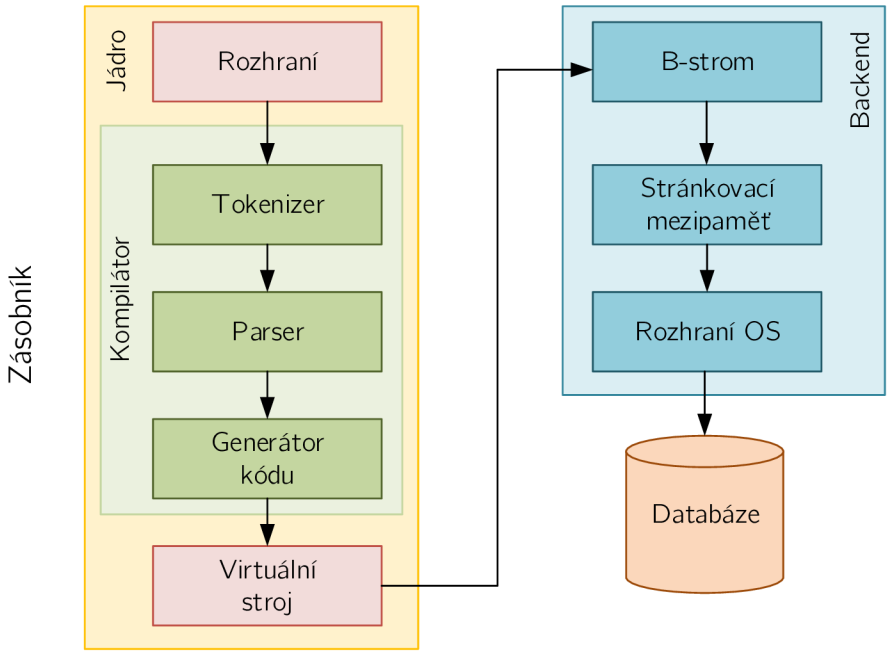
4.1.1 Vlastnosti

Databáze vyžaduje pouze minimální podporu externích knihoven a nulovou potřebu administrace databáze před jejím spuštěním. Z tohoto důvodu je velmi vhodná pro použití v embedded zařízeních. SQLite pracuje jako transakční databáze v níž jsou všechny operace atomické, konzistentní, izolované a odolné tzv. ACID (Atomicity Consistency Isolation Durability), to zaručuje správnost provedení transakcí i v případě nekorektního ukončení aplikace, systému či chyby napájení [15].

4.1.2 Architektura

Při návrhu SQLite databáze bylo využito modulární architektury, která přináší několik unikátních přístupů ke správě relačních databází. Datbáze se sestává z osmi samostatných bloků seskupených do tří hlavních subsystémů, jak lze vidět na Obr. 4.1. Tyto moduly rozdělují zpracování dotazů do diskrétních operací, které jsou postupně

vykonávány. Na vrcholu zásobníku se provádí kompilace dotazů, ve středu jsou dotazy vykonávány, spodní část zásobníku řídí ukládní souborů a spolupráci s operačním systémem [11].



Obr. 4.1: Architektura SQLite [11].

Rozhraní

Rozhraní se nachází na vrcholu zásobníku a sestává se z SQLite C API. Pomocí tohoto rozhraní mohou s databází komunikovat externí programy, skriptovací jazyky a knihovny. Toto rozhraní je využito také při komunikaci s konektorem SQLite JDBC, který slouží pro komunikaci s jazykem Java a databází SQLite. V tomto případě je pro komunikaci s rozhraním jazyka `C` využito JNI (Java Native Interface), které je součástí jazyka Java [11].

Překladač

Proces kompilace (překládání) začíná v modulech tokenizer a parser Obr. 4.1. Tyto dva moduly spolupracují na ověření syntaxe dotazů jazyka SQL (Structured Query Language) v textové podobě. Dále jsou dotazy převedeny do hierarchické datové struktury se kterou mohou nižší vrstvy architektury snadněji pracovat. Modul tokenizer je napsán ručně avšak modul parser je vytvořen pomocí SQLite generátoru

nazývaného Lemon. Parser Lemon je vytvořen pro co nejvyšší výkon a velký důraz je kladen na ochranu před paměťovými úniky. Jakmile je dotaz rozložen do tokenů, vyhodnocen a převeden do podoby parsovacího stromu, je strom poslán do generátoru kódu.

Generátor kódu přeloží parsovací strom do jazyka symbolických instrukcí specifického pro SQLite. Tento jazyk symbolických instrukcí se sestává z instrukcí, které jsou proveditelné pomocí virtuálního stroje, viz následující text. Jediným úkolem generátoru kódu je převedení parsovacího stromu do kompletního malého programu napsaného v jazyce symbolických instrukcí a předání tohoto programu do virtuálního stroje [11].

Virtuální stroj

Uprostřed zásobníku je umístěn virtuální stroj, někdy také nazývaný VDBE (Virtual Database Engine). VDBE je virtuální stroj založený na systému registrů pracující s bajtovým kódem, pracující nezávisle na použitém operačním systému, procesoru nebo systémové architektuře. Bajtový kód VDBE se sestává z více než 100 možných příkazů známých jako `opcodes`, sloužících pro databázové operace. VDBE je navržen speciálně pro zpracování dat. Každá instrukce v instrukční sadě slouží ke specifické databázové operaci např. vytvoření záznamu, získání sloupce nebo začátek transakce. Dohromady a ve správném pořadí dokáže instrukční sada VDBE provést jakýkoliv SQL příkaz, avšak složitě. Každý SQL výraz v SQLite (vybrání a aktualizace řádku, vytváření tabulky atd.) je nejprve přeložen do jazyka virtuálního stroje, který vytvoří posloupnost instrukcí odpovídající SQL příkazu.

V mnoha ohledech VDBE slouží jako jádro SQLite. Všechny moduly před virtuálním strojem pracují na vytvoření programu pro VDBE, zatímco včechny moduly za virtuálním strojem vykonávají tento program po jedné instrukci [11].

SQLite Backend

Backend se skládá z B-stromu, stránkovací mezipaměti a rozhraní operačního systému, viz Obr. 4.1. B-strom a stránkovací mezipaměť spolupracují na přesunu stránek, ani jeden však nezná stukturu souboru. B-strom slouží pro organizaci jednotlivých stránek a udržuje vazby mezi nimi, stránky organizuje ve stuktuře podobné stromu vhodné pro rychlé hledání. Mezipaměť poskytuje B-stromu stránky, její funkcí je přesun stránek z/na disk. Práce s diskem je stále nejpomalejší část procesu, a proto si snaží stránkovací mezipaměť ponechávat často používané stránky v paměti a tím snížit využití disku. Další funkcí stránkovací mezipaměti je řízení transakcí, zamknutí databáze a zotavení po pádu. Mnoho těchto funkcí je zprostředkováno pomocí operačního systému.

Procedury, jako je zamknutí souboru, jsou v různých systémech implementovány odlišně. Proto rozhraní operačního systému poskytuje abstraktní vrstvu, která tyto odlišnosti skrývá před ostatními moduly. To umožnuje snazší práci při portování SQLite databáze na různé systémy, všechny tyto odlišnosti jsou totiž popsány v rozhraní operačního systému [11].

4.1.3 Dynamické typování

Většina databází vycházejících z SQL využívá statické (rigidní) datové typování. Při statickém typování je hodnota určena svým kontejnerem, tzn. datový typ je pevně určen sloupcem tabulky.

SQLite využívá dynamičtější určení datových typů. Datový typ je deklarován samotnou hodnotou, ne svým kontejnerem. Dynamický datový typ je však zpětně kompatibilní s ostatními SQL databázemi pracujícími se statickým typováním, které mohou v SQLite pracovat statickým způsobem. Nicméně dynamické typování umožňuje práci se záznamy, která není v tradičních staticky definovaných databázích možná [15].

4.1.4 Datové typy a třídy

Každá hodnota, která je v databázi uložena nebo zpracována, je určena jednou z pěti datových tříd [15].

- **NULL** – určena pro `NULL` hodnoty.
- **INTEGER** – určen pro ukládání celých čísel v 1, 2, 3, 4, 6, nebo 8bajtových záznamech v závislosti na velikosti čísla.
- **REAL** – určen pro ukládání čísel v pohyblivé desetiné čárce v 8bajtových záznamech dle standardu IEEE.
- **TEXT** – určen pro ukládání textových řetězců s kodováním UTF-8, UTF-16BE nebo UTF-16LE.
- **BLOB** – určen pro nespecifikovaná binární data, která jsou uložena tak jak byla přijata na vstupu.

Pojem datová třída je obecnější, než datový typ. Třída `INTEGER` obsahuje šest datových typů `integer` s odlišnou délkou. To znaméná rozdílnou velikost dat na disku, ale při zpracování jsou data převedena do obecnějšího typu (8bajtový `integer`). Pro většinu datových typů je však datová třída nerozlišitelná od datového typu, a proto mohou být tyto výrazy zaměnitelné.

Kromě sloupce `INTEGER PRIMARY KEY` mohu být ve sloupcích uloženy data jakékoliv datové třídy [15].

Datový typ boolean

Databáze neobsahuje speciální datovou třídu pro typ `boolean`, hodnoty jsou uloženy jako celá čísla 0 (false) a 1 (true) [15].

Datum a čas

SQLite neobsahuje sadu tříd pro uložení data a času. Je však obsažena sada funkcí, které umožňují ukládat data a čas do datových typů `TEXT`, `REAL` a `INTEGER` [15].

- **TEXT** – dle standardu ISO8601 ("YYYY-MM-DD HH:MM:SS.SSS").
- **REAL** – určeno počtem dnů od poledne 24. listopadu roku 4714 př. n. l.
- **INTEGER** – čas v systémech UNIX (počet sekund od 1. 1. 1970 00:00:00).

4.1.5 Typová slučitelnost

Pro zajištění co nejvyšší kompatibility mezi SQLite a jinými databázemi podporuje SQLite tzv. typovou slučitelnost na sloupcích. Jedná se o doporučení pro datový typ ukládaných hodnot ve sloupci.

Datová slučitelnost funguje pouze jako doporučení, každý sloupec může stále obsahovat jakýkoliv typ dat. Pokud však bude mít sloupec na výběr z více datových typů, bude jeden preferován před ostatními. V tabulce Tab. 4.1 je uvedena část typové sloučitelnosti datových typů v SQLite. Argumenty určující maximální délky (např. `VARCHAR(255)`) jsou v SQLite ignorovány, jelikož nejsou kladeny žádné délkové omezení na délku řetězců či numerických hodnot (kromě globální hodnoty limitu `SQLITE_MAX_LENGTH`) [15].

Tab. 4.1: Ukázka slučitelnosti datových typů v SQLite [15].

Vstupní datový typ		Výsledek sloučení
INT TINYINT	INTEGER SMALLINT	INTEGER
VARCHAR(255) TEXT	CHARACTER(20) NVARCHAR(100)	TEXT
BLOB		BLOB
REAL DOUBLE	FLOAT	REAL
NUMERIC DECIMAL(10,5)	DATE BOOLEAN	NUMERIC

4.1.6 Nasazení ve vícevláknových aplikacích

SQLite umožňuje několika uživatelům najednou čtení z databáze, avšak pouze jeden může do databáze zapisovat. Během zapisování je databáze uzamčená, toto uzamčení trvá jen pár milisekund, ale při velkém počtu vláken mohou vznikat problémy. SQLite proto obsahuje tři módy, ve kterých může databáze pracovat.

- **Single-thread** – v tomto režimu jsou všechny mutexy[1] zakázány a není bezpečné používat databázi ve více než jednom vlákně.
- **Multi-thread** – tento režim umožňuje práci ve více vláknech avšak žádné připojení k databázi (`Connection`) nemůže být použito současně ve více vláknech.
- **Serialized** – při použití tohoto režimu může být databáze používána ve více vláknech bez omezení.

Mód databáze může být zvolen v čase kompilace, v čase spuštění, nebo při vytvoření nového připojení do databáze (`Connection`) [15].

[1] Mutexy jsou uzamykatelné objekty, navržené pro signalizaci vstupu vlákna do kritické sekce programu, zabraňující ostatním vláknům souběžně přistupovat do stejného paměťového prostoru.

5 OSGI FRAMEWORK

OSGi aliance definuje seznam doporučení pro dynamický modulární systém v jazyce Java. To umožňuje lepší kontrolu struktury kódu, dynamické řízení životního cyklu balíčků a menší svzázanost kódu mezi balíčky.

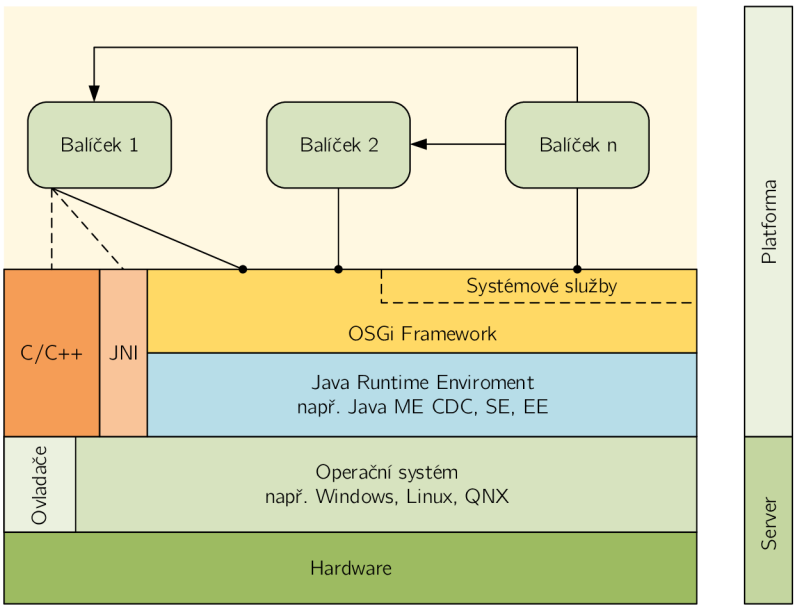
OSGi specifikuje dva dokumenty, jeden s klíčovými službami (Core) a druhý s rozšiřujícími službami (Compendium). Specifikace OSGi jsou definovány také pro podnikové (Enterprise) sektory, které definují některé dodatečné služby [25].

5.1 Architektura

Jak lze vidět na Obr. 5.1, OSGi framework běží nad virtuálním Java strojem (JVM). To znamená že pro svůj běh vyžaduje Java Runtime Enviroment, který je dnes dostupný pro velké množství architektur. Neměl by tedy nastat problém s přenosem programů mezi zařízeními. Jelikož OSGi aliance definuje přesné požadavky na OSGi frameworky, je možné přecházet i mezi jednotlivými frameworky bez jakýchkoliv úprav ve zdrojovém kódu. Jedinou podmínkou je, aby frameworky implementovaly OSGi standard ve stejné revizi, kvůli možné odlišnosti služeb mezi revizemi.

OSGi framework se skládá z vrstvy systémových služeb a vrstvy modulů. Vrstva služeb je přímo definována standardem OSGi, který popisuje požadavky na implementaci služeb.

Vrstva modůlu je již definována uživatelem a obsahuje uživatelské balíčky, které se v terminologii OSGi nazývají bundle [1], [13].

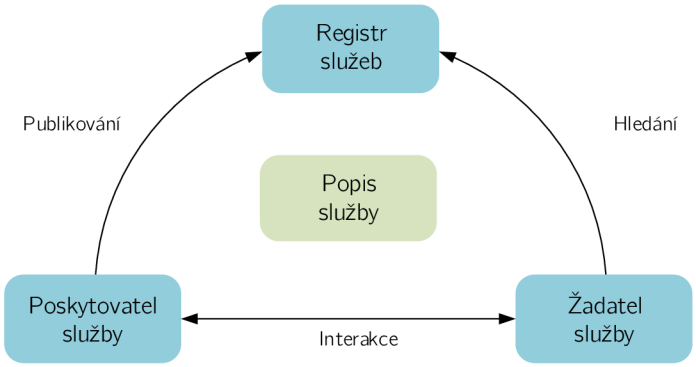


Obr. 5.1: Architektura OSGi.[29]

5.1.1 Vrstva služeb

Tato vrstva umožňuje balíčkům nabízet a hledat poskytované služby. Poskytovatel registruje své služby do registru služeb, zatímco klienti hledají v registrech dostupné služby k použití, jak lze vidět na Obr. 5.2.

Vrstva služeb nabízí vývojový model na základě rozhraní `Interface`, to umožňuje oddělit rozhraní služby od jeho implementace. Služby v OSGi frameworku jsou tedy pouze Java rozhraní, které reprezentují abstraktní kontrakt mezi posktovatelem služby a jejím klientem. To činí vrstvu služeb velmi jednoduchou, protože poskytovatelé služeb jsou pouze Java objekty přímo přístupné přes metody rozhraní [1].



Obr. 5.2: Princip vrstvy služeb [1].

5.2 OSGi bundle

Základní jednotka se v OSGi nazývá balíček tzv. `bundle`, který je tvořen programem v jazyce Java rozšířeném o soubor metadat `manifest.mf`. Příklad jednoduchého manifest souboru lze vidět na List. 5.1. Jediný parametr vyžadovaný OSGi standardem je `Bundle-SymbolicName`, ale je vhodné doplnit soubor o další informace.

Listing 5.1: Jednoduchý manifest soubor.

```
Manifest-Version: 2.0
Bundle-Name: simplebundle
Bundle-SymbolicName: simplebundle
Bundle-Version: 1.0.0
Bundle-Description: Demo Bundle
Bundle-Activator: org.tutorial.simplebundle.Activator
Bundle-Category: example
Import-Package: org.osgi.framework
```

Parametr `Import-Package` slouží pro definici všech balíčků vyžadovaných pro správnou funkci – pokud není balíček ve frameworku obsažen nelze program spustit.

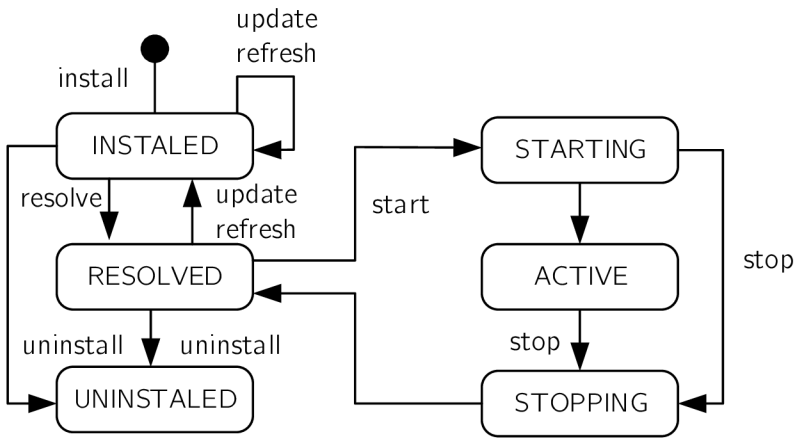
Pokud manifest obsahuje položku `Export-Package`, slouží balíček jako poskytovatel služby a parametr obsahuje název balíčků poskytovaných klientům [13].

5.2.1 Životní cyklus balíčku

OSGi umožňuje dynamicky spustit/zastavit, instalovat/odnistalovat/aktualizovat jednotlivé balíčky bez nutnosti restartovat aplikaci. Pokud není balíček spravován frameworkem, nachází se ve stavu `uninstalled`, po přidání do frameworku je ve stavu `installed`, v této fázi se pokouší o sestavení všech závislostí definovaných manifest souborem. Pokud se podaří tyto závislosti sestavit, změní balíček stav na `resolved`, ve kterém jsou jím exportované balíčky dostupné všem modulům. V případě nesplnění závislostí zůstává balíček ve stavu `installed`.

Ze stavu `resolved` může být modul spuštěn, zavoláním metody `start`. K tomuto účelu obsahuje balíček speciální třídu `Activator`, která umožňuje správné spuštění či zastavení balíčku. Implementace třídy není povinná, ale balíčky bez implementace této třídy se používají pouze jako knihovny funkcí. Při zpracování metody `start` se balíček nachází ve stavu `starting`. Pokud se nevyskytne výjimka v některé z metod, dostává se balíček do stavu `active`. V tomto stavu zůstává, dokud není vyvolána metoda `stop` z aktivátoru.

O spouštění a zastavení modulů se může starat sám framework, může být vyvoláno z konzole nebo je mohou inicializovat také ostatní balíčky (pokud to umožňují bezpečnostní omezení). Přehled všech stavů životního cyklu balíčku lze vidět na Obr. 5.3 [1], [13].



Obr. 5.3: Životní cyklus balíčku [7].

5.3 Knopflerfish framework

Jednou z dostupných implementací OSGi frameworku s otevřeným zdrojovým kódem je Knopflerfish, který ve své poslední verzi splňuje specifikace OSGi R5. Ve volně dostupné verzi obsahuje všechny klíčové (Core) služby a částečně implementuje rozšířené (Compendium) služby. V placené verzi je dostupných více rozšiřujících služeb a také podnikové (Enterprise) služby. Přehled klíčových funkcí implementovaných v Knopflerfish 5 lze nalézt v Tab. 5.1, čísla sekcí odpovídají kapitolám v dokumentu OSGi R5.

Tab. 5.1: Klíčové funkce implementované v Knopflerfish 5 [19].

Sekce	Popis
2. Vrstva zabezpečení	Vrstva zabezpečení OSGi dle zabezpečovacího mechanismu Java2.
3. Vrstva modulů	OSGi řešení modularizace aplikací.
4. Vrstva životního cyklu	Vrstva řídící životní cyklus balíčků.
5. Vrstva služeb	Zajišťuje poskytování, nalezení služeb v registru služeb.
6. Rozhraní prostředků	Rozhraní pro model Požadavek-Schopnost.
7. Rozhraní provázání balíčků	Poskytuje přístup k vnitřnímů sdílení balíčků.
8. Názvosloví	Názvosloví pro model Požadavek-Schopnost.
9. Rozhraní úrovně spuštění	Rozhraní pro řízení pořadí spouštění a zastavení balíčků.
10. Rozhraní frameworku	Umožňuje interakci balíčků s frameworkem.
50. Administrace podmíněného oprávnění	Systém bezpečnostní politiky a oprávnění pro balíčky. Nahrazuje službu Administrace oprávnění.
51. Administrace oprávnění	Systém oprávnění pro balíčky.
52. URL manipulátor	Mechanismus rozšiřující Java run-time o nové URL schémata a manipulátory skrze balíčky.
53. Resolver	Sada mechnismů pro bližší interakci vrstvy modulů řešící požadavky a možnosti balíčků.
54. Balíčky	Mechanismy pro řízení viditelnisti mezi balíčky.
55. Služby	Sada mechanismů pro bližší interakci se službou registrů.
56. Propojení	Mechanismy umožňující bajt kódu propojení s třídami načtenými balíčky.
701. Sledovač	Prostředek pro sledování balíčků a služeb.

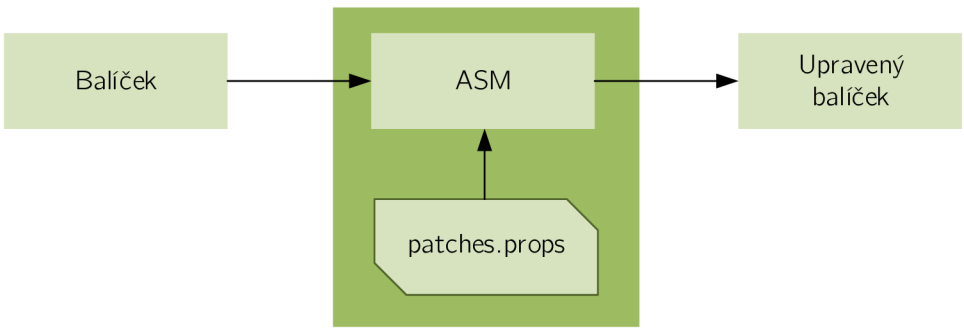
Knopflerfish obsahuje také sadu balíčků, které zejdnodušují práci s frameworkem, jejich seznam lze nalézt v Tab. 5.2 [19].

Tab. 5.2: Balíčky obsažené v distribuci Knopflerfish [19].

Komponenta	Popis
Plocha	Grafický nástroj pro správu balíčků.
Logovací nástroje	Nástroje pro jednoduší logování.
Ostatní nástroje	Třídy pro zápis/čtení souborů.
Konzole	Služba poskytující konzoli umožňuje balíčkům definovat vlastní příkazy.
TTY konzole	Konzole využívající stdin/stdout.
Telnet konzole	Konzole využívající Telnet.
Konzole frameworku	Konzole pro obsluhu jádra Knopflerfish frameworku.
Příkazy logování	Příkazy pro řízení logování.
Konzole CM	Příkazy pro správu konfiguračních dat.
Commons Logging	Balíček Apache Commons Logging pro využití v OSGi.
Sériový port	Balíček pro komunikaci přes sériové rozhraní.
KF test	Sada nástrojů pro testování Knopferfish frameworku založená na JUnit.

5.3.1 Implementace Knopflerfish

K načítání balíčků využívá Knopflerfish ASM bajt-kódovou manipulační knihovnu. Konfigurační soubor specifikuje, které balíčky/třídy mohou být modifikovány, balíčky jsou automaticky modifikované/aktualizované při načítání. Princip implementace v Knopflerfish lze vidět na Obr. 5.4 [29].

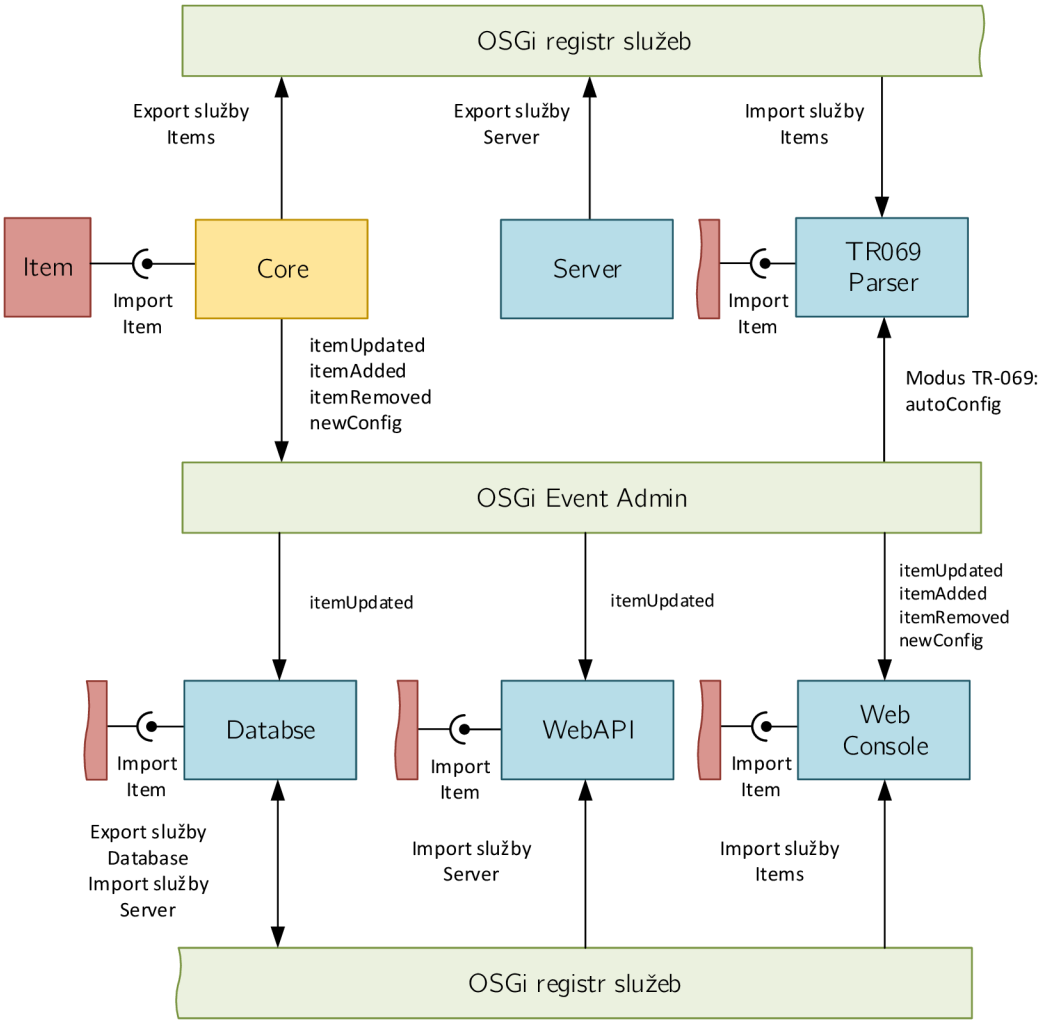


Obr. 5.4: Implementace Knopflerfish.

6 VYTVOŘENÁ APLIKACE

Výsledná aplikace je implementována v jazyce Java s využitím platformy OSGi, která byla vybrána pro svou univerzálnost a schopnost dynamicky spravovat jednotlivé balíčky vytvořené aplikace. Pro běh aplikace je využito frameworku Knopflerfish, který implementuje OSGi doporučení dle revize R5.

Vzdálená konfigurace zařízení skrze protokol TR-069 je realizována s využitím klienta modus TR-069 3.7.3, který byl vyvinut dle doporučení OSGi. Funkci auto-konfiguračního serveru zajišťuje aplikace genieacs 3.6.1. Pro sběr, zpracování, uložení a vizualizaci M2M dat byly vytvořeny balíčky Item 6.1, Core 6.2, TR069 Parser 6.3, Server 6.4, WebConsole 6.8, Database 6.5 a WebAPI 6.6. Dále bylo vytvořeno uživatelské rozhraní 6.7, které je postaveno na technologiích HTML5 (HyperText Markup Language 5) a JavaScript.



Obr. 6.1: Komunikace mezi jádrem a balíčky.

6.1 Balíček Item

Tento balíček slouží jako knihovna bez vlastního aktivátoru, zajišťuje jednotný formát zpráv, kterými mezi sebou balíčky komunikují. Proto musí tento balíček ve svém manifestu importovat všechny balíčky, které zproztředkovávají komunikaci s `Core` balíčkem. Jak vyplývá z Tab 6.1, balíček `Item` obsahuje dvě třídy `Item` a `Value`.

Tab. 6.1: Položky v balíčku Item.

Třída	Položka	Typ	Popis
Item	serial	`String`	Sériové číslo, tato hodnota musí být unikátní pro každou položku.
	vendor	`String`	Jméno výrobce zařízení.
	store	`Boolean`	Určuje zda má být položka ukládána do databáze, výchozí hodnota je `false`.
	values	`HashMap`	Obsahuje seznam hodnot jednotlivých položek. Jako klíč slouží název položky. Hodnoty jsou obsaženy ve třídě `Value`.
Value	value	`String`	Hodnota položky.
	unit	`String`	Jednotka hodnoty.

6.2 Balíček Core

Jako jádro aplikace slouží balíček `Core`, poskytuje všem balíčkům službu `Items`, která slouží jako registr všech dostupných položek. Jako klíč pro adresaci položek slouží jejich sériové číslo, které musí být pro každou položku unikátní. Registr položek využívá objekt `ConcurrentHashMap`, který zaručuje bezpečný přístup k položkám i ve vícevláknových aplikacích. Balíček `Core` musí vždy běžet jako první, v opačném případě není možné spustit ostatní balíčky.

Důvodem této implementace je jednoduché rozšíření služeb bez zásahu do zdrojového kódu `Core` balíčku, pokud je nutné vyvořit další službu stačí aby balíčk importoval službu `Items`.

Jak vyplývá z Obr. 6.1 tento návrh umožňuje jednoduchou komunikaci všech rozšiřujících balíčků s `Core`, ale neumožňuje komunikaci opačným směrem. Proto jádro aplikace využívá pro komunikaci s klienty systémovou službu `OSGi Event Admin`.

6.2.1 Komunikace služby s klienty

Jak bylo uvedno v předchozí sekci 6.2, pokud potřebuje `Core` balíček komunikovat s klienty využívá k tomu služby `OSGi Event Admin`. V tomto případě slouží jádro jako zdroj událostí `Events`, kterým rozšiřující balíčky naslouchají. Jednotlivé události jsou od sebe odlišeny pomocí položky `topic`, události balíčku `Core` mají prefix `symphony/event/`, za posledním lomítkem je uveden druh události, seznam generovaných událostí je uveden v Tab. 6.2. Událost také definuje datovou část `property`, která obsahuje jeden nebo více objektů typu `Item`, kterých se událost týká.

Pro zpracování událostí využívají klienti službu `EventHandler`. Pomocí pole `topic` lze u klientů definovat, kterým událostem budou naslouchat, ostatní události jsou ignorovány.

Tab. 6.2: Události generované jádrem aplikace.

Událost	Popis
`itemUpdated`	Informuje klienty o změně hodnot položky.
`itemAdded`	Událost informující o nové položce.
`itemRemoved`	Informace o odstranění položky z registru.
`newConfig`	Oznámení o načtení nové konfigurace.

6.3 Balíček TR069 Parser

Tento balíček slouží pro načtení počáteční konfigurace, ale také pro zpracování nové konfigurace přijaté protokolem TR-069.

Z důvodu spolupráce s firmou Telekom Austria [31] je pro získání konfigurace použito metody `Download`, která slouží ke stažení nové konfigurace z ACS. Ve firmě Telekom Austria je využíváno toto řešení a proto bylo aplikováno i v této práci.

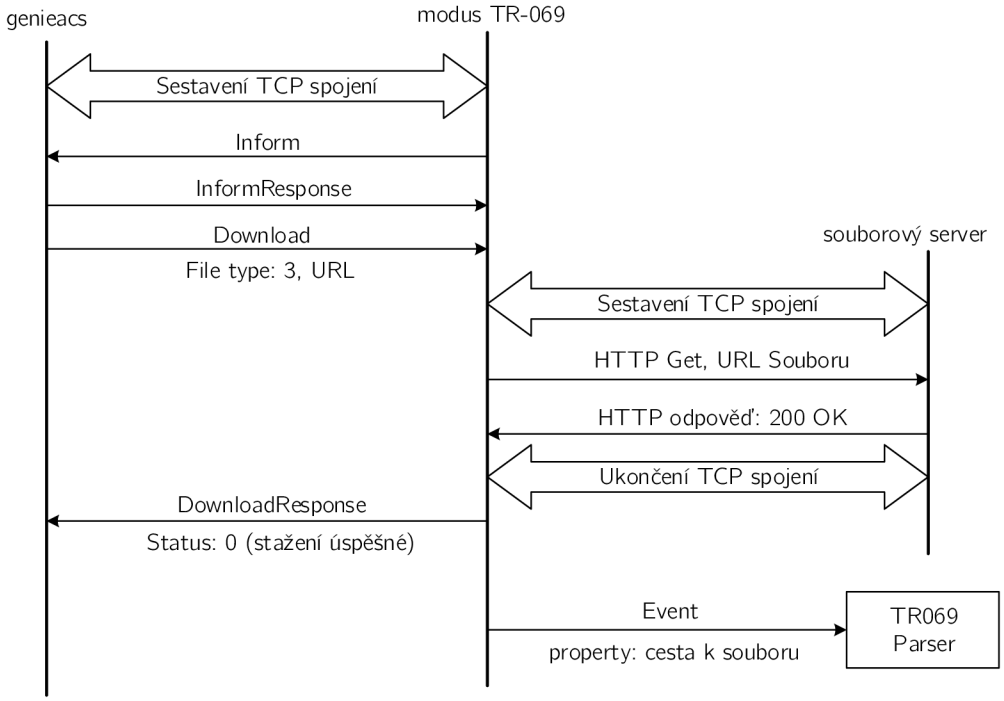
6.3.1 Získání konfiguračního souboru

Jak lze vidět na Obr 6.2. je protokol TR-069 využit pouze k informaci o novém konfiguračním souboru, dle názvolsloví TR-069 `Configuration Files`, v poli `FileType` označen číslem 3. Ke stažení souboru je využito protokolu HTTP a nová konfigurace je zpracována balíčkem `TR069 parser`.

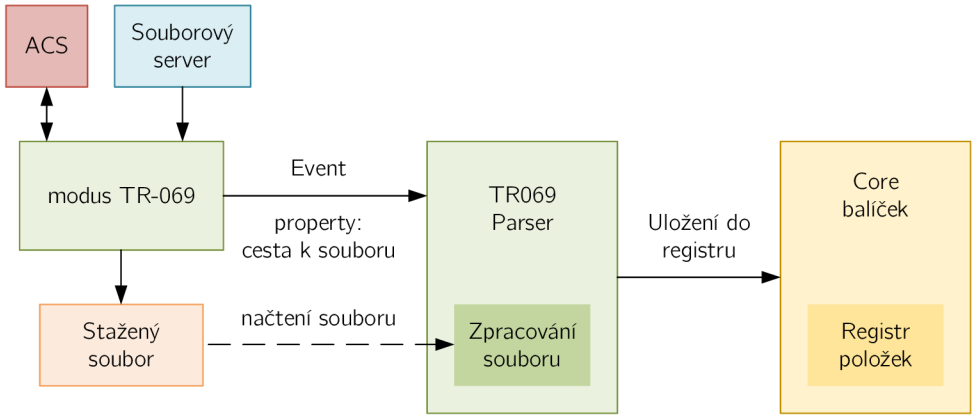
Jelikož klient modus TR-069 nebyl vyvíjen pro přímou spolupráci s dalšími systémy, bylo nutné jej upravit, aby tuto spolupráci umožňoval. TR-069 klient nyní po úspěšném stažení nové konfigurace vytvoří událost (`Event`), která obsahuje cestu k právě staženému souboru s konfigurací. Této události naslouchá balíček

`TR06 Parser`, který stažený soubor zpracuje a do systému nahraje novou konfiguraci. Celý proces je uveden na Obr. 6.3.

Protokol TR-069 definuje také metodu `Upload`, která umožňuje na vyžádání ACS odeslat soubory z CPE na server. Tato metoda však není povinná a proto její implementace není výrazněji rozšířena. Použitý kilent modus-TR069 metodu podporuje, avšak server genieacs ne.



Obr. 6.2: Získání konfiguračního souboru protokolem TR-069.



Obr. 6.3: Aplikace vzdálené konfigurace.

6.4 Balíček Server

Výsledná aplikace využívá více než jeden servlet sloužící pro přístup k aplikaci skrze webové rozhraní. Není proto vhodné s každým balíčkem implementující servlet vytvářet novou instanci serveru. Z tohoto důvodu byl vytvořen balíček `Server`, který implementuje kontejner webového serveru Jetty. Server Jetty byl vybrán z důvodu podpory technologie WebSocket a plné kompatibility s OSGi standardem.

Balíček `Server` tedy umožňuje přidávání a odebírání servletů, přímo za běhu aplikace. Jako vstupy rozhraní slouží instance samotného servletu a cesta na které bude servlet dostupný. Balíček také implementuje souborový servlet, skrze který je možné přistupovat ke složkám na serveru. Tento souborový servlet je určen zejména pro přístup k uživatelskému rozhraní a je dostupný na adrese `adresa_serveru:port/gui`.

6.5 Balíček Database

Jak bylo zmíněno v kapitole 4, je v této práci využito databáze SQLite. Vytvořená databáze umožňuje uložení dat po dobu dvou let a obsahuje mechanismy pro agregaci dat. Poskytuje také servlet, který umožňuje přístup k datům skrze protokol HTTP.

6.5.1 Řadič databáze

Aby bylo možné SQLite databázi v jazyce Java využívat, je nutné do aplikace importovat databázový řadič `sqlite-jdbc driver`. Ten slouží jako rozhraní mezi databází napsanou v jazyce `C` a aplikací v jazyce Java. Velkou výhodou tohoto databázového řadiče je nutnost nulové konfigurace před použitím. V základní distribuci jsou obsaženy také všechny knihovny potřebné pro běh na nejpoužívanějších architekturách (x86, x64, ARM) i operačních systémech (Windows, MacOS, Linux, UNIX). Jelikož má aplikace otevřený zdrojový kód, je možné zkompilovat jej i pro jiné architektury či operační systémy.

Od verze 3.8.11.2 obsahuje řadič také soubor `manifes` se všemi potřebnými údaji pro OSGi Class Loader a je proto možné jej využít bez jakýchkoliv úprav jako OSGi balíček [20].

6.5.2 Komunikace s balíčky

Balíček databáze, jako všechny ostatní, musí importovat balíček `Item`, aby byla zajištěna správná struktura dat. Přístup k datům pro ostatní balíčky je zajištěn službou poskytovanou databází. Jednou z možností vstupu dat je služba `Event Admin`, databáze naslouchá jako `Event Listener` událostem typu `itemUpdated`, ze kterých

získá objekty typu `Item`. Z tohoto objektu lze pak získat všechna potřebná data pro vytvoření záznamu v databázi. Je však také možné využít přímo poskytovanou databázovou službu, toto řešení však zvyšuje provázanost mezi balíčky a jde tak proti filozofii OSGi. Při vyšším provázání balíčů se ztrácí možnost dynamické správy služeb bez vzájemného ovlivnění.

6.5.3 Struktura dat

Data jsou v databázi ukládána do tabulek dle typu zařízení definovyných v objektu `Item` položkou `type` např. elektroměry jsou ukládány do tabulky `electricityMeter`. Všechny položky typu `electricityMeter` jsou tedy ukládány do této tabulky. Pokud tabulka odpovídající typu zařízení není v databázi obsažena, je vytvořena.

Struktura řádků v každé tabulce je stejná a skládá se ze šesti položek, jak lze vidět v Tab. 6.3.

Tab. 6.3: Struktura dat v databázi.

Název	Datový typ	Popis
serial	TEXT	Sériové číslo položky.
type	TEXT	Typ uložené hodnoty, např. teplota.
timestamp	NUMBER	Časová značka ve formátu UNIX.
perid	TEXT	Časová perioda dat.
value	TEXT	Hodnota záznamu.
unit	TEXT	Jednotka záznamu.

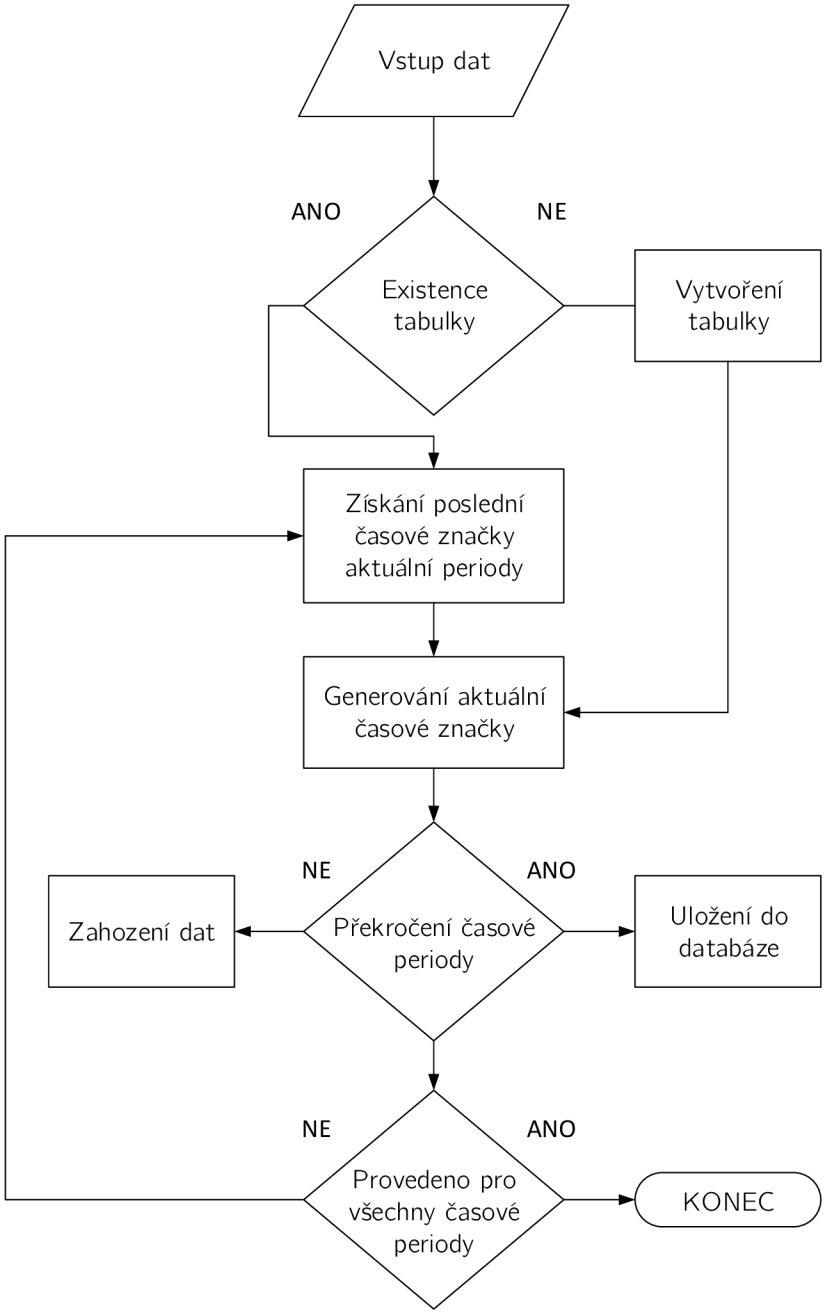
6.5.4 Agregační mechanismus

Balíček databáze obsahuje agregační mechanismy, které se starají o řízení granularitu ukládaných dat. Jak lze vidět v Tab. 6.4, databáze obsahuje tři různé hodnoty granurality pro různé časové periody.

Tab. 6.4: Granularita dat v databázi

Označení	Popis	Rozlišení
2D	Dva dny.	1 minuta
1Y	Jeden rok.	1 hodina.
2Y	Dva roky.	1 den.

Při přijetí dat databáze nejprve zkontroluje, zda je položka již v databázi. Pokud v databázi obsažena není, je přímo uložena. Jestli se v databázi položka vyskytuje, je zkontrolována její poslední časová značka. Pokud je překročen časový limit určený pro danou časovou periodu, je hodnota uložena do databáze. Pokud časový limit překročen není, hodnota se do databáze neuloží a je zahozena. Celý proces, který lze vidět na Obr. 6.4, se opakuje pro každou časovou periodu.



Obr. 6.4: Agregační mechanismus databáze.

6.5.5 Databázový servlet

Balíčky využívající službu přímo poskytovanou balíčkem databáze mohou k datům v databázi přistupovat přímo skrze sdílené rozhraní. Avšak aplikace pracující mimo OSGi framework se k datům dostat nemohou, z tohoto důvodu byl vytvořen databázový servlet poskytující přístup k datům i externím aplikacím. Servlet je dostupný na adrese `adresa_serveru:port/database`, parametry dat navrácených z databáze lez ovlivnit pomocí přepínačů. Seznam přepínačů a dostupné hodnoty jsou uvedeny v Tab. 6.5. Získaná data jsou zapsána ve formátu JSON, který je vhodný pro strojové zpracování dat. Struktura získaných dat je uvedena v List 6.1.

Tab. 6.5: Dostupné přepínače.

Označení	Popis	Dotupné hodnoty
serial	Sériové číslo.	–
type	Typ položky, odpovídá názvu tabulky.	–
value	Požadovaná hodnota.	–
period	Perioda vrácených dat.	5m, 1h, 1D, 2D, 1W, 2W, 1M, 1Y, 2Y
step	Velikost kroku vrácených dat.	–

Listing 6.1: Struktura dat získaných ze servletu.

```
"items":[
  {"timeStamp": 1458921900, "value": "25.4"},
  {"timeStamp": 1458921960, "value": "25.2"}]
```

6.6 Balíček WebAPI

Při tvorbě uživatelského rozhraní je nutné, aby existovala možnost komunikovat s OSGi aplikací skrze webovou stránku či jinou uživatelsky přívětivou cestu. Pro tento účel byl vytvořen balíček `WebAPI`. Jako každý balíček vytvořené aplikace poskytující servlet, musí `WebAPI` bundle využívat službu poskytovanou balíčkem `Server`. Jelikož `WebAPI` komunikuje i sbalíčkem `Core` je nutné, aby importoval i službu s `Items` poskytovanou balíčkem `Core`. Pro zajištění jednotného formátu objektů využívaných pro komukaci mezi balíčky je nutné importovat knihovnu položek poskytovanou balíčkem `Item`.

Komunikace s uživatelským rozhraním je poté realizována s využitím protokolu WebSocket, který slouží jako technologie pro přenos zpráv ve formátu JSON.

Balíček `WebAPI` se nestará pouze o přenos dat o aktuálním stavu položek, ale také o načtení počáteční konfigurace pro uživatelské rozhraní. V tomto konfiguračním souboru jsou obsaženy informace pro vykreslení uživatelského rozhraní např. typ položky, název položky a umístění do skupiny.

6.6.1 Struktura konfiguračního souboru

Jak bylo zmíněno v předchozím textu, slouží balíček `WebAPI` i pro přenost počáteční konfigurace z níž je vygenerováno uživatelské rozhraní. Tento konfigurační soubor je umístěn v kořenové složce aplikace a je pojmenován `config.json`. Jak vyplývá již z názvu, jeho vnitřní struktura je typu JSON. Tento formát byl vybrán zejména pro snadné zpracování pomocí jazyka Java ale i JavaScript, který je použit na straně uživatelského rozhraní. Příklad struktury konfiguračního souboru je uveden v List 6.2.

Listing 6.2: Struktura konfiguračního souboru.

```
{"pages":[
    {
     "name":"Kitchen",
     "widgets":[
        { "id":"1234", "label":"Warm", "type":"label", "item":"warm",
          "state":"-", "units":"-"
        },
        { "id":"34", "label":"Light kitchen", "type":"switch", "item":"lihght1",
          "state":"-", "units":"-"
        }]
    },
    {
     "name":"Living Room",
     "widgets":[
        { "id":"26", "label":"RGB Light", "type":"rgb", "item":"rgb_1",
           "state":"-", "units":"-"
        },
        { "id":"124", "label":"Dimm Light", "type":"slider", "item":"lihght_1",
           "state":"-", "units":"-"
        }]
     }]}
```

Pole page

Jak je patrné z ukázkového souboru, soubor je tvořen polem `pages`. Toto pole slouží jako rozdělení v nejvyšší sekci, každá položka `page` slouží jako samostatná stránka v uživatelském rozhraní. Objekt `page` obsahuje hodnotu `name`, která slouží jako jméno stránky zobrazené v navigačním menu rozhraní. Jednotlivé položky zobrazené na stránce jsou uloženy v poli `widgets`.

Pole widgets

Toto pole je v hierarchii souboru o řád níže než pole `page` a obsahuje položky vykreslené na jedné stránce. Každý prvek pole `widgets` obsahuje šest povinných položek, které jsou uvedeny v Tab. 6.6.

Tab. 6.6: Povinné položky pole widgets.

Název	Popis
id	Sériové číslo položky.
label	Titulkek položky zobrazený v uživatelském rozhraní.
type	Typ elementu generovaný v uživatelském rozhraní.
item	Jméno položky, v objektu `Item`.
state	Aktuální hodnota položky.
units	Jednotka položky.

Položka type

Jak je uvedeno v Tab. 6.6, položka `type` definuje typ elementu, který je vygenerován v uživatelském rozhraní. Pro použití v uživatelském rozhraní je podporováno šest typů položek, které jsou uvedeny v Tab. 6.7.

6.6.2 Zpracování konfiguračního souboru

Jelikož je již při přenosu souboru přenášena i aktuální hodnota položky tak aby, již při generování uživatelského rohraní byly dostupné aktuální hodnoty všech položek. Je nutné před samotným odesláním konfiguračního souboru provést jeho zpracování. Jelikož je položka `state` přímo závislá na hodnotě v registru, není možné aby její hodnota byla součástí souboru. Proto je před samotným přenosem konfigurační soubor prohledán, a ke každé položce v poli `widgets` nalezena aktuální hodnota. Pro získání hodnoty z registru položek je využito hodnot z pole `id` (sériové číslo) a `item` (název hodnoty v objektu `Item`). Poté je soubor již odeslán k WebSocket klientovi.

Tab. 6.7: Dostupné typy položek v uživatelském rozhraní.

Název	Popis
label	Slouží pro zobrazení aktuální hodnoty položky.
switch	Element slouží jako klasický vypínač se stavy `On/Off`.
updown	Element s tlačítky `Up|Down` a nastavenou hodnotou, lze použít např. pro nastavení teploty termotstatu.
rgb	Element se třemi posuvníky sloužícími pro nastavení RGB diod.
slider	Element s jedním posuvníkem sloužící pro nastavení hodnoty v rozsahu 0–100 např. u stmívačů.
chart	Objekt vykreslující grafy.

6.6.3 Přenos konfigurace

Konfigurační soubor je přenášen vždy při vytvoření nového WebSocket spojení. Každé nové spojení však vytvoří nový servlet a je proto nutné mít seznam všech dostupných servletů. Pro tento účel je vytvořena třída `WSHandler`, která obsahuje objekt `ArrayList` v kterém jsou uloženy všechny WebSocket servlety. Při vytvoření nového spojení je právě vytvořený servlet přidán do seznamu a ke klientovi náležícímu k servletu je odeslán konfigurační soubor.

Jelikož je při ukončení spojení zničen i samotný servlet, je nutné jej odstaranit ze seznamu. V opačném případě by v seznamu zůstávali i dávno neexistující položky.

6.6.4 Přenos hodnot a příkazů

Při změně stavu položky vyvolá registr položek událost typu `itemUpdated`. Této události nasluchá balíček `WebAPI` a ze získaných dat vytvoří data pro přenos k WebSocket klientům. Struktura dat je shodná s položkami v poli `widgets`, je však přenesen pouze jeden JSON objekt.

Při přenosu příkazů z uživatelského rozhraní je využito stejné struktury dat jako v předchozím případě. Aktualizované hodnoty jsou poté uloženy do registru položek, který zajistí, aby se změny projevily v celém systému balíčků.

6.7 Uživatelské rozhraní

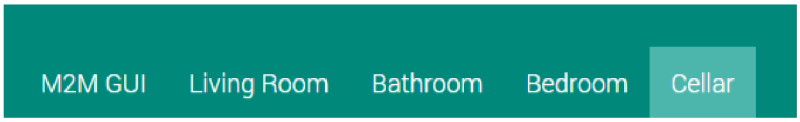
Pro komfortnější ovládání aplikace bylo vytvořeno uživatelské rozhraní dostupné na `http://adresa_serveru:8080/gui`. Ukázky rozhraní vygenerovaného z přiloženého konfiguračního souboru lze nalézt v příloze A. Jelikož je aplikace navržena

pro nasazení v domácích bránách založených na méně výkonném hardware, bylo pro generování uživatelského rozhrání použito technologie HTML5 a JavaScript. Ten je, na rozdíl od PHP či Java, zpracováván až v koncovém zařízení a není tedy zatěžován procesor domácí brány. Ta však musí poskytovat data nutná pro správné fungování uživatelského rozhrají. Pro tento účel je vytvořen komunikační kanál využívající protokol WebSocket. Tento protokol pracuje na principu klient-server. V tomto případě funguje OSGi aplikace jako WebSocket server (je poskytován balíčkem `WebAPI`) a klientem je uživatelské rozhraní. WebSocket je standardní součástí jazyka HTML 5, na straně klienta není nutné instalovat žádné rozšiřující knihovny.

6.7.1 Generování uživatelského rozhraní

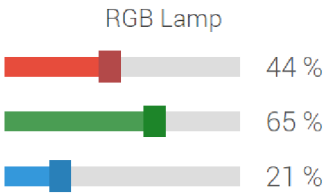
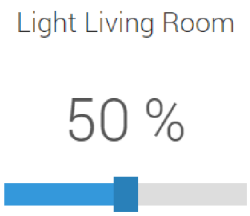
Struktura konfiguračního souboru uživatelského rozhraní byla popsána v kapitole 6.6. Jak lze vidět na Obr. 6.6, je po úspěšném připojení k serveru přenesen konfigurační soubor, který prochází dalším zpracováním.

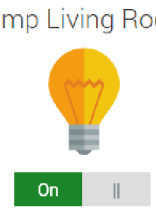
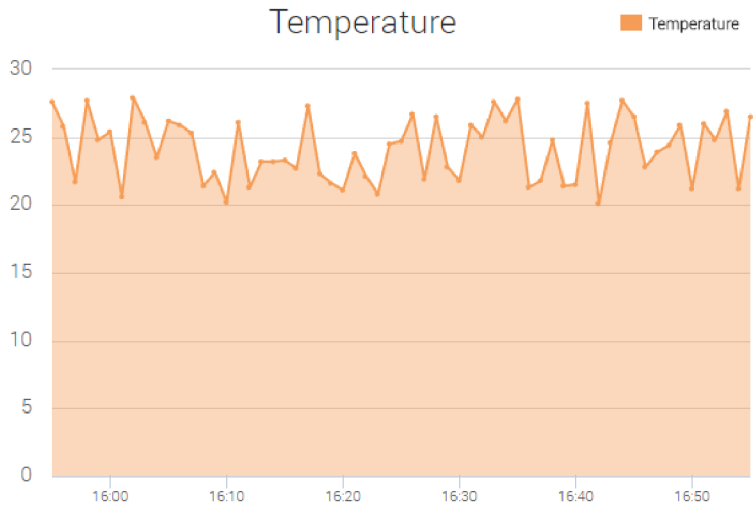
Pole `pages` slouží pro logické dělení do sekcí hlavní nabídky, jak lze vidět na Obr. 6.5. Do těchto sekcí jsou poté vkládány jednotlivé položky z pole `widgets`. Vzhled a funkcionalita položky je určena hodnotou v poli `type`, v Tab. 6.8 lze nalézt seznam všech dostupných položek. Výsledkem průchodu konfiguračním souborem je vygenerované rozhraní. Jelikož konfigurační soubor obsahuje i stavy položek, rozhraní již poskytuje aktuální hodnoty dat. Pro pozdější aktualizaci hodnot, je každá položka označena unikátním `id`. To je složeno z hodnot konfiguračního souboru kombinací položek `id` a `value`. Tím je docíleno, že lze položku se stejným sériovým číslem s více hodnotami zobrazit v uživatelském rozhraní v oddělených elementech. V tomto bodě je již celé grafické rozhraní vygenerováno, avšak v elementu `chart` nejsou dostupné hodnoty. Ty jsou získány z databázového servletu.

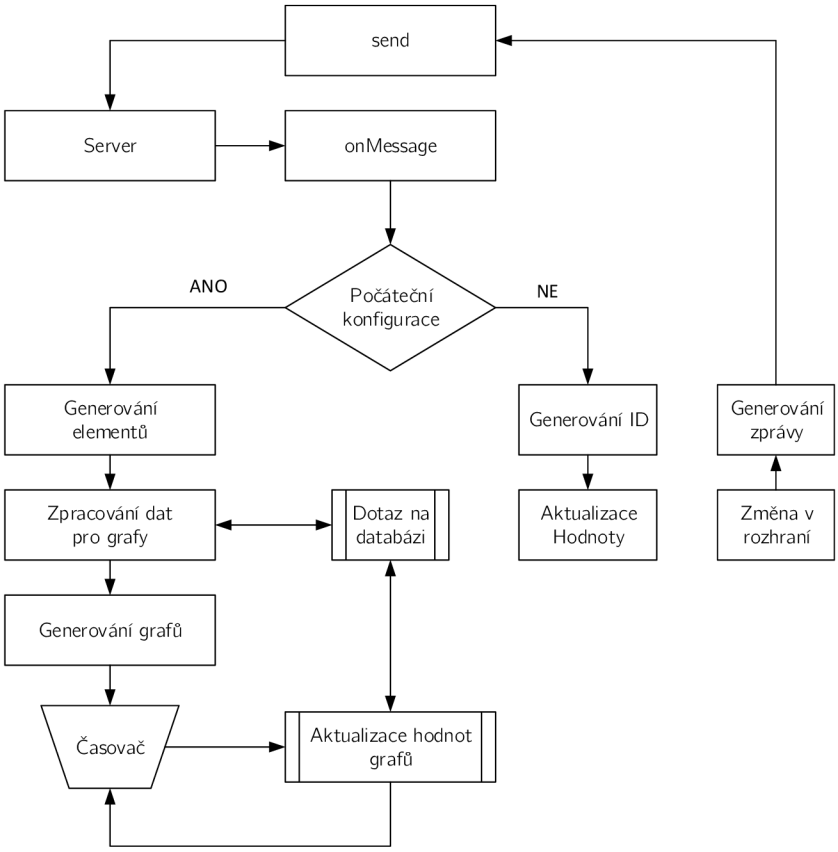


Obr. 6.5: Hlavní nabídka uživatelského rozhraní.

Tab. 6.8: Dostupné typy položek v uživatelském rozhraní.

Název	Element	Název	Element
`label`		`rgb`	
`updown`		`slider`	

Název	Element
`switch`	
`chart`	



Obr. 6.6: Blokový diagram funkce uživatelského rozhraní.

Generování grafů

Pro zobrazení grafů je využito frameworku Highcharts [14]. Ten vykresluje data do elementu `div`, který je určen unikátním ID, viz předchozí text. Hodnoty, které jsou v grafu vykresleny mají strukturu pole. Nejprve je však nutné získat hodnoty z databázového servletu. K tomuto účelu je využito AJAX (Asynchronous JavaScript and XML) dotazů, které navrací data v JSON struktuře. Tyto data však nelze přímo zobrazit, je nutný převod na objekt typu pole. Hodnoty v této struktuře lze již přímo zobrazit.

Tímto způsobem jsou data zobrazena pouze na počátku. Při aktualizaci hodnot by bylo zpracování celého pole hodnot znovu zbytečné. Proto jsou z databázového servletu získány pouze poslední hodnoty. Data jsou pak již pouze přidána do grafu pomocí funkce `addPoint`, která je součástí frameworku. Ta kromě přidání nového bodu automaticky odstraní i data která leží mimo zobrazovanou periodu, tedy data, která jsou zastaralá.

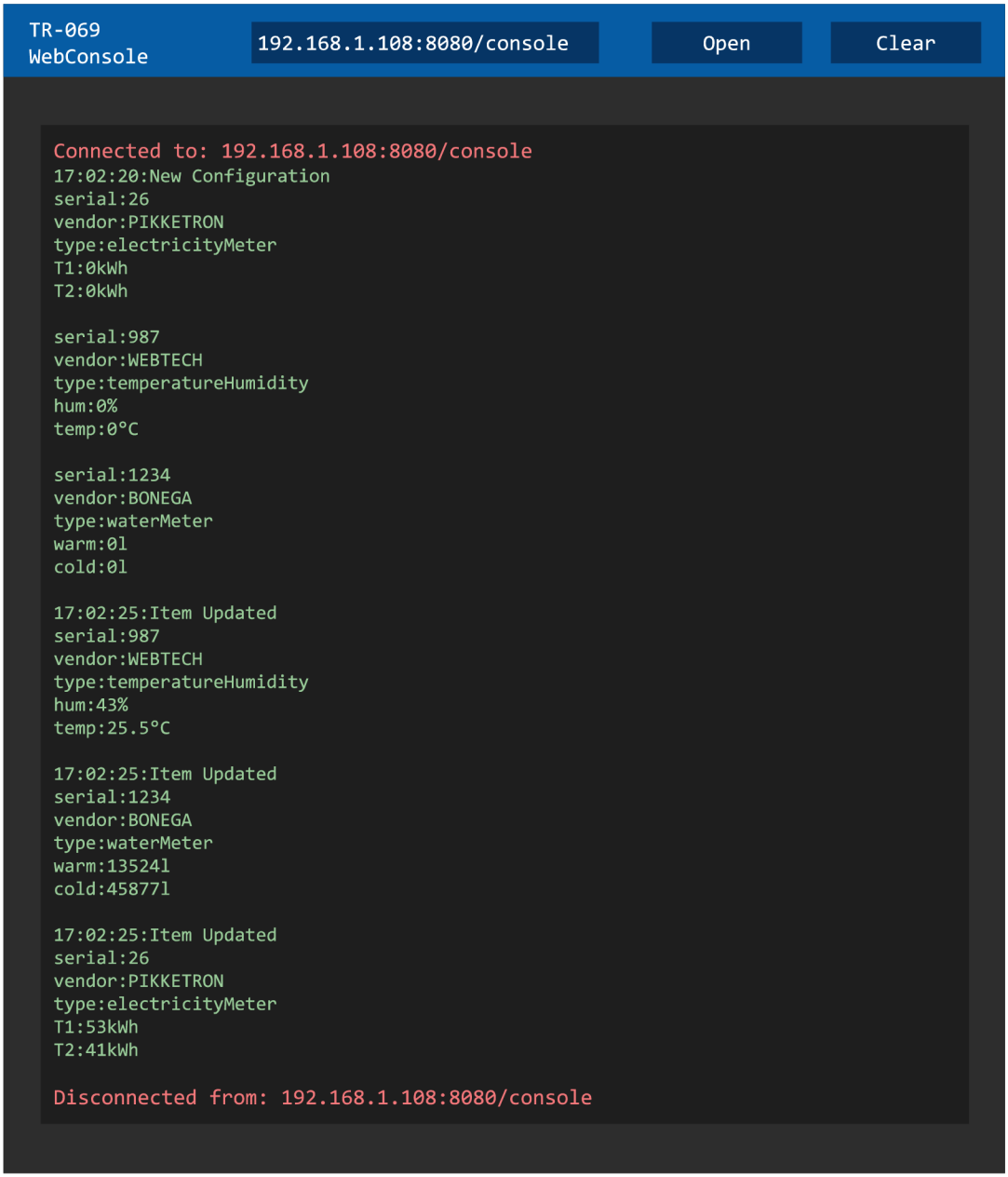
6.7.2 Aktualizace hodnot

Jak bylo popsáno v kapitole 6.6, jsou při změně hodnot položek odeslána data ve formátu JSON. V uživatelském rozhraní je z položek `id` a `value` vytvořen identifikátor, pomocí nějž lze dohledat správný prvek a aktualizovat jeho hodnotu. V opačném případě, kdy je zdrojem změny uživatelské rozhraní, je proveden opačný proces a data jsou odeslána na server. Ten již zajistí aby byla data přeposlána do všech instancí uživatelských rozhraní.

6.8 Balíček WebConsole

V některý případech není možné zobrazit výpis události do systémové konzole např. pokud OSGi framework běží jako démon na pozadí systému. Z tohoto důvodu byl vytvořen balíček `WebConsole`, který funguje jako webová služba a do konzole vypisuje jednotlivé události (`Events`), jak lze vidět na Obr. 6.7.

Webová služba konzole využívá pro komunikaci s balíčkem `WebConsole` technologii WebSocket, která je v základní součístí HTML 5. Balíček `WebConsole` slouží jako `EventHandler`, který naslouchá všem událostem s prefixem `symphony/events`. Tyto události jsou poté zpracovány a data odeslána do webové služby skrze WebSocket. To zaručuje téměř okamžitou odezvu bez zpoždění, která je patrná u služeb využívající HTTP pooling.



Obr. 6.7: Webová konzole.

7 ZÁVĚR

Tato diplomová práce přímo navazuje na semestrální projekt, jehož hlavním cílem bylo vytvoření aplikace pro vzdálenou správu zařízení s využitím protokolu TR-069. V této práci byly doplněny zbylé body zadání, zejména vytvoření databáze a uživatelského rozhraní.

Vytvořená databáze je založena na projektu SQLite a dokáže uchovat naměřená data až po dobu dvou let. Databáze obsahuje také agregační mechanismus, jelikož kapacita úložiště je na embedded zařízeních značně omezena. Výhodou pro poskytovatele služeb, je že databáze nevyžaduje téměř žádnou interakci s uživatelem. Všechna data jsou obsažena v jednom souboru a není potřeba žádná dodatečná konfigurace či instalace knihoven.

Uživatelsky přívětivé ovládání aplikace zajišťuje uživatelské rozhraní. To není založeno na technologii OSGi, jelikož systémy založené na generování rozhraní na straně serveru jej značně zatěžují. Pro domácí bránu je tento problém ještě výraznější, jelikož výkon těchto zařízení je mnohokrát nižší než je tomu v porovnání s klasickými stolními počítači/notebooky. Z tohoto důvodu je rozhraní generováno na straně klienta s využitím jazyka JavaScript. Rozhraní umožňuje zobrazení základních elementů pro ovládání zařízení či zobrazení naměřených hodnot. Pro porovnání s dříve získanými hodnotami umožňuje rozhraní vykreslení grafů. Pro tento účel je využito frameworku Highcharts, který umožňuje, na rozdíl od od ostatních řešení, použití bez připojení k Internetu. S ohledem na moderní trend chytrých zařízení s malým displejem a velkou hustotou bodů je rozhraní plně responzivní. To umožňuje pohodlné zobrazení i na mobilních telefonech.

Pro přístup k aktuálním hodnotám, datům v databázi nebo ovládání zařízení lze kromě uživatelského rozhraní využít také programátorské rozhraní. Je tedy možne vytvoření vlastního uživatelského rozhraní. Programátorská rozhraní pro ovládání aplikace a přístup do databáze jsou oddělena a mohou běžet nezávisle na sobě.

Z hlediska dalšího rozšíření práce je možná implementace agregačních mechanismů. Ty byly popsány v kapitole 2 teoreticky popsány avšak nejsou implementovány. Z dostupných algoritmů se jako nejlepší jeví mechanismus CSDA. Ten umožňuje rozsáhlé nastavení pravidel na základě XML souborů.

LITERATURA

[1] ALVES, Alexandre de Castro. *OSGi in Depth*. Shelter Island: Manning Publications Co., 2012. ISBN 9781935182177.

[2] ANISETTI, Marco, Claudio A. ARDAGNA a Ernesto DAMIANI. Fine-Grained Modeling of Web Services for Test-Based Security Certification. *2011 IEEE International Conference on Services Computing*. IEEE, 2011, s. 456-463. DOI: 10.1109/SCC.2011.27. Dostupné z: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6009294`

[3] BOSWARTHICK, David, Omar ELLOUMI a Olivier HERSENT. *M2M communications: a systems approach*. Hoboken, N.J.: Wiley, 2012, xxiii, 308 p. ISBN 978-1-119-99475-6.

[4] BERNSTEIN, Jeff a Tim SPETS. DSLHOME-TECHNICAL WORKING GROUP. *CPE WAN Management Protocol*. 2004. Dostupné z: `https://www.broadband-forum.org/technical/download/TR-069.pdf`

[5] BROADBAND FORUM TR-069 STANDARDS SUPPORT. INCOGNITO. *Incognito* [online]. 2015 [cit. 2015-11-10]. Dostupné z: `https://www.incognito.com/products/auto-configuration-server/standards/`

[6] CHEN, Min, Jiafu WAN, Sergio GONZALEZ, Xiaofei LIAO a Victor C.M. LEUNG. A Survey of Recent Developments in Home M2M Networks. *IEEE Communications Surveys*. 2014, vol. 16, issue 1, s. 98-114. DOI: 10.1109/SURV.2013.110113.00249. Dostupné z: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6674156`

[7] THE ECLIPSE FOUNDATION. *OSGi Concepts* [online]. 2011 [cit. 2015-11-22]. Dostupné z: `http://bit.ly/1kOyKnc`

[8] FRANCETELECOM. *Modus TR069* [online]. 2011 [cit. 2015-11-22]. Dostupné z: `http://modus-tr-069.sourceforge.net`

[9] FREDJ, Sameh, Mathieu BOUSSARD, Daniel KOFMAN a Ludovic NOIRIE. A scalable IoT service search based on clustering and aggregation. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 2013, č. 1. s. 403-410

[10] GENIEACS.COM. *GenieACS* [online]. 2015 [cit. 2015-11-22]. Dostupné z: `https://genieacs.com`

[11] GRANT ALLEN, Mike Owens. *The definitive guide to SQLite [take control of this compact but powerful tool to embed sophisticated SQL databases within your applications!]*. 2nd ed. New York: Apress, 2010. ISBN 978-143-0232-261

[12] *H2 Database Engine* [online]. 2015 [cit. 2016-02-27]. Dostupné z: `http://www.h2database.com`

[13] HALL, Richard S. *OSGi in action: creating modular applications in Java*. Greenwich [Conn.]: Manning, 2011, xxv, 548 p. ISBN 19-339-8891-6.

[14] *Highcharts* [online]. Norsko: Highcharts, 2016 [cit. 2016-05-01]. Dostupné z: `http://www.highcharts.com/products/highcharts`

[15] HWACI - APPLIED SOFTWARE RESEARCH. *SQLite* [online]. 2015 [cit. 2016-02-27]. Dostupné z: `https://www.sqlite.org/`

[16] *JCWMPServer* [online]. 2013 [cit. 2015-11-25]. Dostupné z: `http://sourceforge.net/projects/jcwmpserver/`

[17] JUNIPER NETWORKS, Inc. *MACHINE-TO-MACHINE (M2M) — THE RISE OF THE MACHINES*. 2011.

[18] KALLEL, Mohamed. *EasyCwmp* [online]. 2015 [cit. 2015-11-22]. Dostupné z: `http://www.easycwmp.org`

[19] THE KNOPFLERFISH PROJECT. *Knopflerfish* [online]. 2015 [cit. 2015-11-22]. Dostupné z: `https://www.knopflerfish.org`

[20] L. SAITO, Taro. *SQLite JDBC Driver* [online]. 2015 [cit. 2016-03-26]. Dostupné z: `https://bitbucket.org/xerial/sqlite-jdbc`

[21] LAN/MAN STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY. *IEEE standard for local and metropolitan area networks* [online]. New York: Institute of Electrical and Electronics Engineers, 2011 [cit. 2015-11-15]. ISBN 978-073-8166-841. Dostupné z: `https://standards.ieee.org/getieee802/download/802.15.4-2011.pdf`

[22] LEGG, Gary. ZigBee: Wireless Technology for Low-Power Sensor Networks. *EETimes* [online]. 2004 [cit. 2015-11-10]. Dostupné z: `http://www.eetimes.com/document.asp?doc_id=1275760`

[23] NAKAMURA, Yuichi, Akira MORIGUCHI a Toshihiro YAMAUCHI. CSDA: Rule-based Complex Sensor Data Aggregation System for M2M Gateway. *2015 Eighth International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*. 2015, č. 1., s. 108-113

[24] ORACLE. *Oracle Berkeley DB 12c* [online]. 2016 [cit. 2016-02-27]. Dostupné z: http://bit.ly/1QtyUKG

[25] OSGI™ ALLIANCE. *OSGi: The Dynamic Module System for Java* [online]. 2015 [cit. 2015-11-22]. Dostupné z: https://www.osgi.org

[26] PERKOV, Luka. *Freecwmp* [online]. 2012 [cit. 2015-11-22]. Dostupné z: http://freecwmp.org

[27] RAIMA INC. *Database Management System Solutions - Raima* [online]. 2016 [cit. 2016-02-27]. Dostupné z: http://raima.com/

[28] STACK, Anis. TR-069 cwmp client implementation: open sources comparison. In: *Stackoverflow* [online]. 2014 [cit. 2015-11-22]. Dostupné z: http://bit.ly/1PTzzWS

[29] ŠTŮSEK, Martin. Performance Analysis of the OSGi-based IoT Frameworks on Restricted Devices as Enablers for Connected-Home. *2015 7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. 2015.

[30] VOJÁČEK, Antonín. Sběrnice Wireless M-BUS - jde to i bezdrátově... *Automatizace.hw.cz* [online]. 2010 [cit. 2015-11-10]. Dostupné z: http://automatizace.hw.cz/sbernice-wireless-mbus-jde-i-bezdratove

[31] WIRELESS SYSTEM LABORATORY OF BRNO. *Wislab* [online]. 2015 [cit. 2015-12-08]. Dostupné z: http://wislab.cz

[32] XING, Congcong. A Novel Trust-Based Secure Data Aggregation for Internet of Things. *The 9th International Conference on Computer Science & Education (ICCSE 2014)*. 2014, č. 1, s. 435-439.

[33] The Zettabyte Era—Trends and Analysis. CISCO SYSTEMS, INC. *Cisco.com: Visual Networking Index* [online]. 2015 [cit. 2015-10-24]. Dostupné z: http://bit.ly/1ojjtoE

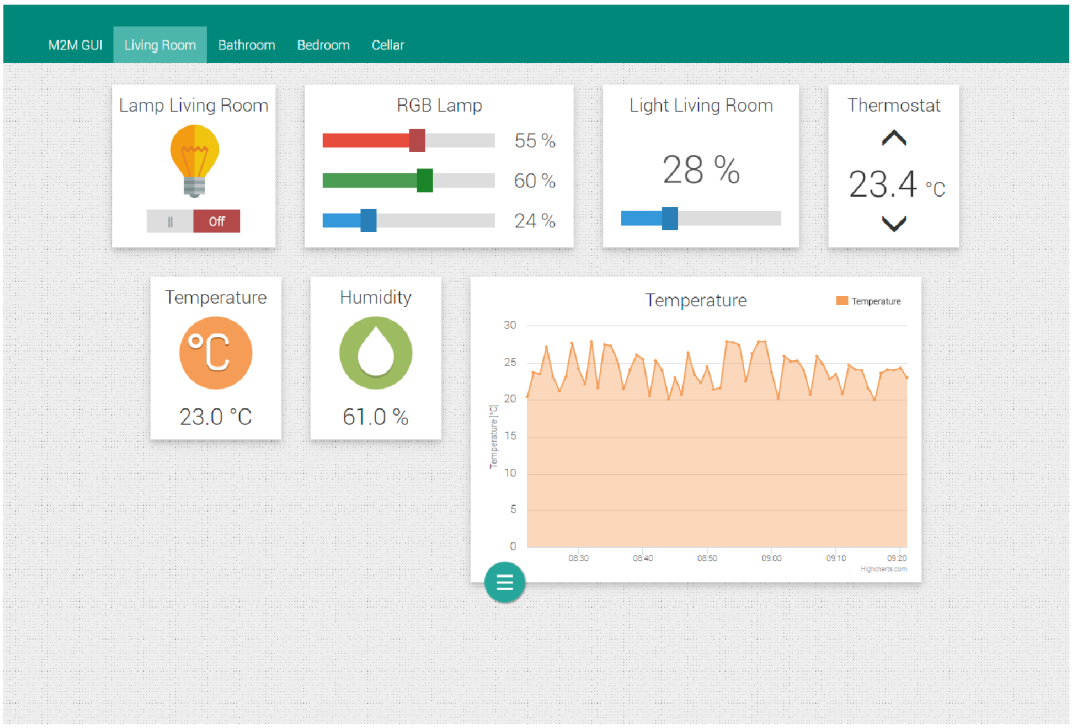
SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ACID	Atomicity Consistency Isolation Durability
ACS	Auto-Configuration Server
AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript and XML
AMR	Acorn RISC Machine
ATA	Analog Telephone Adapter
CSDA	Complex Sensor Data Aggregation
CPE	Customer Premise Equipment
DHCP	Dynamic Host Configuration Protocol
DLNA	Digital Living Network Alliance
DNS	Domain Name System
FTP	File Transfer Protocol
GPLv2	General Public License version 2
HGi	Home Gateway Initiative
HomeRF	Home Radio Frequency
HTML5	HyperText Markup Language 5
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
H2H	Human-to-Human
IP	Internet Protocol
IPv6	Internet Protocol version 6
JNI	Java Native Interface
JVM	Java Virtual Machine
M2M	Machine-to-Machine

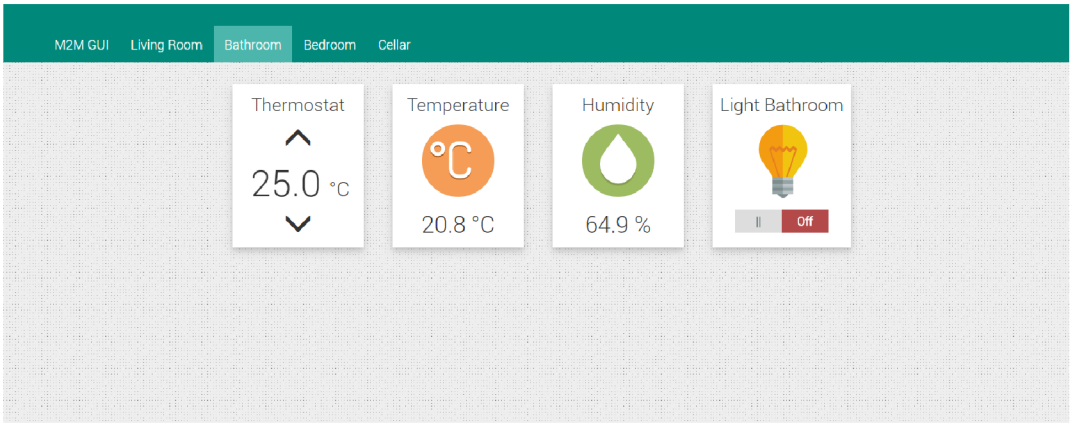
MIPS	Microprocessor without Interlocked Pipeline Stages
MTCG	Machine-type Communication Gateway
NAT	Network Address Translation
opkg	Open PacKaGe Management
OSGi	Open Services Gateway initiative
PAN ID	Personal Area Network ID
POSIX	Portable Operating System Interface
PHP	Hypertext Preprocessor
REST API	Representational State Transfer Application Programming Interface
RPC	Remote Procedures Calling
RSSI	Received Signal Strength Indicator
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
SQL	Structured Query Language
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
UWB	Ultra-Wideband
VDBE	Virtual Database Engine
VoIP	Voice over IP
XML	Extensible Markup Language

SEZNAM PŘÍLOH

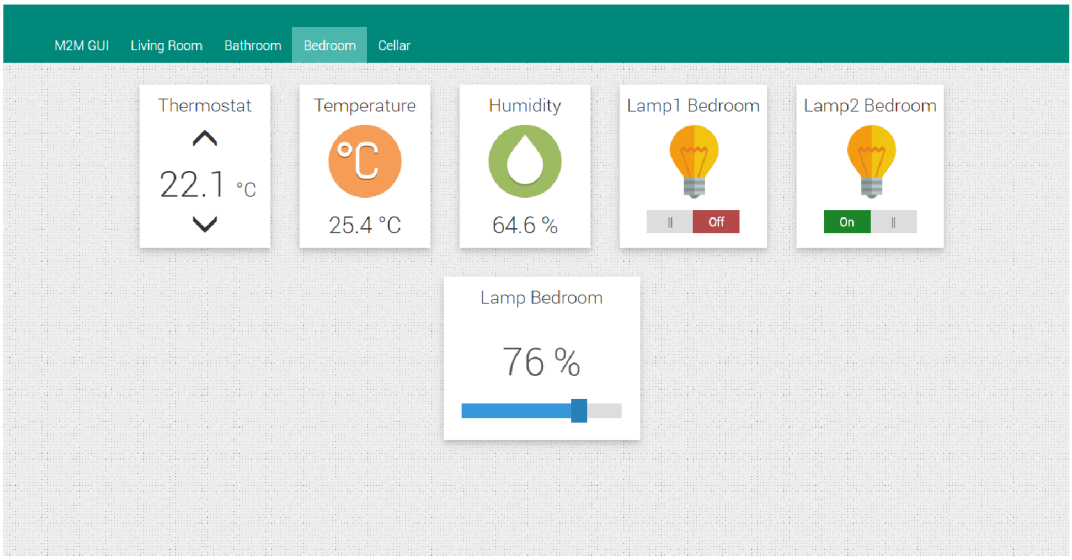
A UKÁZKY UŽIVATELSKÉHO ROZHRANÍ



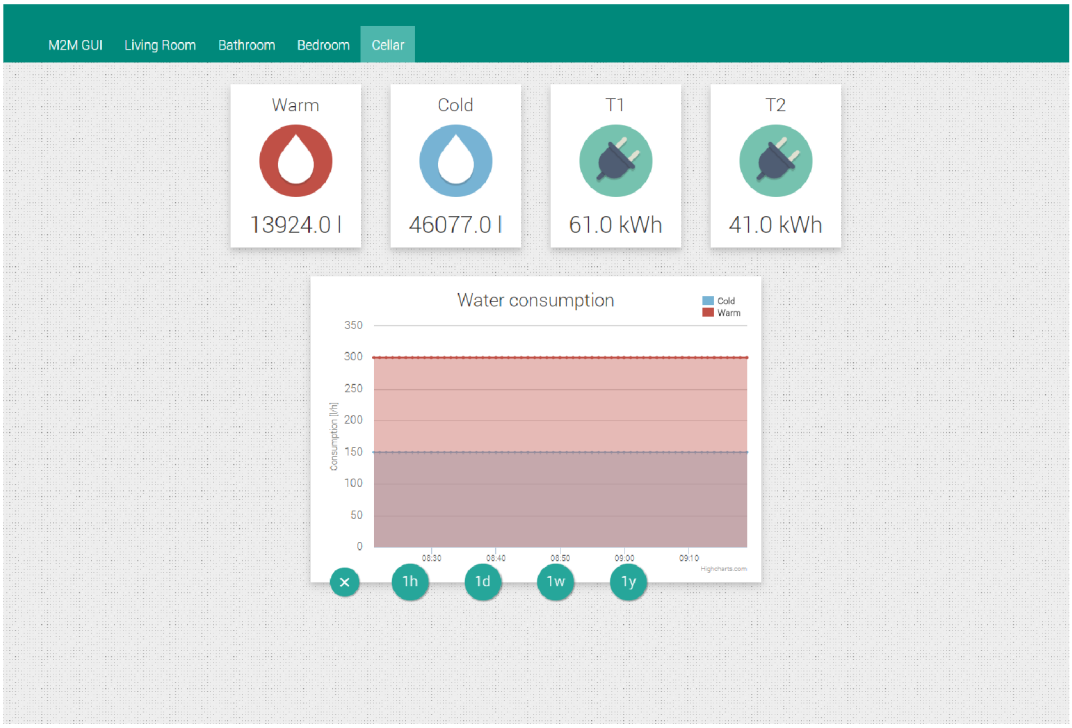
Obr. A.1: Uživatelské rozhraní – Living Room.



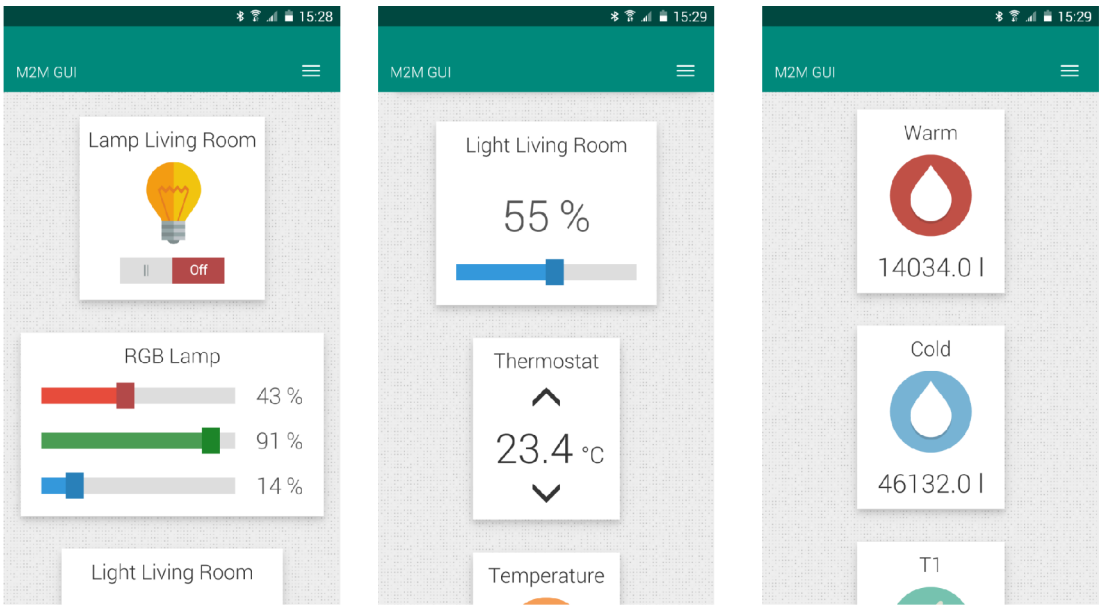
Obr. A.2: Uživatelské rozhraní – Bathroom.



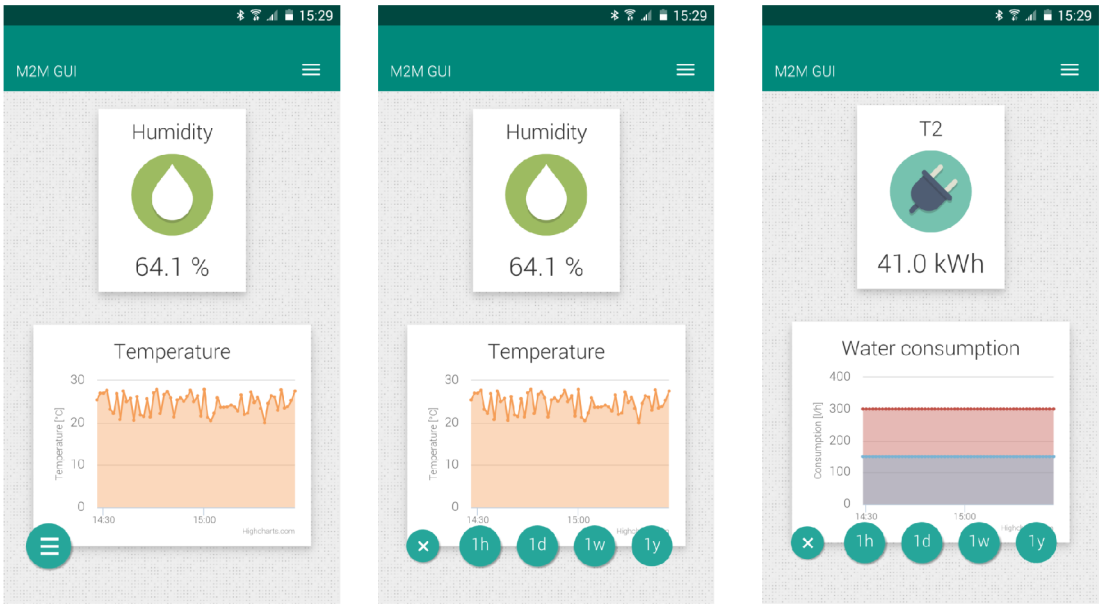
Obr. A.3: Uživatelské rozhraní – Bedroom.



Obr. A.4: Uživatelské rozhraní – Cellar.



Obr. A.5: Uživatelské rozhraní na mobilním telefonu.



Obr. A.6: Zobrazení grafů na mobilním telefonu.